DISS. ETH NO. 17836

A Distributed Cognitive Map for Spatial Navigation Based on Graphically Organized Place Agents

A dissertation submitted to the

SWISS FEDERAL INSTITUTE OF TECHNOLOGY (ETH) ZURICH

for the degree of

DOCTOR OF NATURAL SCIENCES

presented by

JÖRG-ALEXANDER CONRADT

Dipl.-Ing., Technische Universität Berlin, Germany
MS.-CS., University of Southern California, CA, USA

born 25. December 1974

citizen of Germany

accepted on the recommendation of

Prof. Dr. Rodney J. Douglas
Prof. Dr. Rolf Pfeifer
Prof. Dr. Rüdiger Wehner

2008

A Distributed Cognitive Map for Spatial Navigation Based on Graphically Organized Place Agents

- Abstract -

Jörg Conradt
Institute of Neuroinformatics, UZH/ETH-Zürich

Animals quickly acquire spatial knowledge and are thereby able to navigate over very large regions. These natural methods dramatically outperform current algorithms for robotic navigation, which are either limited to small regions (Arleo 2000) or require huge computational resources to maintain a globally consistent spatial map (Thrun 2002). We have now developed a novel system for mobile robotic navigation that like its biological counterpart decomposes explored space into a distributed graphical network of behaviorally significant places. Each such a place is represented by an independent "place agent" (PA) that actively maintains the spatial and behavioral knowledge relevant for navigation in that place.

The collection of such place agents does not represent space in a common consistent data structure as current topological or metric-based approaches do (Figure A1, left and middle). Instead, our system incrementally builds a graphical network consisting of PAs that each only knows its local space and its local nearest neighbors. Each of these PAs is unaware of its position within the network and the position it represents in global space. The topology of the network - which reflects the structure of traversable external space - only exists implicitly, as PAs only communicate with their direct local neighbors (Figure A1, right). No process in our system maintains or operates globally on the network; thus, it is only necessary to maintain spatial consistency locally within the graph.

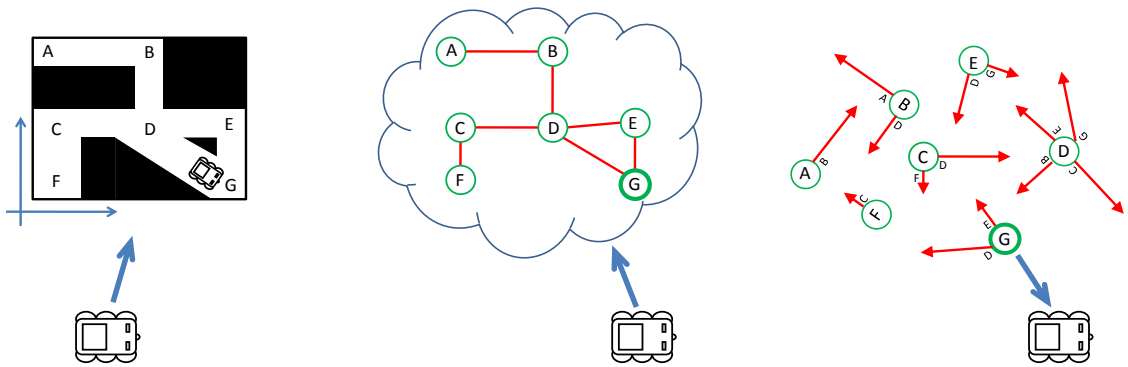


Figure A1: Left: a global Cartesian map (top-down view), maintained by a mobile robot. Middle: a globally consistent topological representation of the same environment maintained by a robot. Right: A collection of individual programs as active place-representations, each only knowing its respective local space and direct neighbors. The true graphical structure exists only implicitly.

This simple strategy significantly reduces computational complexity, is robust to local perturbations, scales well with the size of the navigable region, and permits a robot to autonomously explore, learn, and navigate large unknown environments in real time. The system has explored a floor in our institute of 60x23m using a mobile robot, and created about 150 independent PAs to represent behaviorally relevant spaces. The resulting distributed network can demonstrate globally consistent navigation behavior without a global supervisor involved, e.g. guiding the robot to previously

encountered stimuli. We expect that we can represent significantly larger environments without suffering degraded performance, as all PAs are independent programs that we can distribute among multiple computing units.

Neural hardware is well suited to implement such a “distributed cognitive map”, as brains are intrinsically distributed processing systems without global access to all memorized information - which is required by traditional algorithms for navigation. We therefore are convinced that the proposed system resembles biological information processing much more closely than current methods for spatial navigation.

Table of Content

1 Introduction

1.1 Moving in Space

"Where am I?"

... having to think about this question often surprises people, so much are we used to "knowing" where we are. Only under exceptional circumstances, visiting a foreign town or a large shopping mall, we might temporarily be confused and feel uncomfortable - but typically even then we are not completely unaware of our location. When asked, people often describe their current position with respect to behavioral relevance: "at home", "in the bus", etc. - and equally often reply with abstract concepts, such as "on the way", or "on holiday". We do not typically think of our current spatial location as coordinates relative to a fixed origin, such as the earth's latitude and longitude, or a street name and house number, but rather link our interpretation of our current position to the behavioral relevance of places.

Not just our current position, but also our future position is relevant: people move in space all the time; and purposeful motion - i.e. goal directed - is arguably amongst the most important requirements for human survival. Whether fetching a drink at night from the kitchen or hunting pray at their preferred feeding sites: we need to return to those places that we earlier learned are crucial for survival. So the initial question "where am I?" can well get extended to "how do I get from here to there?" - which implies a mental representation of "there". How do we describe "there"? Although people for long time have spend unbelievable effort creating drawings to represent space (Harwood 2006), such Cartesian maps do not reflect how we typically think of "there". We would rather describe our notion of "there" as a sequence of actions to be taken to get there, or refer to related sites that our communication partner knows. Reaching "there" probably includes hundreds of complex choices on different levels, each step, each direction, whether to walk, bike or take a train, etc.

We move in space, all the time. Spatial behavior is central to everyday live. It is tightly coupled with our perception and cognition of space, action, and behavior. How we do that is a research problem routed in various different areas, such as psychology, geography, architecture, computer science, cognitive science, biology, zoology.

How do we know where we are? How do we know where we want to go? And how do we get there?

1.2 Problem Statement - Cognitive Mapping

Animals face similar problems when moving in space: they need to return to places they have visited before (Gould 2004). A large range of animal-environment interactions can be explained by simple stimulus-response strategies - such as photo-taxis, path following, or snapshot-based homing - and have often been implemented in robots (Consi and Webb 2001). Some particular behaviors, such as animals' impressive ability to remember locations (Olton and Samuelson 1976), to navigate accurately towards a hidden goal by novel routes (Morris 1981), to take shortcuts (Chapuis, Durup et al. 1987) and detours (Tolman and Honzik 1930) although cannot get explained solely on such

“primitive principles”. Whenever simple explanations fail, people quickly postulate the use of a mental cognitive map, which provides some form of spatial representation, to solve such tasks.

But what is a cognitive map? - and how is it implemented?

The term "cognitive map" is used widely, but in slightly varying conceptions. In general it refers to any form of mental representation of a global environment. (Tolman 1948) originated the term cognitive map, proposing that animals (rats) do not learn space as a collection of independent paths from a starting place to a target (a sequence of movements), but instead develop an internal abstract representation of the structure of external space; i.e. create a representation for the relative geometric arrangement of features in space. (Downs and Stea 1973) refers to cognitive mapping as “a process composed of a series of psychological transformations by which an individual acquires, stores, recalls, and decodes information about the relative locations and attributes of the phenomena in his everyday spatial environment”, whereas (Todd 1997) describes a cognitive map as "an internal representation of an environment gained by a comprehensive set of observations, and is used to travel between locations in the environment." Various other definitions - yet similar in meaning - can be found in (Lagoudakis 1998). The term cognitive map therefore shall not be taken as an exact description of an internal spatial representation, but rather refer to any internal representation that might be composed of a number of things that facility representing and acting in an external environment (Downs 1981). For more detailed discussions about cognitive maps refer to (Kitchin 1994; Bennett 1996; Jacobs 2003).

Now we have some understanding of what people refer to when they mention a "cognitive map": a mental way of representing external space, applied to solve navigation problems. As we will see in the following chapters, ideas and concepts how such a representation might be implemented in brains - or on robots - diverge substantially.

Looking at robotic models and methods to represent space, people generally agree that generating such a map of an unknown environment and localizing the robot on that map currently being built is a chicken-and-egg problem (Thrun 2002): that is a problem that would be much easier if one of the two aspects was already solved. Imagine a robot had a perfect map representing its environment - then finding the robot's position based on current sensor readings seems possible. In contrast, if the robot had a perfect estimate of its Cartesian position with respect to a global reference - then building a map based on current sensor perceptions seems possible. What turns the problem so difficult is that build-up and localization need to happen at the same time, based on incomplete and unreliable information provided by the other. The problem turns significantly simpler if we allow infinite computational resources and super precise sensors; in fact in such a scenario this problem in principle is solved, as presented in chapter 1.4.1.

But animals and humans (and most robots) do not have infinite resources; yet the first perform impressively well and vastly outperform current robotic methods for navigation within large areas. Brains do not have the power to store all acquired information, possibly re-inspect all previously recorded sensory stimuli, and finally compute a global consistent map. It seems they have to use a very different approach from current technological methods to represent a cognitive map. How do brains do that? In this thesis we are investigating a biologically plausible principle for acquiring, storing, maintaining and ultimately using spatial knowledge for short and long-range navigation between behaviorally significant places: a distributed cognitive map.

1.3 Introduction to Distributed Cognitive Mapping

We briefly present current models for navigation in engineering and in biology, which allows up to outline the innovative aspects in our model afterwards. For a more detailed discussion of current models refer to chapter 1.4.

Navigation in Engineering and Neuroscience

In today's engineered navigation systems (Thrun 2002), an active agent such as a robot typically stores all acquired information about its spatial environment in a global map relative to an absolute origin (Bosse, Newman et al. 2003; Montemerlo and Thrun 2007), shown in Figure 1, left. Adding or updating information and using stored knowledge for navigation typically require considerable computational resources and involve a single active agent - such as a computer program - having access to all accumulated information. Alternatively, several topological approaches for navigation exist (Shatkay and Kaelbling 1997; Tapus 2005), shown in Figure 1, middle, and also hybrid topologic-metric approaches (Kuipers 2000; Tomatis, Nourbakhsh et al. 2001; Jefferies, Baker et al. 2008). But all these approaches rely on a single active agent that is reasoning based on all previously acquired data. Such a mechanism is unlikely to be performed in animal or human brains: there does not appear to be any one active region of our brain "inspecting" other passive regions to use stored information for navigation.

In the last few decades place cells and - much more recently - grid cells have been in the center of neuroscientific research about navigation (O'Keefe and Nadel 1978; Redish 1999) and its robotic implementation (Arleo 2000; Hafner 2008; Milford 2008). These families of cells in Hippocampus (O'Keefe and Nadel 1978) and Entorhinal Cortex (Sargolini, Fyhn et al. 2006) show significantly increased activity whenever an animal happens to be within a well defined region in an experimental setup. It is widely agreed that activity of a collection of such place cells represents the animal's current spatial position within a local frame of reference. These cells show a stable firing pattern over a long time within an environment, perform a complete remapping of their firing pattern when the observed animal enters a distinct new environment (Markus, Qin et al. 1995), and revert to the previous firing pattern upon returning to a familiar environment. However, experiments so far are constrained to relatively small areas of about 2m in diameter, whereas wild animals typically live in areas of several thousands of square meters. It is yet to be investigated how navigation in larger areas impact firing patterns in place cells.

A Distributed Cognitive Map

In this thesis we explore a new principle of perceiving, maintaining and operating on spatial knowledge: every element of the system operates exclusively on locally available information and is constrained to decide actions based only on knowledge about its vicinity. We do not represent space in a global consistent data structure as traditional topological or metric-based approaches do. Instead, starting without prior spatial knowledge, our system autonomously creates a graphical network of independent "patches of knowledge" at behaviorally relevant places. Such patches contain spatial and behavioral information only about their local environment. They actively control the physical agent - a robot - in their vicinity, and maintain and update their limited knowledge about their environment, instead of serving as passive data storage containers. As these active patches know about and communicate with their local nearest neighbors, they implicitly establish the topology of a network that represents behaviorally significant space. Each of these nodes of the network is unaware of its position within the network and the position it represents in global space.

No process in our system maintains or operates globally on the network; in fact, the whole network only exists because of message-passing between autonomously acting neighboring nodes. This principle is illustrated in Figure 1, right.

Note that the distributed system does not explicitly represent cycles in the environment. Figure 1, left and middle, show a cycle in the environment between places D-E-G-D, which is computationally expensive to handle (Thrun 2002). In the distributed representation, no actor has access to D, E, and G simultaneously. The system is unaware of the cycle, but still knows all individual traversable paths that together constitute the cycle.

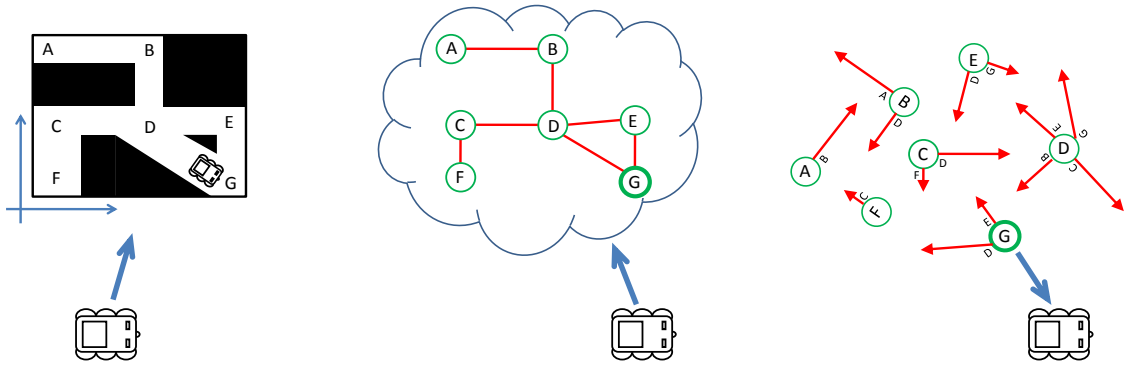


Figure 1: left: a global Cartesian map showing a top-down view of an environment, maintained by a single computer or a mobile robot. Middle: topological representation of the same environment recorded as a consistent data structure. A computer program/robot maintains this structure and operates based on stored information. Right: A collection of individual programs as active place-representations, each only knowing their respective local space and direct neighbors. The true graphical structure exists only implicitly in the collection of all places, to which no actor has access. At any time only a single one of these individual programs interacts with the robot; i.e. it directs the robot and obtains sensed information about the current local environment.

Implementation and Results

The system we developed starts without spatial knowledge in an unfamiliar environment which it automatically explores using a mobile robot. While exploring, the robot constantly maintains a representation of its current local environment[1] which consists of fused information from various on-board sensors. When detecting a behaviorally relevant stimulus[2], a new place agent captures a snapshot of the current local environment, records the previously active place agent as a neighbor, and directs the robot to further explore unknown regions. Repeating this process incrementally builds up a collection of independent place agents. All these agents only know their local spatial environment and communicate with their direct neighbors, as shown in Figure 1, right. Despite these severe constraints on knowledge and communication, the collection of all individual place agents shows emergent globally consistent behavior without having a single globally acting entity in the system. An example of such global behavior is the guidance of the robot to a previously recorded target represented elsewhere in the network.

Our system has not only successfully explored spaces as small as a single room, but also our whole institute of about 60x23m (Figure 2). Only few operations require time scaling linearly with the

[1] "Current local environment" refers to space within a few times the robot's body-length, typically within sensor reach. The overall operating area of the robot, in contrast, covers several 100 times the robot's body-length.

[2] Such as an intersection of paths, a battery charger or fresh coffee, depending on one's preference.

number of nodes in the network; most operations are performed on local data only. Currently, all place agents required to represent our institute run in real-time on a single computer; but as they are independent of each other they could be distributed among multiple possibly less powerful computing units. We are convinced that we can map significantly larger environments without suffering degraded performance.

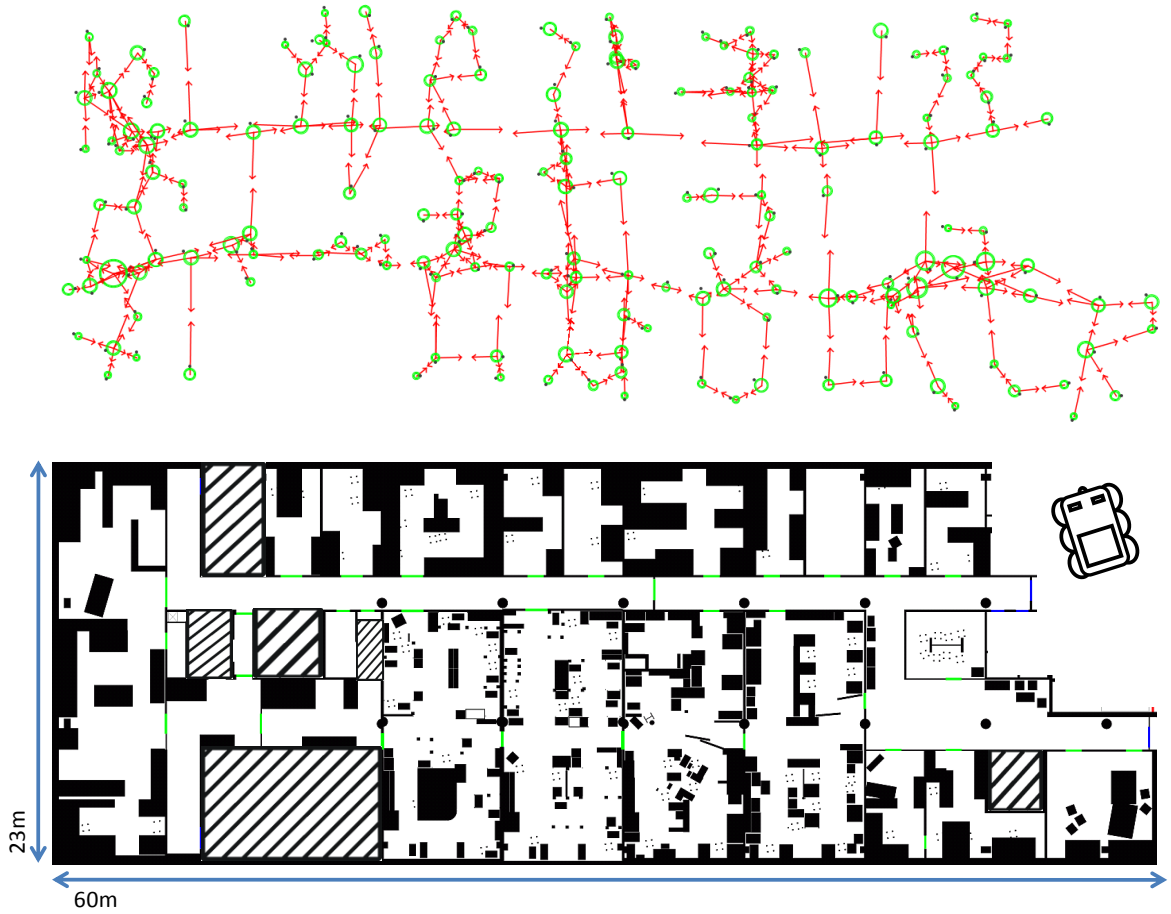


Figure 2: Top: A distributed graphical representation of our institute established by 177 independent place agents (PAs, green circles). The spatial arrangement of PAs only happens for this display; no actor in the system is aware of the structure. Red arrows show 45% of the recorded distance to a neighboring PA. Places of behavioral relevance are covered densely with nodes, whereas places such as long aisles show sparse coverage. Bottom: a plan-view of the institute (60x23meters).

We believe that neural hardware is well suited to implement such an "active distributed cognitive map", as brains are intrinsically distributed processing systems without global access to all memorized information - which is required by traditional algorithms for navigation. We therefore believe that the proposed system resembles biological information processing much more closely than current methods for long-range spatial navigation.

Relating our system to place-field neurons found in Hippocampus (O'Keefe and Nadel 1978; Redish 1999), we interpret observed stable activity-patterns of place-cells within a small experimental setup as an indicator for a short-range local map in animals brains. Such a local map is implemented by a single place-agent in our system. When an observed animal changes to a different place, a complete

remapping of firing patterns across place-field neurons occurs (Markus, Qin et al. 1995), which corresponds to transferring behavioral control to a new place agent in our model.

Our system explains results from typical behavioral experiments with rats (Tolman 1948; Blancheteau and Lorec 1972), without requiring a global actor such as a computer program or an omni-conscious homunculus having access to all acquired information.

1.4 Review of Alternative Approaches

This brief review points to research about localization and mapping, in areas such diverse as robotics, biology, psychology, architecture, and urban-planning. The main focus in all these areas is common: understanding how a mobile agent - animal, human or robot - is capable of moving intentionally in an artificial or natural environment. However, the particular focus within these areas differs substantially. Robotics on one extreme is mainly concerned about creating maps that reflect existing environments as adequately as possible to determine a robot's current position in such a map (Adams 2007; Patnaik 2007; Thrun 2008), while simultaneously reducing the computational complexity of algorithms that have been shown to create an optimal representation given limited sensor precision (Thrun, Burgard et al. 2005; Montemerlo and Thrun 2007). The question how to exhibit purposeful behavior with such spatial knowledge is typically completely decoupled from the process of building and maintaining a map. Biology research, in contrast, typically tests animals' navigation capabilities, describes observed principles, and models such behavior (Healy 1998; Burgess, Jeffery et al. 1999; Golledge 1999). Cognitive Science and/or urban-planning typically investigates in how humans perceive an environment and act within such (Lynch 1960; Kitchin and Freundshuh 2000; Allen 2006). The aforementioned references are pointing to summarizing review papers and books on the subject, whereas the following subchapters point in more detail to individual research.

1.4.1 Robotic Implementations

"Knowing where I am" is arguably the most important issue in mobile robotics. Hence a lot of research has focused on the problem of localizing a robot with respect to a spatial representation and simultaneously building such a spatial representation. However, as we will see below, the main line of research in robotics has a very different focus compared to our understanding of perceiving and representing space: all such examples create precise duplicates of the underlying structure of the environment in some form - such that a robot can use this acquired spatial knowledge for navigation.

(Leonard and Durrant-Whyte 1992) introduced for the first time the concept of SLAM (Simultaneous Localization and Mapping) as incrementally constructing maps while a mobile agent explores an environment - and at the same time localizes itself with respect to the partially built representation. Most work in this area uses global metric maps, such as stochastic map techniques (Leonard and Durrant-Whyte 1992; Castellanos and Tardos 2000; Dissanayake, Newman et al. 2002) and occupancy grid approaches (Thrun 1998). More recent research allows purely vision-based metric approaches (Se, Lowe et al. 2002). However, such SLAM methods quickly become computationally expensive for large environments (Thrun 2002). To address this problem, (Thrun 2000; Montemerlo and Thrun 2007; Thrun 2008) propose probabilistic methods that increase speed and robustness in SLAM. All these approaches provide a fine-grained abstract representation of an environment, which

is well suited for self-localization when given a new sensory signature from within this environment. However, they are poorly suited for higher level reasoning about space, such as representing targets, finding paths to targets, etc.

Topological approaches to SLAM attempt to overcome such drawbacks by modeling spatial structure using graphs. Kuipers (Kuipers 1977; Kuipers 1978; Kuipers 1983) was amongst the first to present a model of space decomposed into individual places that get connected in a topological fashion representing the places' connectivity; later (Kortenkamp and Weymouth 1994; Gutierrez-Osuna and Luo 1996; Shatkay and Kaelbling 1997; Fox 1998) have presented different approaches for topological navigation. All such approaches define the concept of gateways, which mark the transition between two directly adjacent places in the environment. Because topologic approaches break metric space in topological nodes, these approaches generally are less precise than fully metric SLAM - or end up computationally intractable when fine-grained places are used. Furthermore, many implementations suffer from perceptual aliasing, as observations taken at various different locations look similar.

(Mallot, Bülthoff et al. 1995; Schölkopf and Mallot 1995; Franz, Schölkopf et al. 1998) presented a model designed to explore open areas connected in a maze-like structure, and to build graphical representations that rely on local view-based homing methods (Vardy and Möller 2005), which compare a current view against a stored view at a target location. (Lambrinos, Möller et al. 1999; Möller, Lambrinos et al. 2001) introduced average landmark vectors for local visual homing, whereas recently (Smith, Philippides et al. 2007) proposed a linked sequence of waypoints - each providing a local homing target - to allow navigation over larger areas. These methods capture a new snapshot whenever a significant change happens in the environment, e.g. a similarity measure falls below a threshold or the number of perceived discrete landmarks changed.

A small number of recent approaches started to combine topological and metric mapping, the two most advances approaches are briefly presented here:

(Bosse, Newman et al. 2004; Newman, Leonard et al. 2005) present the "Atlas Framework" that uses local metric maps (SLAM techniques) in their own coordinate frame, but arranges these in a global network where nodes correspond to a local metric map and edges to the transitions between these maps. Each local map represents a large region in the environment that is limited by processing complexity rather than size, allowing a fixed upper limit for computing requirements. This typically enforces a hard limit on the number of features in a particular map, such that currently sensed features might fall into a "neighboring" map - which by definition is directly adjacent to the current map to represent all space. One or multiple such local maps are active at a time. Cycle closing happens by a global search mechanism that determines overlap of a map with parts of one or multiple other maps, increasing matching certainty when multiple matches are found. This approach is similar to our understanding of creating a topological structure of metric places; however, the Atlas framework still maps all space equally independent of behavioral significance and requires global maintenance of all data.

In (Tomatis, Nourbakhsh et al. 2001; Tomatis, Nourbakhsh et al. 2003), Tomatis et al. develop another hybrid representation of a global topological map with local metric maps associated to each node. Tapus, Siegwart et al (Tapus 2005; Tapus, Battaglia et al. 2006; Tapus and Siegwart 2006; Tapus and Siegwart 2006) extend this model, such that local places get represented by "fingerprints"

of an environment. A fingerprint here denotes a circular list of features visually present at that place, which is matched against the robot's current perception. Loop closing is induced by a global supervisor detecting congruent motion of multiple peaks in the overall position hypothesis. Although this model allows topological maps without global metric consistency, the approach requires a single supervisor accessing all data, trying to frame the representation in a consistent global structure.

1.4.2 Observations and Models in Psychology or Zoology

How animals move in their environment has been subject of a large body of research investigations grounded in areas around biology or psychology, with "animal" here ranging all the way from insects like desert ants and honeybees over vertebrates such as rats, mice or hamsters up to mammals including humans. The spatial environments under considerations in such experiments are similarly wide-spread, ranging from completely devastated unlabeled deserts over open-field studies with little structure, up to studies in human made environments such as mazes for rats or cities for human experiments. Such studies typically provide two important outcomes: 1) insight in what animals can do in space, and 2) models of how such behavior might work. The later range from pure theoretical analysis (Abbott 1994; Eichenbaum, Dudchenko et al. 1999; Hahnloser, Xie et al. 2002) to validation or testing through software models or in simulated environments (Borenstein 1994; Touretzky, Wan et al. 1994; Balakrishnan, Bousquet et al. 1998; Stringer, Rolls et al. 2002; Koene, Gorchetchnikov et al. 2003), up to implementing models on real robotic platforms (Lambrinos, Möller et al. 1999; Arleo 2000; Hafner 2000; Möller, Lambrinos et al. 2001; Krichmar, Nitz et al. 2005).

Animals in such studies show a vastly different repertoire of spatial awareness and of behaviors performed in such environments. The following pointers to research projects provide an overview of experiments, which - taken together - lead to conclude that up to now it is unclear for some animals whether they possess some form of a cognitive map, or if their exhibited behavior can get explained solely by simpler navigation mechanisms, such as homing and path-integration.

Ants and bees forage in areas several 1000 times their body-length searching for food and returning to their nests or hives, respectively. They are known to rely on such diverse sensory input as path integration (Müller and Wehner 1988), optic flow (Srinivasan, Zhang et al. 2000), polarized light (Fent 1986), and distal and local landmarks (Collett, Graham et al. 2007). One school of thought suggests that honeybees indeed possess and apply general landmark memory in a "map-like" fashion (Menzel, Brandt et al. 2000; Menzel, Greggers et al. 2005), whereas ants were shown to generally rely on a homing vector (Müller and Wehner 1988) that - in cases of strong mismatch - can get corrected by distal visual landmarks (Collett, Collett et al. 2001). A recent study has clearly demonstrated that ants operate on one-way routes rather than maps (Knaden, Lange et al. 2006; Wehner, Boyer et al. 2006).

A large number of behavioral experiments have been performed on rodents, since Tolman in 1948 suggested that rats have the ability to form a mental cognitive map of their environment in (Tolman and Honzik 1930; Tolman, Ritchie et al. 1946; Tolman 1948), and received another boost with Morris' famous experiments in water mazes (Morris 1981). The basic similarity amongst all such experiments is demonstrating that rats learn the structure of their environment without receiving a significant reward during exploration: typically, a rat explores an arena that contains a feeding side which initially is not particularly attractive for the rat, as it is well fed before exploration. During testing -

when deprived of food - they show the ability to run directly to the feeding side, possibly using shortcuts they have never used before or detours, if required. Tolman argues that this is clear evidence for a cognitive map. We will revert to such experiments in chapter 7.5. Other research, in contrast, claims to find no evidence for cognitive maps in rodents (Benhamou 1996; Olthof, Sutton et al. 1999; Mackintosh 2002).

Several studies (Dyer 1991; Gallistel 1993) have cited an experiment by (Menzel 1973) in which chimpanzees took shortcuts to hidden food as evidence for cognitive mapping in chimpanzees; however, (Bennett 1996) disputed that conclusion arguing that recognizing snapshots of landmark constellations suffices for solving this task.

Finally, (Gillner and Mallot 1998; Mallot, Gillner et al. 1998) performed studies testing human navigation ability in a virtual environment, suggesting that humans acquire a graph-like representation of the virtual environment that connects possibly inconsistent local patches (places or views) which do not necessarily combine into a metric survey map. They do not present a model that explains human's ability to generate such representations.

1.4.3 Discoveries and Models in Neuroscience

The discovery of place cells (O'Keefe and Dostrovsky 1971; O'Keefe and Nadel 1978) and shortly after head-direction cells (Ranck 1984; Taube, Muller et al. 1990) started a vast amount of research investigating in these cells' representations of space; for reviews see (Muller 1996 ; Leutgeb, Leutgeb et al. 2005; McNaughton, Battaglia et al. 2006). Place cells show a unique firing pattern whenever the animal under investigation happens to be at a particular position in space; whereas head-direction cells show increased activity whenever an animal's head points in a particular direction. The ongoing debate about such cells discusses whether they simply represent the animal's assumption about its own position in space, or if they are the substrate of an internal cognitive map. A book by Redish (Redish 1999) suggest a common use of the Hippocampus for acquiring spatial but also episodic memory, whereas a recent study shows physiological evidence that navigation and event memory are interlinked in the Hippocampus (Foster and Wilson 2006). The study by (Ekstrom, Kahana et al. 2003) reports such place cell activity also in human Hippocampus.

A large number of studies investigate in the Hippocampus' ability to correct errors in place cell firing, typically by providing conflicting reference frames during a rat's motion, such as an offset between path integration and external landmarks, e.g. (Gothard, Skaggs et al. 1996; Rosenzweig, Redish et al. 2003). Several other studies (Bures, Fenton et al. 1997; Cressant, Muller et al. 2002) test rats in mazes that have been misaligned against external landmarks. All such studies suggest that place cells get reset by distal external landmarks.

Moser and colleagues reported position related activity not only in the Hippocampus, but also in Entorhinal Cortex (Fyhn, Molden et al. 2004; Hafting, Fyhn et al. 2005; Sargolini, Fyhn et al. 2006; Fyhn, Hafting et al. 2007). Cells recorded showed the same properties as hippocampal place cells, but fired in a regular multi-peaked (grid) fashion instead of at a single spatial peak, suggesting that such grid cells support internal path-integration. A recent study (Hok, Lenck-Santini et al. 2007) proposes that place cells not only code for spatial position, but also for reward expectation at a given target position, suggesting that place cells are already involved in path planning.

Various models of navigation based on place cells acquire a pool of neurons that 'learn' to behave like place cells (Samsonovich and McNaughton 1997; Kulvicius, Tamosiunaite et al. 2007), a small number of such implemented in real robots (Arleo and Gerstner 2000; Arleo 2005; Chokshi, Wermter et al. 2005). Poucet presented a model (Poucet 1993) in which place cells code for local places within a large environment, all of which the model orients to a common reference and connects in a graph-like structure to form a globally consistent representation of the rats' environment.

Gaussier at al. (Gaussier, Joulain et al. 2000; Cuperlier, Quoy et al. 2007; Gaussier, Banquet et al. 2007) implement a navigation strategy based on place cells, transition cells and motor cells. In their model, place cells cover the available open space for navigation densely, allowing a mobile robot to localize itself within one of such places based on the constellation of visual landmarks and an on-board compass. Transition cells in contrast represent the set of all possible transitions between neighboring place cells, which generates stimuli for motor-transition cells to move the robot to an adjacent place. All such cells cover all available space uniformly, without an explicit relation between behavioral significance and spatial position. The cells in their model are implemented as passive containers representing environmental information, upon which an externally run algorithm performs basic computations such as diffusion to find a path to a distant target; but these cells do not reason about the spatial arrangement, e.g. they cannot find shortcuts or close loops in the environment themselves.

A different line of research applies MRI techniques to study structural modifications in human Hippocampus related to navigation experience (Maguire, Gadian et al. 2000; Maguire, Woollett et al. 2006). They find that extensive training in spatial navigation (observed in London Taxi Drivers) significantly increases posterior Hippocampus regions. A recent study (Spiers and Maguire 2007) extending this direction of research reports that activity in the medial prefrontal cortex was positively correlated with goal proximity, suggesting that such brain regions are involved in navigation guidance.

1.4.4 Literature Summary

The different approaches reviewed above investigate spatial perception and representation at very different levels, ranging from behavioral observations to mathematically proven optimally correct spatial representations. They all describe a mobile agent's spatial behavior and present models that suit their respective focus, and thus differ substantially in their level of complexity, abstractedness, resource-management, etc. Several models combine behaviors - or at least elementary actions - with spatial perception, but yet only limited to local spaces. All models reviewed above share a global supervisor - i.e. a single omnipresent instance that inspects all acquired spatial knowledge and operates upon that. No approach has yet presented a model that operates on distributed locally limited knowledge only, still achieving globally consistent spatial behavior. Two recent books by (Jefferies and Yeap 2008) and (Milford 2008) - that both explicitly argue about using "nature" as a design guideline - present various models that all rely on a single computing instance with access to all data. But in contrast to today's computers, human and animal brains do not consist of a data area that is accessed by a processing unit - still we are able to perform all such spatial behavior that is currently only explained by global access.

1.5 Thesis Outline

In this thesis we demonstrate that a collection of independent active agents - each constrained to operate on locally limited knowledge - can exhibit globally consistent navigation behavior. Such a system without a global supervisor and without a global reference frame resembles distributed information processing in brains much closer than existing models of long-range navigation.

What we want to achieve in this thesis

- autonomously build-up a representation of possibly large environments, starting without initial spatial knowledge
- maintain acquired spatial knowledge in an active distributed system that controls the robot, instead of in a single large passive data structure that is accessed by a program controlling the robot
- connect behavioral significance of places to the distributed spatial representation, instead of representing all space with equal relevance in a regular grid fashion
- demonstrate that the system performs well in large real-world environments without suffering performance in large areas, as current robot mapping concepts do

What we have done

- developed a model that represents space as a collection of independent "Place Agents" (PA), which each act autonomously only based on its available local data in its local coordinate frame
- build a mobile robot platform to explore large areas (one floor of the whole Institute of Neuroinformatics, of about 1400m^2)
- designed on-board processing (Linux PC) and various sensors that include stereo pan-tilt cameras
- designed and implemented a data structure for local environments that extracts behaviorally relevant information from an arbitrary number and type of sensors
- implemented the model as distributed Java programs (independent threads) that control a mobile robot during initial exploration and later navigation phases
- performed various autonomous exploration experiments of large cyclic environments, both in simulation and in real-world settings
- characterized the system in terms of performance, reliability and efficiency
- discussed ideas for future experiments, comparing our model against human/animal behavior

Structure of Ph.D. thesis:

ch. 1 Introduces the research problem, presents a brief overview of our suggested solution, and points to several related works.

ch. 2 Presents a data structure **("Place Signature")** that is well suited to represent local places of an environment, handles a stream of continuously incoming data from various sensors, yet staying upper-bounded in memory requirements. The "place signature" inspects raw local data and provides unified abstract behaviorally relevant information, such as "open space" or a distance metric between multiple place signatures.

ch. 3 Introduces our customization and additions to the **mobile robot "Koala"**, explains on-board sensors and on-board computation, as well as local targeting and obstacle avoidance mechanisms implemented to decouple the distributed navigation system from low-level reactive robot control.

ch. 4 Demonstrates a **single Place Agent (PA) as active local representation of a behaviorally relevant place** in a simple environment. Our system starts with a single PA that creates copies of itself to explore initially unknown space. Repeating such duplication and exploration cycles incrementally builds up distributed spatial knowledge, maintained in multiple independent PAs.

ch. 5 Discusses a collection of individual PAs that together maintain **a distributed graphical representation of large environments**. Each single PA only stores and acts on individual local knowledge represented in its local coordinate reference; no global supervisor exists in the system. Although the combined spatial representation might not be globally consistent in a metric sense, all PAs contribute to exhibiting globally consistent spatial behavior based on their local knowledge.

ch. 6 **Presents and discusses results** obtained from various explorations (real world and simulation), characterizes such distributed representation of space, and addresses possible difficulties in closing loops.

ch. 7 Summarizes by discussing overall **results and conclusions** about a distributed representation, points to open problems and offers paths to extend the current system.

Each of the following chapters starts with a brief discussion and closes with a summary.

2 Representing Information about the Local Environment

Why at all spending energy and time on developing a sophisticated representation of sensor data from a mobile agent? There are many answers to that question, probably the most important being that the data representation reflects much about the nature of the solution one has in mind regarding the large problem. In our case, we are developing a distributed network consisting of active patches of knowledge to represent a large environment; so representing data at various levels of granularity and at various places within the system emphasizes the distributed nature of the system – and helps us thinking and developing ideas in such a distributed fashion. For simplicity and algorithmic elegance, the same data structure shall get used for single sensor data as well as for complex local-range place representations. So on one extreme it needs to memorize such diverse elementary information as “here exists a coffee machine”, “this direction is blocked”, and the “distance and direction towards a neighbor”. But on the other extreme, it needs to represent fused knowledge from a large number of different sensors taken in the vicinity of a particular place to provide a feature-rich signature that possibly varies over time, as e.g. a door might be open or closed depending on day-time.

Conceptually, we want the data representation to be a storage container, rather than an active processing agent. This means that some other active unit - as e.g. a sensor on a robot or a program - dumps information in the structure for later retrieval, or possibly forwards the structure to other active units for processing. The structure shall simplify maintaining information, but it shall not need to take navigation decisions based on understand stored information. Reasoning about the information is in the responsibility of the various active units in our system, such as “place representations” (chapter 1) and “behaviors” (chapter 5.3). Following such a simplistic principle for the data-structure allows applying it to accommodate vastly different active units, such as single sensors and complex algorithms, but also allows adding further active units without conceptual changes.

In our setup we use sensors offering different levels of richness: the simplest sensors only point in a direction, e.g. towards north, or - mimicking the vestibular system - report changes in rotation. Other sensors provide more complex data, such as distances and directions to objects, report wheel translation, or identify previously seen objects. All these different kinds of data need to be represented in the data structure. In general, not all existing information about the environment is visible to the sensors all the time; in fact some environmental-knowledge is not visible at all and can only be inferred from previous experience, e.g. already explored directions leading into a dead-end - and thus being depreciated for further exploration. Most of the sensors, however, have a temporally and spatially limited field-of-view, such that the data structure needs to remember and conserve previous knowledge.

In this chapter we will discuss in detail the data representation we developed for local spatial information, while keeping all the above mentioned requirements in mind.

2.1 Pre-Processing Sensor Information

Data directly from a physical sensor is hardly suited to be stored as is. Sensors typically measure and report voltage levels, which correspond to some physical quantity such as 'angle to north', 'distance to object', or 'brightness of a pixel in a camera image'. To simplify operations on the data, we need to convert the raw data into a useful form. Here arises an important question about how much preprocessing we want, or - alternatively - how "raw" we want to keep data in our representation. As an example of such data, one can think of distance measures to objects in a 180° field of view taken at regularly spaced angles: in one extreme we can store individual distances along with their corresponding angles; whereas in the other extreme, we can extract behaviorally significant aspects such as corners, edges, junctions, and possibly objects from the data - and remember only the type and position of these identified objects.

Both approaches - raw and preprocessed - have advantages and disadvantages; Table 1 compares both extremes. Different types of sensors on the robot require different levels of pre-processing before providing relevant data for navigation; e.g. pixel-based color-values from the on-board video cameras need at least simple feature extraction before being useful. Each sensor's pre-processing algorithms will be discussed later in this chapter when introducing the sensors. In this project, we decided to use an adaptive tradeoff between the two extremes: Initially, we want to operate on data as raw as possible, so that we can interpret and verify information easily and do not accidentally neglect potentially important aspects. In the following discussion about the data representation, however, we will keep in mind that adding pre-processed stimuli or even replacing all raw data by behaviorally significant stimuli has to comply with all our design decisions. In fact, as outlined below, adding pre-processed data of arbitrary abstraction-levels will simply introduce additional "layers" of information, which will be treated identically to the possibly co-existing layers of raw information. Creating such a flexible data representation allows simple extension of the system.

	Raw Data		Pre-Processed Data
+	simple principal, fast performance, no computational overhead for data inter-pretation	−	difficult to argue why a particular pre-processer is ideal, various pre-processing options exist
−	no distinction between significant and non-significant data, wasting memory	+	abstracts data immediately to a compact representation, discarding "junk data"
+	no bias towards human interpretation of objects: What does an intersection of aisles mean? Why is an object's corner interesting?	+	highlights potentially significant aspects contained in data
−	memorized data difficult to interpret for higher-level processes	+	facilitates abstract decisions processes relying on the data, e.g. about traversable space
+	all available data is maintained, can potentially be used for abstract processing	−	potential danger to miss relevant aspects which might be invisible for the pre-processor
+	allows to inspect sensor's data directly instead of inferring from processed data about sensor behavior		

Table 1: Comparing raw sensor data with pre-processed sensor data.

2.2 Design of the Data Structure

2.2.1 Preparing Raw Data

All sensors on the robot continuously report data of a different kind. However, we want to maintain all information in a common framework. In order to fuse the different types of information into a unified framework, we use a trick and translate all data to "events" with associated locations and/or directions relative to a local coordinate frame. This approach is straight forward to see for some sensors: e.g. distance sensors report an event called "blocked space" at a given position. Other sensors' translations are more difficult to imagine, e.g. wheel rotation measurements. All details about the individual transfer functions are explained in chapter 2.4 where we introduce the sensors individually.

Continuously receiving those events requires storing them in an orderly fashion for later processing. In a simple approach, all events can get stored in a large list or lookup-table, possibly with indices for different sensors to facilitate data retrieval. A strategy such simple quickly creates huge data sets that grow computationally intractable. To reduce memory and processing power we group near-by events (regarding space and time) of the same type into a single bin, and immediately discard the event's individual representation. Thus for each type of event we only memorize the amplitudes of occurrences close to various positions within a grid. The spatial extend of these bins for storing events is discussed in the following chapters, where we initially assume a single stationary sensor providing a single type of events. Only later, in chapter 2.3 we extend the structure to accommodate events from multiple sensors, whereas in chapter 2.5 we extend the structure in time.

Note that all sensors either provide one-dimensional, two-dimensional, or three-dimensional events. Examples for these types of events are compass directions (1d), objects blocking space at a given position (2d), or a motion-target at a given position and orientation (3d). The data structure

introduced in this chapter accommodates all different types of sensor information following the same basic principle for representation.

2.2.2 Representing 1-Dimensional Events

One-dimensional events provide a tuple of sensed quantity and direction; e.g. a light sensor returning brightness readings from different angles or a compass providing angle to north. We represent received events in a binned polar structure with each bin corresponding to a particular angle (see Figure 3, left). The proportion filled of the circles equally spaced around 360° denotes the event intensity from the respective direction. In the scenario of a light source providing brightness stimuli, Figure 3 corresponds to the light source placed at +45° in the robot's perspective. The angular spread of the signal can be explained by light beams hitting several nearby receptive elements of the sensor (weaker towards the outside of the beam), or by sensor noise. Events falling between the angular positions represented in the grid get distributed to the two directly neighboring bins based on their position: A single brightness event of amplitude 0.8 at an angle exactly between two bins will contribute 0.4 to each of the two bins.

Assuming a second light source elsewhere in the field of view, we would see a second region of filled circles. For sensed values that are known to originate from a unique source (e.g. the magnetic field) the population-average of all circles decodes a corresponding angular value, as shown by the dark blue vector in Figure 3, left. For details refer to chapter 2.2.5.

A computer's internal representation is not polar-coordinate based; instead we map the bins onto a linear array, each entry representing a bin of the original polar arrangement (see Figure 3, right). Note the red arrow indicating a wrap-around at ±180°. As the two representations are identical in terms of data content, we will continue using the left display (a top-down view of polar bins, conforming to the physical arrangement) throughout the rest of this thesis for simplicity.

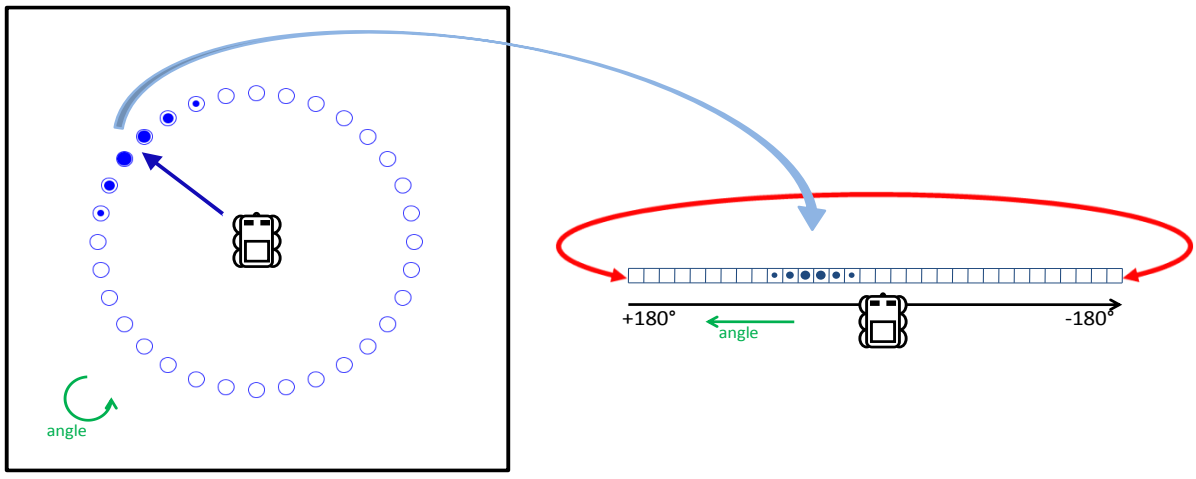


Figure 3: Left: Top-down view of the system's 1-dimensional event representation. The proportion filled of circles denotes the amplitude of reported events; the dark blue vector shows the population-coded average angle. Right: A computer's internal linear representation of identical data, wrapping around at ±180°.

2.2.3 Representing 2-Dimensional Events

Two-dimensional events report an additional dimension, typically distance. So we need to represent distance, angle, and amplitude for a 2d-event. It is important to note that in the context of navigation, nearby information typically requires higher spatial precision than distant information, as shown by the following two examples:

- The maximal eligible forward speed strongly depends on small uncertainties in position of nearby objects, but not on identical uncertainties about distant objects.
- An estimate of angle-to-object varies drastically with position-error of nearby objects, but substantially less with the same absolute position-errors of distant objects.

We design our binned 2d data structure to minimize memory requirements while providing adequate precision for all distances: The distance between the centers of neighboring bins increases with distance from the center of the representation. Nearby positions get a fine grained representation of bins within a few cm distances; whereas distant positions get mapped into coarse bins. We chose an exponential function to calculate the distance d_n of bin n [0..n_{tot}-1] from the center as follows:

$$d_n = (n + 0.5) \cdot s_{min} + \left(d_{tot} - (n_{tot} \cdot s_{min}) \right) \cdot \frac{\left(b^{\frac{n}{n_{tot}-1}} \right) - 1}{(b - 1)}$$

with s_{min} the minimal bin size (5cm), d_{tot} the total distance to cover (5.0m), n_{tot} the total number of bins (32), and b the base of the exponential function (1000). For a discussion of the values for these parameters refer to chapter 2.2.6.

The first factor guarantees a reasonable minimal bin size for small n where the exponential function is close to zero. The following term in parenthesis computes the maximal distance reached by the exponential function, given that a fraction of d_{tot} is already covered by the first factor. The last term in the equation implements the exponential function, normalized to values between [0..1]. This equation provides small bins of sizes close to s_{min} for events happening within 1m from the center. Events occurring beyond 3m distances get represented in much coarser bins, as shown in Figure 4.

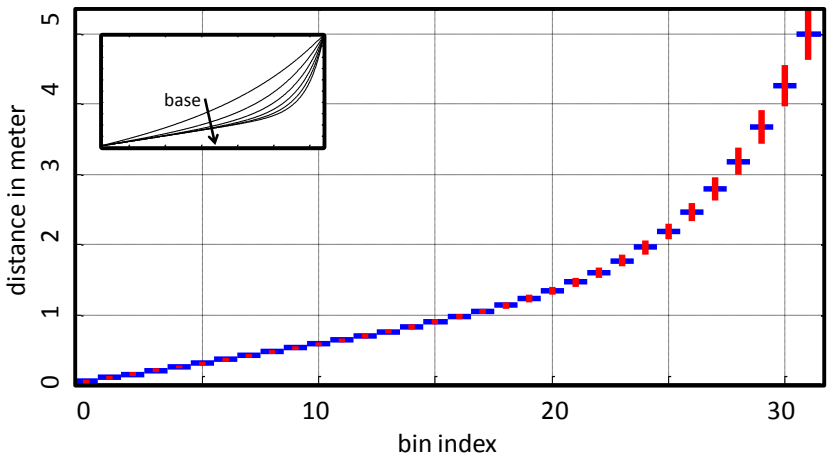


Figure 4: Distance (blue) and length (red) of bins in the 2-dimensional event representation. The inset shows bin-distances from the center as a function of increasing base values ranging from 10^1 to 10^6.

We keep the angular extend of bins constant independent of distance to center, such that the angular uncertainty remains constant independent of distance. Such a “binned log-polar” representation (see Figure 5, left) allows representing sensor events in a compact form while maintaining high precision near the center as required for navigation. The black dots in Figure 5 represent received distance information from an exemplary sensor: the darker a bin the more certain this space is blocked. One can detect an open path towards the left and possibly towards the front in the robot’s perspective; whereas the passage towards the right is blocked.

Incoming events get assigned to bins according to their associated position. Usually, events do not exactly fall in the center of a single bin, so that each event’s amplitude is distributed amongst the four direct neighboring bins according to spatial vicinity. The apparently “too solid” representation of obstacles in Figure 5 arises from distributing event data, but also from sensor uncertainty and from the obstacle’s true surface distance crossing through multiple neighboring bins, as we will further discuss when introducing individual sensors in chapter 2.4.1.

Today’s computers do not operate directly on log-polar coordinate systems; instead we use a mapping to a regularly spaced 2-dimensional array (Figure 5, right) to represent the reported events. The horizontal axis represents angles (note the red arrow indicating a wrap-around at ±180°), whereas the vertical axis shows distances from the center. The green insets in Figure 5 show the two different coordinate systems; the blue circles show a corresponding patch of data in both coordinate systems. It is obvious to see in the right diagram that space near the center gets represented in finer detail: all the “empty” space within a 1m radius around the robot takes up more than half the storage space. All the blocked space in the 5m circle around the robot, which significantly dominates the left diagram, is mapped in less than ½ of the available bins towards the top of the right diagram. The open passage to the left of the robot is still clearly visible by a vertical opening at around +75 degree.

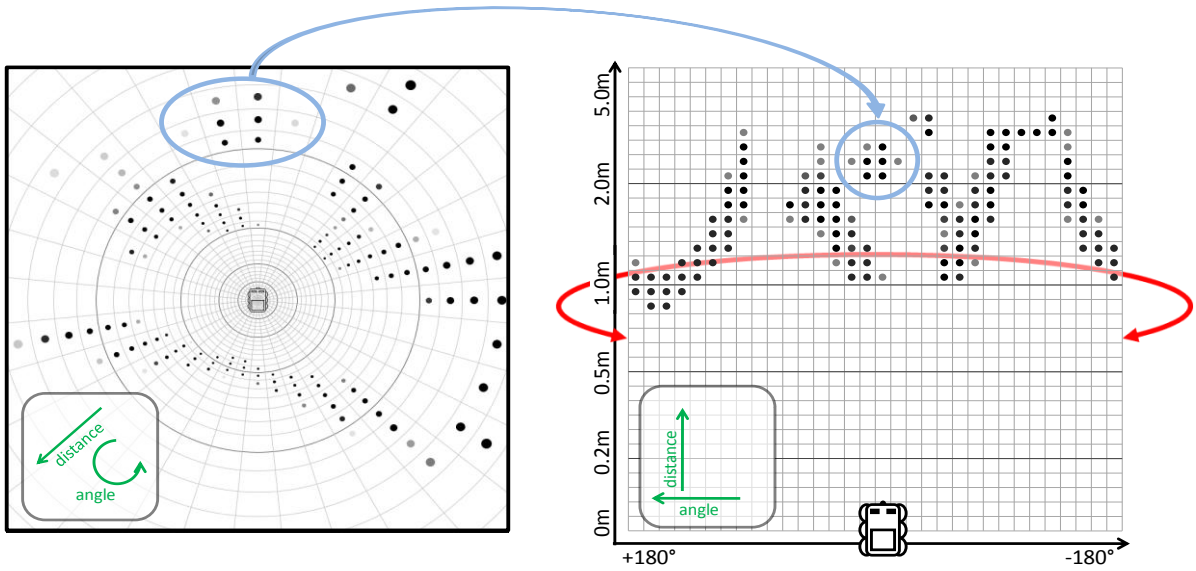


Figure 5: Left: Top-down view of a robot-centered binned log-polar representation. Right: Mapping of the log-polar structure to a regular Cartesian grid. The darker a circle in a bin, the more certain this space is blocked. The intensified circles (left) and horizontal lines (right) represent distances of 0.2, 0.5, 1.0 and 2.0 meters from the center, respectively.

As the two representations are identical in terms of data content, we will continue using the left display (a top-down view of log-polar bins, conforming to the physical arrangement) throughout the rest of this thesis for simplicity.

2.2.4 Representing 3-Dimensional Events

Reading "3-dimensional" data one typically thinks of space in x-, y-, and z-coordinates. Here, however, we limit ourselves to operation in a plane or almost planar surface (see chapter 2.4.2). Instead of typical 3d, we have data that represents a combination of a 2d (position) and a 1d (orientation) event, as e.g. recognizing an object in a distant position together with the object's orientation. Note that position and orientation are independent of each other: an object at a distant place can be rotated, hence keep the position but have a different orientation. For simplicity we decided to represent such 3d-events in two structures: a 2d-structure for position and a separate 1d-structure for orientation. We can re-use the previously developed data structures and acquire one of each to store 3d-events. We maintain both structures in a single object referred to as a 3d structure.

We believe that an extension of the data structure to represent a full 3d-world with an up to 3-dimensional position event and a related up to 3-dimensional orientation event is easily possible. Refer to chapter 2.8 for a discussion.

2.2.5 Storing Additional Attributes of Events

The described polar and log-polar binned data structures remember positions and amplitudes of events. To reflect the nature of the data stored in a particular instance of the structure, each instance maintains a small number of additional attributes, set by the source of the event. These attributes allow an external observer to operate on subgroups of the data (e.g. inspecting all "blocking" events at once, see below) rather than handling each type of events individually.

The simplest attribute is a flag denoting whether the observed event is physically blocking space, making it non-traversable for the robot. Events from a distance-to-obstacle sensor by definition block space, whereas information about visible landmarks does not necessarily indicate blocked space: A landmark might simply be a colored cross drawn on floor that is well traversable. Inspecting this flag allows the data structure to provide information about traversable paths without having a 'deep understanding' of the data stored.

An additional flag indicates if recorded events are absolute values with respect to an external baseline, or result from an integration process with respect to a previously set baseline. Absolute recordings, as e.g. the direction to the magnetic north pole, show data with a mean-variance of zero over time for a stationary sensor. Some sensors, however, integrate signals with respect to a baseline, e.g. a rotation sensor mimicking the vestibular system that initially needs to obtain a "forwards" direction. These sensors generate slowly drifting signals over time, so they require a periodic reset and capture a new baseline. Setting this flag according to the nature of the sensor allows handling all integration-based sensors periodically without knowing further details about the kind of data they provide.

A scalar value describes exponential time decay applied to the stored data (refer to chapter 2.5), reducing the certainty that a received event still describes the true environment. Some sensors report events at high frequencies, such as e.g. the magnetic north. These kinds of events decay

quickly over time, as updated data is available quickly. Other sensors as e.g. vision are more expensive to process such that new events get reported muss less frequently; hence we decay this information slower.

The last principle difference amongst sensor events specifies whether a single source is responsible for the reported events. Alternatively, events can originate from multiple indistinguishable sources or multiple individual sources. In the first case, all events received from a sensor share a single binned representation. A target-sensor is such an example, where a single desired target position provides stimuli. All consecutively received events get averaged using a population code weighing schema to receive the best estimate of the target's true spatial position (see Figure 6, left). In contrast, a distance sensor reports events from multiple indistinguishable sources: All reflecting objects provide various events consisting of angle and distance. All these reported events are represented in a single log-polar map, but it would be very little useful to compute a population vector average: Looking at Figure 6, middle, the population average of the blocked space points roughly to the center, not providing any information about traversable space.

Thirdly, sensors can report events from multiple distinguishable sources, as e.g. a landmark-detection sensor does. Such a sensor continuously reports a set of visible landmarks, each elementary observation consisting of a landmark-identifier in addition to position and possibly orientation. We represent such events by creating a separate binned log-polar structure for each identified landmark, labeled with the landmark identifier (see Figure 6, right). Consecutive events reporting the same landmark get added into the existing structure. Each landmark's true position is computed using a population code average amongst the data in its binned log-polar structure. One can also think of such a landmark sensor detecting 10 different landmarks as a set of 10 sensors detecting one landmark each. However, as the reported events are of the same type, it seems conceptually more elegant to represent the data in a common framework, having a single reference.

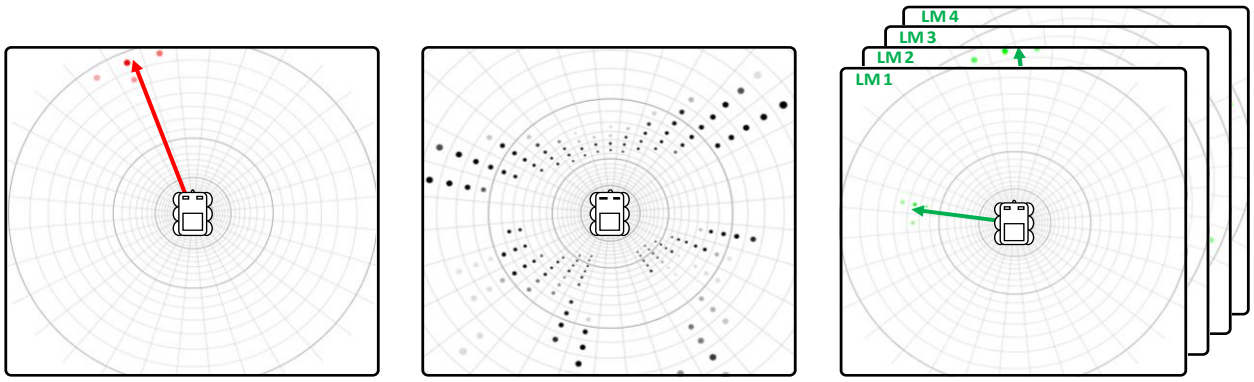


Figure 6: Left: representation of a sensor reporting events from a single source, population code vector points to the most likely position of the source. Middle: Distance data showing blocking objects. Right: Events from multiple individual sources represented in multiple binned log-polar structures, each computing the population code estimate of a source's most likely position.

2.2.6 Tradeoff between Memory Requirements, Processing Speed, and Accuracy

We have introduced a binned data representation to reduce storage space and speed up computation. How much do we gain using bins? This question is difficult to answer, as we have no

absolute reference to compare against. In this chapter, however, we will discuss the parameters that generate the structure of the log-polar based event representation and discuss our settings.

The first decision taken was about the total distance to cover in the data structure: Conceptually we design a short-range representation for data that is typically within sensor's reach. When the robot changes to a new "place", we create a new data structure (see chapter 4.2). But how long is "short-range" - and how large is a "place"? The answer strongly depends on the size of the robot and the structure of the environment. We argue that a typical place shall know events within only a few body-lengths of the robot. Only in exceptional cases, such as a large empty room, or a long corridor without intersections, a place needs to know events beyond. Consequently, we chose a 5m radius for the representation, allowing all events reported within a 5m radius to get mapped directly into one or multiple bins. We do, however, use a trick to represent occasional events from larger distances: We introduce an additional "virtual" bin in a distance that by far exceeds the sensor outreach, e.g. at 100m. Events happening beyond the local 5m range get mapped partially to the 5m-bin and to the virtual 100m-bin according to their spatial distance. The position of a neighboring place (an event that typically has large distances) at 20m will get mapped strongly in the 5m-bin and weakly in the 100m-bin. In contrast, a neighbor at a distance of 80m will get a weak representation in the 5m-bin and a strong representation in the 100-m bin. Using a population code for readout (refer to 2.2.5) we can reconstruct a precise estimate of such long-distance events.

A second parameter to discuss is the smallest reasonable bin size: The smaller a bin, the more precise its position-estimate. However, the smaller the bins get, the more computing power and memory we need. Again, we decided to relate our estimate to the body-length and additionally to the driving speed of our system. As we require a security distance to walls of 20cm, a fraction of this seems sufficiently precise to avoid hitting blocked space. We chose the shortest bins (positioned in the center of the representation) to be of length 5cm, in order to stay well below the security distance. This bin size still is significantly larger than most sensors' resolution of a few millimeters, so we still group multiple events into bins. Reducing the bin size further ultimately leads to individual events stored in individual bins, similar to remembering each event in a long list.

The last parameter to discuss is the total number of bins for the representation: A larger number of bins results in more precise positions for events, but also in more occupied memory and more complex computations. Fewer bins, in contrast, reduce memory and speed up computation. To find a good tradeoff between the two, we need to balance desired spatial precision and available computing resources. Running several experiments on the real robot (chapter 3.2) and in the simulator (chapter 3.3) showed that choosing 32 bins for each dimension allows accurate short range navigation, while still performing in real-time with small memory requirements:

- 1-dimension: 1x32 = 32 bins
- 2-dimensions: 32x32 = 1024 bins
- 3-dimensions: 32x32 + 1x32 = 1076 bins of a single value each.

Comparing these numbers against for example a distance sensor reporting 120 distances in the field-of-view with a typical update rate of 5Hz, the 2-dimensional binned event representation saves memory within less than 2 seconds. The main advantage, however, occurs when collecting sensor events at a single place over long time: All events get incorporated into the existing representation,

so no additional memory is required. Such a long-term storage reduction gets especially efficient when introducing representations for behaviorally significant places in the environment (chapter 1).

In the current implementation, the parameters discussed above stay constant at runtime. In principle, an adaptive implementation of the data structure can modify these values during operation, accommodating for differently structured environments: Navigation in a large empty hall, e.g., might benefit from a coarser grid. Navigating in a tight maze with blocking walls every few centimeters on the other hand will benefit from a significantly reduced total distance. The principle we present holds true for a representation with fixed parameters, though: Both these places can get represented in a non-adaptive structure.

2.3 Representing Events from Multiple Sensors

We started this chapter stating that we initially explain the principles of the data structure looking at a single exemplary stationary sensor. Now time has come to extend the presented concept to maintain data from multiple sources.

We cannot place events from different sensors into a single representation, as this would require fusing different types of representations (1d, 2d, and 3d), but also mix non-related events before computing a population coded average. Instead, we use multiple instances of 1d-, 2d-, and 3d-structures to remember each sensor's events individually, grouped inside a container called "*O*". The symbol *O* was chosen to indicate the accumulative and comprising nature of the container.

The first event from any sensor arriving at such a container *O* creates an instance of the required 1d-, 2d-, or 3d-structure to store that event, and assigns the sensor's name as a tag to it. Consecutive arriving events from the same sensor automatically get added to the existing structure, determined by the sensor's name. The container *O* does not need to understand events to handle them correctly: it will simply add them to existing slices of data or create a new representation. Common operations on stored events, such as translation or decay (chapters 2.5), operate on all slices or on a selected subset determined by the slices' attributes (chapter 2.2.5), respectively.

For simplicity, in the following chapters we will display such a container *O* as a "see-through" view of all contained representations, as shown in Figure 7. For representations using a population code to estimate the event's true position, only the single estimated position will be shown as a colored dot to increase readability. However, for common operations on the data, the container internally always maintains the true binned event representation for all sensors.

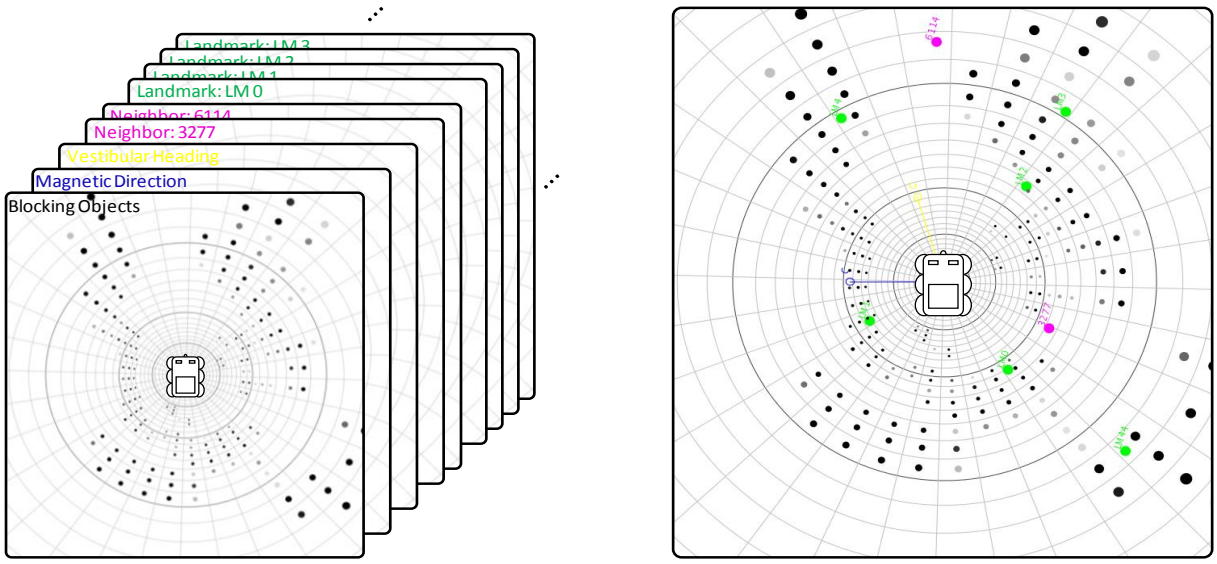


Figure 7: Left: Individual slices of representations constitute a container *O*. All stored events in the container can be addressed at once or individually. Right: a simplified "see-through" view of the events inside the container from the left.

2.4 Introducing Types of Sensors and Reported Events

In this chapter we will discuss the different types of sensors available to perceive an agent's environment, describe the types of events they report, and speculate about additional kinds of sensors that could improve the system's performance. We outline similarities and differences between sensors, and discuss consequences arising out of the differences. However, we do not describe details of the sensors' underlying hardware implementations here, and we do not discuss complex algorithms for event pre-processing, such as e.g. extracting objects from captured video images. This will happen when introducing the existing robot in chapter 3.2. The current discussion treats conceptual sensors reporting abstract events.

Using events from all sensors described in the following chapters increases the richness of the sensor-signature at a robot's position. The more details about the local environment available for navigation, the simpler the problem. However, the more events require processing, the more complex and slower the overall system gets. Ultimately, we need to find a tradeoff between computational complexity and richness of environmental knowledge. In the following chapters we will present the implemented sensors in our system.

2.4.1 Obstruction Sensors

Examples: Laser-Range-Finder, Infra-Red-Transceivers, Mechanical Whiskers, Ultrasonic-Transceivers

Probably the most important sensors in the context of navigation are those that report distances and directions to surrounding blocked space: A mobile agent needs to know where it currently can pass and where it needs to stop. Such sensors report a single type of event called "blocked space", with an associated direction and distance. Usually, such a sensor reports a large number of events spread over various different angles. A typical example is a sensor emitting a directed light beam in various angles and computing distances to the objects based on reflections (see Figure 8). Such a sensor potentially reports objects from very close to medium distances at relatively high update rates as required for look-ahead driving. Unfortunately, such a sensor does not report all objects

reliably; e.g. a large glass inset in a door is invisible to the light beam, although it absolutely blocks space for the robot.

Autonomous robots typically apply a second type of distance sensor to increase the chances of detecting blocked spaces reliably. Such sensors might be mechanical whiskers reacting on touch or ultrasonic transceivers emitting and receiving a sound wave. Both these sensors can detect “invisible” glass objects, but compromise in distance outreach, spatial precision, or update rate. These sensors often extend forwards and backwards and serve as an emergency-stop signal with low spatial precision for the moving robot.

All these types of sensors are absolute (refer to chapter 2.2.5), meaning that they continuously provide identical events for a stationary sensor, but also for a robot returning to exactly the same position and orientation in space after traveling. These sensors, however, do not provide information about the identity of objects: a blocked space is a blocked space, no matter if the blocking is caused by a door, a sofa or a chair. For several of such sensor, the robot can infer that space between itself and a “blocked space”-event is free space. We chose to represent such “free space”-events storing negative values in the corresponding bins. In contrast, bins representing invisible space behind objects as well as space out of the robot’s current field-of-view contain values of zero.

Figure 8 shows that such a sensor’s range is limited: the sensor’s outreach does not hit obstacles in the front; so the event representation on the right side reports open space towards the front. Also note that the “thickness” of walls in the event representation is due to multiple nearby light-rays sensing different distances, as highlighted by the blue circles.

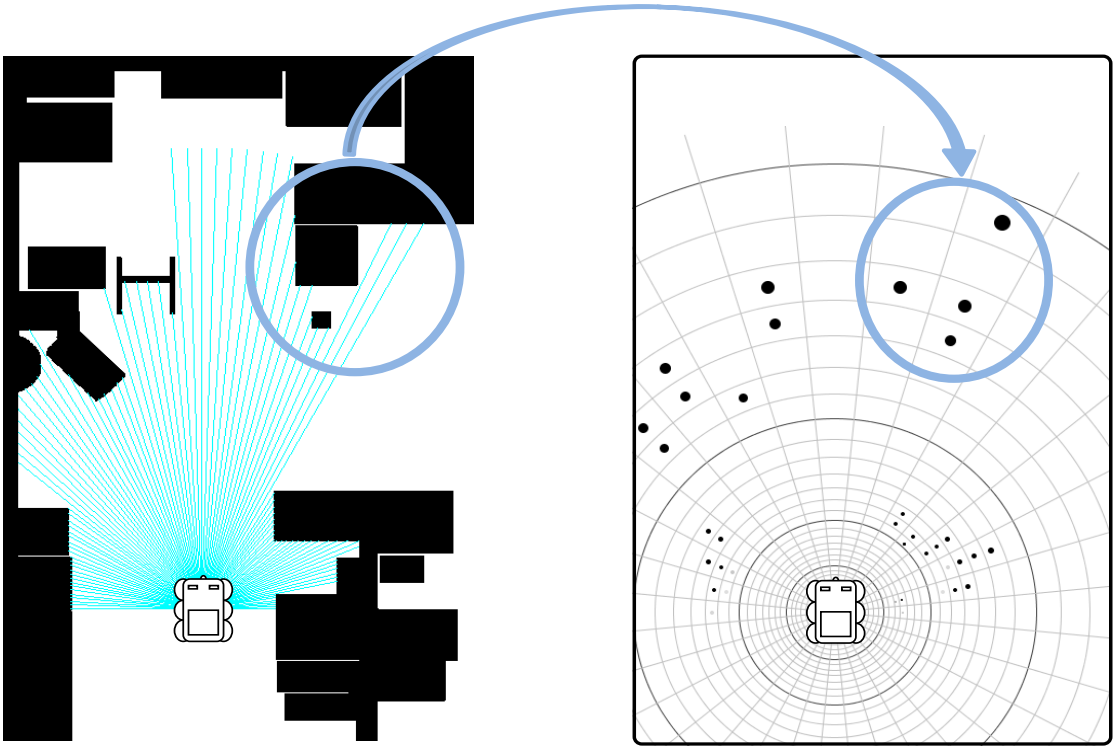


Figure 8: Left: A top down sketch of a robot equipped with an obstruction sensor, here a laser-range-finder, in a complex environment. Right: Representation of reported events in an instance of a binned 2d-log-polar map, with dark points denoting occupied bins according to the environment on the left.

2.4.2 Object-Detection- and Object-Classification Sensors

Examples: Vision, Barcode Scanner, RFID

These sensors identify an individual object, or the class an object belongs to, and report identity, position, and possibly orientation of the object. Examples of such sensor modalities are vision to detect objects like desks, chairs, or sofas; but also RFID sensors to scan nearby electro-magnetic tags. Even a simple sensor detecting doors based on a narrowing in the current passageway can be treated as such a sensor, detecting the single class of objects "door".

Most of these sensors do not identify objects uniquely ("Peter's sofa"), but rather report on significant objects in an abstract fashion ("a red blob on white ground", referring to Peter's sofa). For basic navigation it is not important to identify objects uniquely, although - if available - it does increase the certainty of position estimates. It is, however, important to reliably report the same description (the object's identity) on multiple consecutive occurrences of the object: When traversing an aisle several times, the same "red blob" (referring to Peter's sofa) shall be reported. Note that some of these objects are stationary, others are not: sofas and doors tend to stay at their places, whereas chairs or coffee mugs might get moved - or even removed. So if a salient object exists, the sensor needs to report its identity; if the object is absent, the sensor shall keep silent.

Such detected objects, which we call "landmarks", are not defined a-priori. They might show up anywhere, or there might not be any landmark for some time within sensor reach. It is important to stress that in our system, we do not provide a collection of existing landmarks, and neither do we provide landmark's positions. The robot must detect salient landmarks autonomously and remember them in its own coordinate frame.

As with the distance sensors described above, these sensors are absolute sensors: they report the same events (landmark identities and positions) whenever the robot returned to a previous position and orientation, unless the external landmarks have been shifted. These sensors detect fewer objects compared to distance sensors, and typically need more processing power to find object identities. Therefore, the typical rate for reporting events is significantly lower; down to not reporting events for some time. Please refer to chapter 3.2.7.1 for a typical view of various landmarks.

2.4.3 Heading-Direction Sensors

Examples: Compass, Gyroscope

A heading direction sensor is a one-dimensional sensor, reporting orientations without associated distances of events. A compass is a good example, reporting the direction towards north: a single type of event ("north") with an associated angle towards north in the sensor's coordinate-frame. If the robot and hence the sensor rotate by 90 degrees, the reported angle-to-north also changes by 90 degrees. The reported events lead to a blob of activity in a 1d-representation as described in chapter 2.2.2. Such a sensor again is an absolute sensor, pointing in the same direction given the same robot position and orientation. Due to electro-magnetic distortions, the same orientation to north at different positions does not necessarily report the same angle, as will be described in detail in chapter 3.2.7.3.

In contrast to a compass, a sensor mimicking the human vestibular system such as a gyroscope, reports changes in orientation. We need to pre-process this sensor's data before storing events in a 1d- representation: The sensor gets reset periodically, defining the current orientation to be "forwards". This resets happen whenever the robot is facing an upcoming target, e.g. a neighbor to go to or open space to explore (refer to chapter 4.3.2 for details). After reset, all signal changes get integrated, keeping track of the current orientation with respect to "forwards". The integration maintains an angle towards "forwards", similarly to the angle towards north. Note that the integrated estimate towards forwards, however, drifts from the true orientation towards "forwards" over time due to sensor noise getting integrated. Figure 9 shows the principle differences in signal-error between a noisy absolute and a low-noise integrated orientation sensor, both reset at the start of the recording.

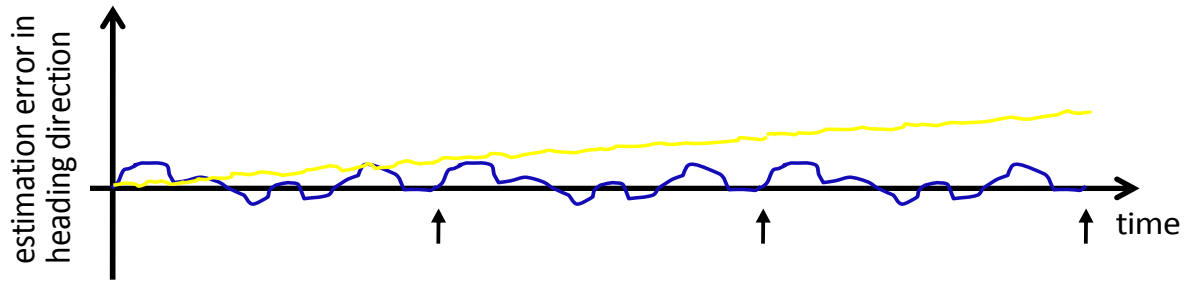


Figure 9: Estimation-error over time for a noisy absolute sensor (Compass, blue) and a low-noise integrated sensor (Gyroscope, yellow). The robot is performing three full rotations of 360° on the spot, with black arrows denoting a completed rotation. Note the periodicity of noise in the absolute (blue) signal, and the slow drift over time in the low-noise integrated (yellow) signal. A stationary sensor, in contrast, would produce a constant error in the absolute scenario, whereas the integrated sensor would still show signal drift.

2.4.4 Ego-Motion-Sensors

Examples: Wheel Integration, Optic Flow

Monitoring and integrating ego-motion provides another source of information regarding position in space. In the current system, the robot integrates its wheel rotations to estimate distances and directions traveled.

This quantity again cannot get represented as is in the data-structure; it requires conversion into a spatial position. For conversion, we periodically specify a new desired "target" position together with a desired orientation at target, e.g. whenever starting a journey to a well-known place (refer to chapter 4.3.2 for details). While moving, information from an ego motion sensor such as wheel-integration updates the imagined target position. These updates create events reporting the current estimated target position and orientation relative to the robots base position, that can get stored in a 3d-representation (2d-position and 1d-orientation). The sensor requires periodic resets to remove accumulated position error from the integration process.

2.4.5 Additional Elementary Sensors

Examples: Horizon Sensor, Acoustic Sensor

Here we will briefly outline additional sensor we discussed while developing the project. Although these sensors are not physically implemented on the robot, they would increase the richness of the sensory signature and therefore improve navigation performance.

A conceptually simple sensor is a 360° horizon sensor, recording quantities such as color, brightness, or 'height-of-object' at the horizon. Such sensors' events consist of a list of angles with sensed quantities to get stored in a 1d-representation. Multiple peaks of the sensed quantity are possible, e.g. various bright spots from multiple light sources in the environment. Such sensors provide only orientation information, similar to heading direction sensors. However, such sensors generate a signature based on local environmental features, causing the signature to be valid only at or close to the original recording site. A compass, in contrast, reports angles towards a global point of reference that remain valid over larger distances.

Acoustic signals can provide another source of navigational information, e.g. a constant hum of a nearby computer. Depending on pre-processing, the events can be 1-dimensional (e.g. the angle of the loudest sound, independent of frequency), 2-dimensional (e.g. the position-estimate of a nearby-sound), or multi-dimensional, if a filter-bank selects various frequency ranges for sounds and processes each frequency band independently.

2.4.6 Derived Events

Examples: Eedge-, Corner-, Junction-sensors

As discussed in chapter 2.1, the data structure accommodates elementary sensor events presented in the previous chapters, but also processed events, such as edges, corners, junctions. Such abstract events get derived from elementary data, as e.g. an "edge" event is computed from events stored in a blocked-space representation. An algorithm computing such events stores its new events in an additional layer maintained in the same container ***O*** that holds the elementary events. The algorithm simply creates new events of type "edge" with associated positions, and gives them to the container ***O*** for storage. The container handles all incoming events in the same fashion; it does not differentiate between elementary and processed events. Cascading this principle of deriving events is possible: Another algorithm can inspect derived edge-, corner-, and junction-events, and create a new "hallway" event to store in the same container.

2.4.7 Abstract Events

Examples: Neighbors, Explored Space

Some actors in the system need to store abstract information related to space, that is not accessible by sensors and that cannot be directly derived from elementary sensor information. Such abstract events include identity and position of interesting places in the vicinity, or preciously explored space that turned out to be irrelevant - and thus needs to get marked to avoid further exploration. The data structure is sufficiently flexible to handle these abstract events, simply by treating them like any other event it receives: It creates a new layer to store the events, labeled with a tag to identify these

events. Ultimately, the active entities that stored the events need to know how to interpret them; the container only memorizes and maintains the events.

2.5 Maintaining Events: Translation, Decay, Normalization, and Integration

2.5.1 Translating and Rotating Data Based on Robot Motion

So far we have only looked at stationary sensors; however, the robot carrying sensors is moving most of the time, requiring changes in the stored positions of events. We could simply forget old events and look at the current snapshot of sensor events, ignoring tiny motion within a single time-frame. Such an approach wastes a lot of existing knowledge about the environment, as events that accidentally did not occur within the current time frame will be unknown. Additionally, if the source of an event moved out of a sensor's field-of-view, it typically still exists in the world. As an example, a previously detected landmark that moved in the robot's rear still is a landmark, although it is currently not visible. The previously gained knowledge about landmark identity and position is still valid for position estimates and thus should remain in the representation.

To maintain previous events at valid positions in the structure, their representations need to get updated based on the robot's motion: Assuming a 90° turn counter-clockwise (positive direction) all stored events rotate clockwise by 90° (negative direction). Rotating events like this moves them to the position they are currently perceived from, given they are still visible. Events falling out of the field-of-view still remain in the representation, and thus still exist in memory. Similarly for translation: a forward-driving robot causes all stored events to shift backwards, to the position they are currently perceived from; ultimately moving in the robots rear where they cannot be sensed any longer but remain in memory.

To perform such translations and rotations of events, we need to know how far the robot has moved. However, sensors providing error-free measures of translation and rotation do not exist. As approximation we can rely solely on an ego-motion sensor such as wheel rotations (chapter 2.4.4) to compute translations and rotations of events. Moving all events based on a single sensor's estimate, however, introduces a large overall uncertainty: measurement errors from this sensor have direct impact on all other sensors' events. But no single sensor shall be in such a powerful position, being able to accidentally overwrite all other sensors' estimates. Therefore, we decided to translate and rotate all events in memory based on the robot's desired motion - independent of the truly occurred motion. Additionally, we introduce a method to correct errors by comparing event-positions from all sensors against their predicted positions, as will be described in detail in chapter 2.5.3. Applying this method estimates the robot's motion based on a prediction, corrected by events from all sensors. This method turns out to be significantly more robust compared to relying on a single sensor.

Translation and rotation do not happen as a singular event: the robot will not jump from a place to the next, rest, and jump again. It rather moves continuously, such that stored events need to continuously translate and rotate in small fractions. Such incremental updates generate a problem regarding the position of individual events: When the robot travels, events stored in one bin do not transfer completely from one bin to another, because the robot is unlikely to travel exactly the difference in distance between two bins. We have to decide how to handle events stored in a bin

when moving: keep all events together in one bin or split them in fractions into multiple bins. Keeping all events together in a single bin does not reflect the robot's motion: typical motion-increments are smaller than the size of a bin, such that all events of a single bin fall back into the same bin after applying a motion-update. Hence, even after large distances traveled composed of many reported incremental motion updates, all events are still in the same bin as before. The alternative approach, splitting events from a single bin into multiple bins, immediately blurs available events all over the place, as a simple example in a 1d-representation demonstrates (Figure 10, left side): Assume an event of certainty 1 is represented in a single bin, called "B0". After traveling a short distance, the certainty from B0 is split into B0 and its direct neighbor B1. Upon receiving a second small motion update, both certainties from B0 and B1 get translated again, contributing to B0 and B1, but also to B1's neighbor B2. Therefore, after only two small motion updates, a fraction of the initial event from B0 already moved to B2, although the total distance traveled was not sufficient to move events from B0 to B2 directly. Such an expansion of initially tightly represented events continues with every additional incremental motion update and further blurs the event's position. This effect is significantly worse in a 2d-representation, where every incremental update moves data from one bin into three new bins, instead of only into one additional bin.

To avoid this problem we separate incoming events and keep multiple slices of representations, separated by time, distance traveled, and angle rotated. Only consecutive events fulfilling the following criteria get added into a single slice:

- Observed within a single time bin of at most 2seconds.
- Distance traveled between first event and current event less than ½ robot body length.
- Robot rotation of at most 12° between reports of any two events.

If one or more of these criteria are broken, further events get grouped in a new slice, and the evaluation restarts. These criteria ensure that only events observed within short time and from nearby robot positions get represented in a single slice, resulting in low position variance of all types of events within a single slice.

All these events in a single slice constitute the "base knowledge" of this slice, which will neither rotate nor translate because of robot motion. In addition to the fixed base knowledge, each slice maintains a motion-integration vector which accumulates the robot's total motion after establishing base knowledge. Whenever current positions of past-time-events are requested, the slice internally translates and rotates a new copy of its base knowledge according to the current motion integrator, and provides this new copy (see Figure 10, right side). All externally reported events thus reflect the robot motion and provide the best position estimate of events as observed from the current robot position. Note that a base event represented in a single bin is still likely to show up in multiple bins as a current event, because the motion integrator will likely translate the event's position to a new position between bins. However, the important aspect is that the base knowledge does not spread out and flood multiple bins, as it is only translated and rotated once in each new copy derived from the base knowledge.

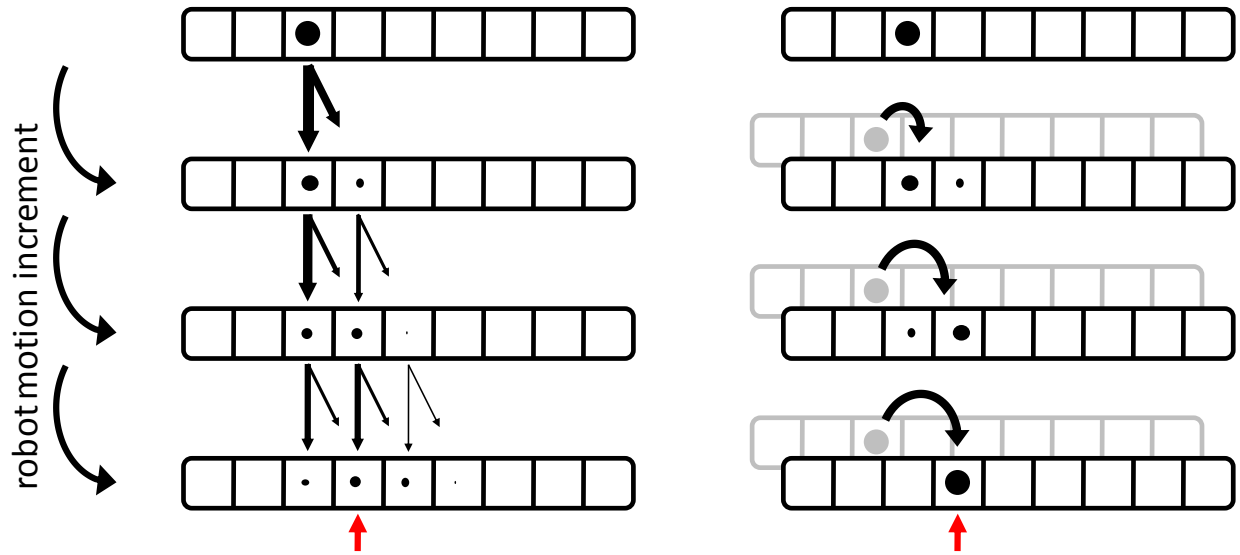


Figure 10: Maintaining events in a 1d-representation while robot driving in three consecutive steps of ⅓-bin-length each. Left: Shifting events for every incremental motion step blurs the initially precise information. Right: Keeping a fixed base (gray) with a motion integration vector reflecting the current total translation (black arrow). Applying the translation to a new copy of the base at every instance keeps a well localized representation. Note that the population code vector (red) still reports identical positions in both cases; however, multiple events stored in such a structure lose their individual representation.

2.5.2 Decaying Stored Information

Maintaining a large number of slices full of events offers a detailed representation about the environment. Some of the available information is new, whereas some stored information is older, i.e. reported earlier in time and possibly reported from distant places. Older information is typically less accurate compared to newer events, as the environment might have changed after recording, or the robot might have accumulated errors in motion estimates. Therefore, generating a combined estimate of the current knowledge should rely more strongly on new data than on old data. To reduce the impact of old data, each individual sensor provides a decay value for the events it reports (refer to chapter 2.2.5): events with high update rates typically decay quickly, whereas rarely occurring events decay much slower. Each slice - in addition to maintaining a motion integrator - monitors time after its creation. Knowing about time past and distance traveled, each slice decays its base knowledge of all bins $b_n(0)$, computing the bins' current values b_n after time t using the following exponential function:

$$b_n(t) = b_n(0) \cdot e^{-\left(\frac{t}{t_0}+\frac{d_{abs}}{d_0}\right)}$$

with t denoting the time past, d_{abs} the absolute distance traveled independent of direction, and t_0 and d_0 the time and distance normalization respectively, as provided by a sensor. To reflect changes in robot orientation, the accumulated absolute rotations contribute to d_{abs}. Note that t and d_{abs} can only increase in value, thus b_n continuously decreases. Applying this formula to compute a bin's current value b_n allows recent events to have stronger impact over decreased older events. Eventually, b_n falls below a significance threshold, such that $b_n(0)$ gets cleared to save processing time. With all $b_n(0)$ decayed, the whole slice with all outdated events is discarded.

2.5.3 Correcting Errors in Ego-Motion

The robot executes motion commands as precisely as possible. Typically, however, it accumulates errors in driving distance and orientation quickly, as small offsets in orientation lead to large position errors while driving. Several reasons for such errors are internal, e.g. misadjusted wheel speed or unbalanced wheel diameters. Characterizing the robot carefully compensates these errors largely, but not completely. Other sources of error in ego-motion are external: driving over uneven surfaces, hitting small objects, or slipping wheels on ground causes the true path to deviate from the desired path. Such errors are unpredictable and cannot get minimized before they occur. Refer to the inset in Figure 11, red trace, to see an example of a trajectory accumulating error during a short exploration phase. The blue trace shows the robot's true trajectory, whereas the green trace shows an error corrected trajectory. Note that the error-corrected version still shows inaccuracies along the way.

The data representation needs a reliable estimate of robot motion to translate past events. Events from the robot's on-board sensors provide information about motion relative to the environment allowing the data structure to detect and correct unexpected perturbations. Using events from a single sensor to modify all recorded events is dangerous: if this sensor reports poor estimates of the true motion, this sensor's error spreads out to all other sensors. In the ultimate case, if this sensor breaks, all other sensors fail as well. Instead, ideally all sensors contribute to a common perceived motion estimate, which determines how to translate recorded events.

To compute such a common motion estimate, the data structure compares the newest recorded slice against all past events, whenever it is about to start recording events in a new slice. It creates a "past signature" (*ps*), summing all stored time- and distance-decayed events in a single slice. Note that all slices have been translated and rotated according to how the robot expects it has moved. *ps* represents the best estimate about the robot's current view without the newest slice of events, and without correction of recent motion-errors.

Comparing *ps* with the current slice (*cs*), as will be explained in chapter 2.6, provides an estimate of the motion error that occurred based on events reported from all sensors: The distance measure ***dist*** = *cs* - *ps* provides a 3d-vector (2d-translation and 1d-orientation) for correcting offsets in motion. The data structure translates and rotates *cs* according to ***dist*** before storing *cs* in the list of past events to keep consistency in the recorded data. In addition, it keeps a list of distance measures ***dist***, to compute the average distance that occurred for the last 8 slices added to memory (***avgDist8***), but also the average distance of the last 4 slices (***avgDist4***). If ***avgDist8*** exceeds a threshold, set to ½ of the robot's length in translation or 15° in orientation respectively, we assume a large error in ego-motion recently happened. ***avgDist8***, however, might not describe the change correctly, as it still contains slices recorded before the error occurred. We will only take an action to correct ego motion once ***avgDist4*** and ***avgDist8*** indicate a similar change, suggesting that the error has happened before recording events. With both ***avgDist4*** and ***avgDist8*** exceeding the threshold, we translate or rotate all memorized data according to ***avgDist4***, subtract ***avgDist4*** from the distance memory, and notify the robot about the detected offset in ego-motion.

Discrepancies between *cs* and *ps* typically have two different sources: temporary sensor inaccuracies or a moderately misplaced robot, e.g. because of driving over a small obstacle. For the first case the recorded sensor data of individual *cs* gets corrected as accurately as possible and added to memory,

but the ***avgDist***-values are unlikely to exceed the threshold; ego motion does not get corrected. In the second case, all *cs* recorded after misplacing the robot show a consistent offset, such that ***avgDist*** slowly increases and - after some time - permanently reports that offset. Offsets below the defined threshold get ignored and new data gets matched for position individually. Once the threshold is exceeded, all previous data get aligned with current events; all future events are aligned correctly until further errors in ego-motion occur. This principle for correcting errors in ego motion results in significantly improved position estimates compares to simple motion integration based on wheel encoders' reports (see Figure 11, red trace showing the robot's position error using wheel encoder integration only, whereas the green trace shows the position error when applying error correction).

As an alternative approach we tried continuously correcting a small fraction of ***avgDist8***: translating and rotating all stored data by ⅛ · ***avgDist8*** corrects large errors in ego motion reliably within a few iterations. Unfortunately, applying small changes continuously also leads to slowly drifting estimates of the robot's position, such that after some time position and orientation accumulated large errors. We decided to stay with the previous approach of correcting only larger accumulated errors once they exceed a threshold.

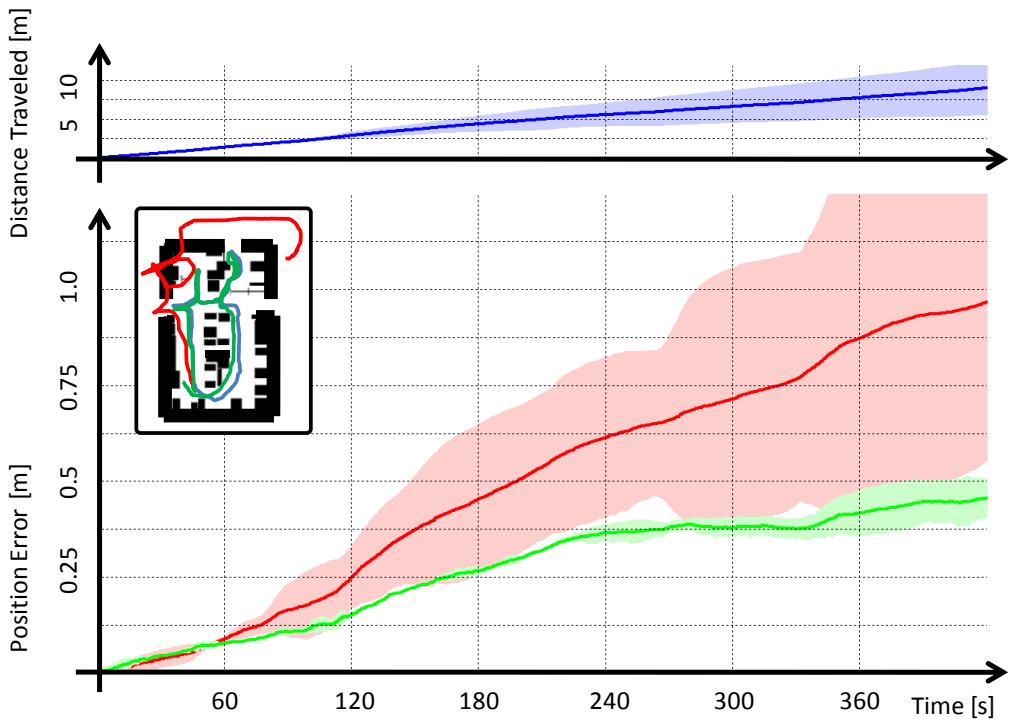


Figure 11: A robot's error in position estimates: red trace shows increase of position error over time as average over 100 trials, the shaded area shows one standard deviation. Green trace shows position error of the same 100 trials with enabled error correction. The blue graph above depicts traveled distance within the time windows of 7 minutes. The inset shows an example of the increase in position error during a single exploration trial over 30 minutes in a top down view of the robots computed positions (red/green) versus its true position (blue) in a room of 8x15m.

2.5.4 Best Estimate of Current Local Environment

Up to now we have presented data as a collection of slices recorded over time. However, when taking behavioral decisions about the environment based on stored data (e.g. fining a traversable

path), having individual slices does not increase content but only complicates computations. The data structure provides a combined representation of all events upon request, showing the current collective knowledge about the local environment, e.g. for storing snapshot views of a local environment. To compute such a collective view, we fuse all translated and decayed events in memory into a single new slice by adding all values of bins at corresponding positions.

Merging all events into a single slice might generate very large values in particular bins and small values in others that have been observed less frequently. This might cause problems in further processing, e.g. when comparing two snapshots. We keep the true sum of all memorized data in the base bin (refer to chapter 2.5.1), but apply either of the following two correcting schemata to the values in the current bins:

- Each bin of events coding for “blocked space” is independent of each other. Therefore, events from bins at identical positions but different slices get added and finally cut-off at peak-values of +1 and -1, indicating that a bin is certainly blocked or certainly free. As discussed in chapter 2.4.1 most such sensors provide blocked-space- and free-space-events, recorded as positive or negative values. We chose to weigh negative values with only ⅛ of their nominal value when summing bins, such that conflicting positive and negative reports for a single bin are more likely to report blocked space.
- Observed events originating from a single source use a population code to estimate the sources’ true position. The data structure initially sums all events from bins of identical positions into a new slice. The total value of all events contained in this new slice, however, shall not exceed one. If it does, all bins get normalized by the sum of all stored events. This normalization does not change the position indicated by population code, but reflects the significance of events: Those reported only briefly stay below a total value of 1, whereas those reported sufficiently often reach a significance of 1 but no more.

Summing all events into such a single slice provides a compact representation of the current environment, called a place-signature (*ps*). Note, however, that further translation and rotation of this *ps* suffers from blurring data as explained in chapter 2.5.1). The *ps* is only useful for temporary snapshots or quasi-stationary scenarios, e.g. when comparing the robot’s current environment with a *ps* of a previously visited place.

Figure 12 show a summary of the processing steps to turn sensory-events into a place-signature.

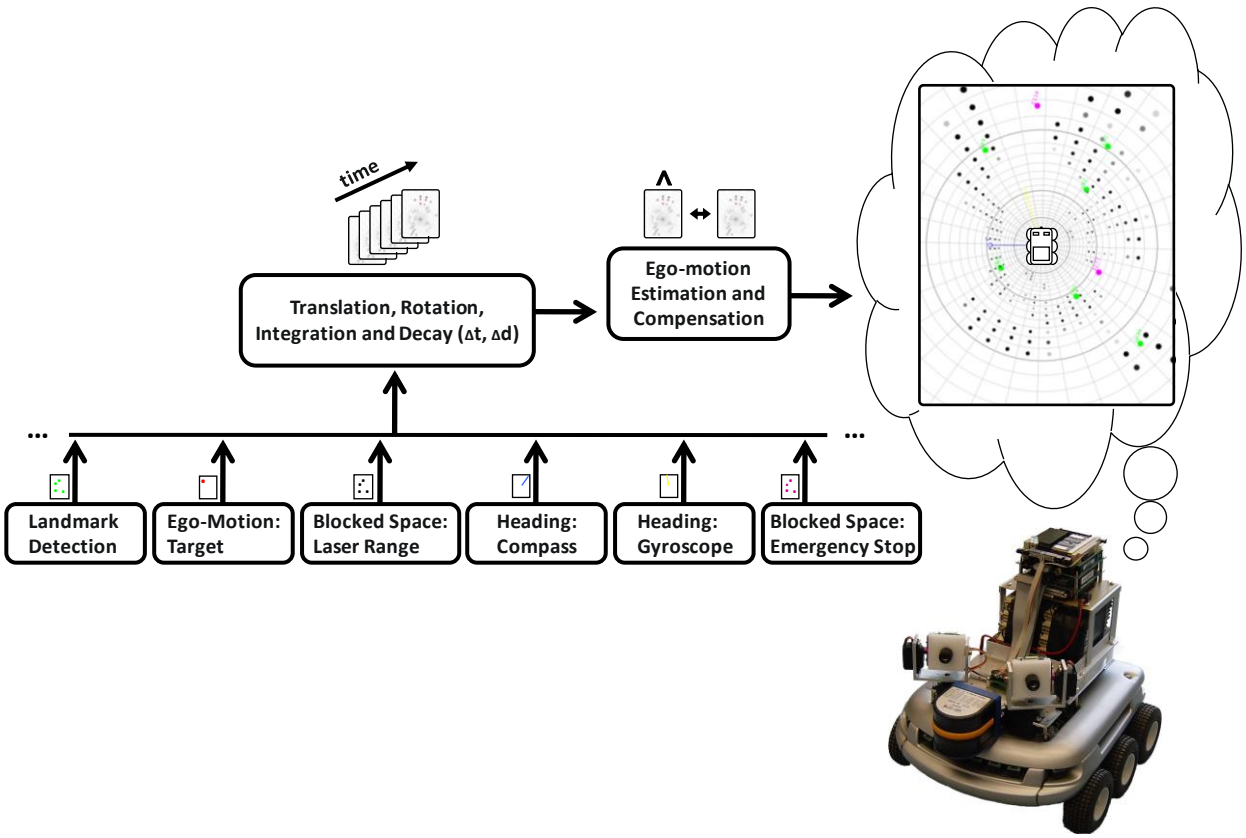


Figure 12: Turning sensory-events into a "place signature": collecting events from all available sensors, maintaining events in memory, comparing expected view with current view to correct errors in ego motion, and finally fusing all events into a snapshot.

2.6 Comparing Containers

Comparing environmental information in two containers is an elementary operation that the data structure performs. A simple scalar similarity measure ("distance") between two containers is not sufficient for navigation: As example, assume a robot wanting to center in a previously visited spot has already returned to the exact place, but faces a different direction. Comparing the stored place signature against the robot's current view should return a high similarity value, but also indicate that the current view is rotated with respect to the recorded place signature. Furthermore, when the robot is close to but not exactly at the stored location, the similarity measure needs to provide translation and rotation offset between the two containers.

Given two containers, only pairs of data from sensors that are available in both containers are compared. However, multiple of such pair-wise comparisons typically report different similarity values and might even report differing offsets for "best matches". The data structure needs to perform multiple pair-wise comparisons, inspect the results and generate a "common best estimate" for similarity and offset. This best estimate should not simply be the average of all individual estimates, because with such a simple algorithm a single confused sensor modifies the overall result significantly.

Internally, a data container O_1 compares each of its individual slices with a slice of the same type in another container O_2 if and only if such a slice exists. For each such pair of slices S_1 and S_2, a copy of the first gets rotated and translated before comparing with S_2. Performing a large number of such

translations and rotations results in a detailed similarity-landscape for various possible offsets between S_1 and S_2, closely resembling a convolution of the two slices. Figure 13, upper row, shows such a convolution for an exemplary 1d data structure. The peak activity of the result S_1*S_2 is at about 160 degrees clockwise from origin, corresponding to the rotation of S_1 for maximal similarity with S_2. Note that the angular resolution of the result is much finer compared to the original data, allowing detecting rotational offsets smaller than the original angular resolution.

Comparing 2d data structures follows the same principle described above; however, in addition to rotating the original data to find the best match, the data also needs to undergo translations. The resulting similarity-landscape is 3-dimensional: for each possible rotation angle the data is also translated to various positions relative to its origin, described by distance and angle. The incremental steps between shifting distances are small close to the origin, but increase further away; allowing precise distance estimates in the vicinity while saving computational resources further away. Comparing 3d data structures - that are internally composed of a 1d and a 2d structure, refer to chapter 2.2.4 - happens simultaneously for the two internal structures, combining the results. Therefore, the similarity-landscape still is 3 dimensional, providing the similarity measure for a combined translation and rotation.

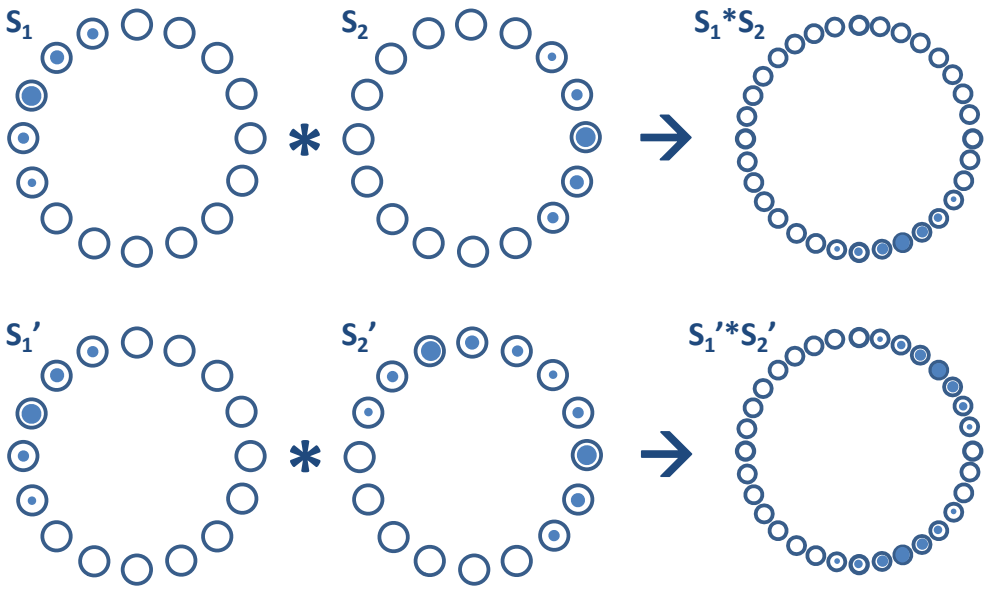


Figure 13: Comparing two 1d containers. Upper row: convolution of two containers with well located stimuli sources (S_1, S_2) provides a strong peak at the offset between both (S_1*S_2). Bottom row: comparing a well located source (S_1') with a multi-peaked uncertain signature (S_2') provides multiple peaks in the result (S_1'*S_2') rather than an average position, indicating that either of the two possible rotations is a preferred solution.

Under real-world conditions, sensor data typically is significantly noisier than in Figure 13, top row, and often shows multiple-peaks from sensor noise or driving errors. Comparing such data leads to multiple peaks in the similarity-landscape, as shown in Figure 13, bottom row. In 2d- or 3d-cases, more than a single offset can lead to optimal overlaps, such as e.g. data marking blocked space of a 4-way junction which looks identical for each 90 degree rotation. In all these cases, the convolution returns multiple peaks for each sensor, rather than a single average offset. This is advantageous, because the container O_1 that is performing the comparison can inspect all individual results and find the best combined solution. In our current implementation, all individual similarity landscapes get

added, such that common peaks support each other. After inspecting all existing pairs of slices and applying a spatial low-pass filter to the combined similarity landscape, the highest resulting peak describes offset and maximal similarity between the two containers.

This operation is computationally rather expensive, but we use a number of implementation tricks to reduce processing time:

- The comparison does not necessarily happen at arbitrary offsets, but might be limited in distance or angle. E.g. when the robot approaching its target, the system typically has an estimate of its orientation and remaining distance with respect to the recorded place signature. Therefore, we limit the comparison to angles within ±30° and to distances typically within 1 meter. However, when no proper match is found the search-range is automatically extended.
- All rotations and translations of data in a structure happen to identical target positions for consecutive comparisons. We do not compute these target positions on-the-fly, but instead pre-compute all possible shifts of the log-polar binned representation at system startup. Generating such a lookup table initially requires processing time (about ten seconds), but speeds up comparisons in the running system significantly.
- 2d representations containing data from multiple distinguishable sources (refer to chapter 2.2.5 and Figure 6, right) need most processing power, as they require a full 3d-convolution for each of their internal layers. Instead of computing these convolutions, we apply a trick: using the population-coded positions of each source we compute the 2d-distance of each pair of positions for all possible rotations. The rotation that produces the smallest variance in vector-distances denotes the best match for orientation with the vector 2d-distance denoting the translational offset.

2.7 Provided Functionality

The main purpose of the designed data structure is to store and maintain sensor-events. In addition, it shall generate and operate on a unified abstracted representation of previously recorded events from a local environment, the place-signature. The active units in the navigation system, such as sensors or place-representations (see chapter 1) shall not care about the underlying principles to represent and handle data. We have explained how the data structure maintains reported events from sensors. In this chapter we will briefly discuss intrinsic functionality that the data structure provides as service for abstract decisions, as e.g. comparing two place-signatures. Providing such a set of abstract functions, the active units do not need to understand or inspect the stored data in detail, but receive all behaviorally significant information abstracted from the underlying stored events.

Table 2 lists and describes provided methods operating on a container *O*:

add(single slice) add(other *O*)	These methods merge events stored in the provided argument into an existing container. Adding an individual slice initially checks if events of this type (i.e. from the same source) already exist. If they exist, the argument's current events (translated and rotated) get merged to the base of the container. If they do not yet exist, the provided slice gets added to the container. Adding a container to the current container adds each slice of the argument individually as describe above, thus ultimately adding the full container to the current container.
enterNewSensorData(identifier, Data)	This method takes a sensor identifier and a data object representing new data from a sensor. It creates a new slice (*s*) of events based on the provided arguments and calls "add(*s*)" to merge the new data into the current container.
getPlaceSignature()	Reports the best estimate of the current environment (place signature, *ps*) by flattening all past slices of events from a single source into a single current slice (refer to chapter 2.5.4).
setDecay(Δt, Δd)	Notifies the container about time past and distance traveled for decaying data (refer to chapter 2.5.2)
distance(other *O*) distance(other *O*, dist)	Comparing two containers returns a similarity measure and the spatial distance between both containers (refer to chapter 2.6). The optional argument "expected distance" causes different sensor modalities to contribute according to their relevance at the assumed distance.
findFreeDrivingAngle() findBestPlace() findDirectionsToMove()	These functions inspect all slices of the current place signature representing "blocked space" and return information about free space: findFreeDrivingAngle inspects events within ±45° orientation, returning direction and length of free space sufficiently wide to allow the robot to traverse. findBestPlace inspects the current place signature to find a nearby "center of free space", reporting an optimal starting position for further explorations. findDirectionsToMove returns a list of directions that are unblocked and sufficiently wide to traverse, starting from the current position.
isSignatureComplete()	Most sensors operate in a limited field-of-view, e.g. reporting events only from the frontal semi-sphere. This function inspects the current place signature and checks for an evenly distributed event occurrence. If at least one kind of events is reported substantially unbalanced, this method reports false. The robot might want to rotate 360° on the spot to refresh events that are currently invisible.
removeRelativeEvents()	Some types of sensors do not report absolute values but only changes with respect to a baseline (refer to chapter 2.2.5). These sensors require a periodic reset to compensate for accumulated drift. Whenever such a reset happens the recorded events get invalid; calling removeRelative-Events deletes all such relative events stored in the current container.
getNeighborList() hasNeighbor(id) getNeighborPosition(id)	The abstract events about neighboring places (see chapter 4.2) require special methods, as information about neighbors gets retrieved individually in contrast to other sensor's data. A list of all neighbors is obtained by calling getNeighborList(); the existence of a particular neighbor is queried by hasNeighbor(id). The method getNeighborPosition(id) reports distance and orientation towards a neighbor in the place-signature's coordinate frame.

display() save(), load()	These are internal functions for our convenience not required to run the program. They allow visualizing data from a container and saving or retrieving data on hard disk respectively.

Table 2: Methods provided by Containers.

2.8 Summary and Discussion

In this chapter we have introduced a framework to store and process information from various sensors related to spatial navigation. The presented "data container" allows the robot to dump raw data from any of its sensors into the container. The container autonomously inspects new information and maintains an up-to-date ego-centered snapshot representation of all currently available local information. We store such information in an ego-centered log-polar-based binned occupancy grid, in which the value of a bin denotes the containers certainty of a particular sensor event. Such a representation reduces memory and computation requirements for distal information, yet it represents nearby information with high accuracy.

The container autonomously performs maintenance of its internal spatial data, such as translation, rotation and decay of information. When requested by a place agent (chapter 1) it provides abstract behaviorally relevant information based on reasoning about its internal data, e.g. open space for exploration, an estimate of traveled distances, or a measure of similarity between itself and another container. Such a module in our system that interprets raw sensor data and provides behaviorally abstract information significantly simplifies further navigation processes: computational blocks in our system no longer need to handle each raw piece of sensor data from the robot independently of all others. Instead, the robot constantly maintains a container that represents its current knowledge of its local environment. Upon request this container provides behaviorally relevant information, or creates an identical copy of itself, which represents all spatial information about the current local environment for later processing.

Note that the concept of such a container only works for patches of local limited knowledge. If such a container were to represent large spaces, it basically performed large scale map building - with all the associated problems due to sensor noise. However, for small regions the sensor noise can be effectively corrected of neglected, such that at any point in time the container represents a valid signature of its particular local environment.

Such local log-polar structures as implemented in our containers is uncommon in the engineering robotics community (Thrun 2002), as robotics research so far typically is concerned about creating or using a globally consistent map. Breaking global space into discrete places that each strictly maintains only local knowledge allows using such a data structure. Typically, robotics algorithms dealing with navigation carefully investigate each available sensor and individually develop models to interpret received data as accurately as possible. Here, in contrast, we merge coarse representations of data from various sensors into a common container, yet each represented on an individual layer. This approach simplifies handling data from different sensors: instead of tediously hand-designing fusion models for each pair of sensors, we keep each type of sensor in a separate layer (chapter 2.3), but maintain these layers together (chapter 2.5), such that all sensors contribute to common decisions (e.g. regarding obstructed space).

Advantages of a Data Container

The data container performs low level maintenance and reasoning on raw sensor data. We will later introduce place agents (chapter 1) which provide a topologic representation of larger spaces and control the robot. Such place agents should not have to interpret raw sensor data, but instead operate on more abstract information, such as “free space” or “similarity between representations”. Each such a place agent maintains its local spatial information in a data container, which provides pre-processed abstract spatial information. Using containers on the robot and in place agents, we can approach the navigation problem to some extend decoupled of the underlying hardware. We can e.g. move to a different robot platform with different sensors and keep the navigation system identical; we only need to adapt the data container to the modified hardware.

Extending the data container to other hardware systems is conceptually simple: data from an additional sensor gets represented as a new individual layer. We can imagine multiple robots, each equipped with slightly varying on-board sensors, operating on the same environmental representation: one robot updates data from a group of sensors, whereas another robot updates data from a partly overlapping group of sensors. When comparing two containers for similarity, e.g. the current environmental information with stored information, only those layers get compared that exist in both containers (refer to chapter 2.6). Sensor information that does not exist in the other container does not contribute, and thus does not corrupt the result. However, when requesting information about traversable space from a container that is maintained by multiple robots, such a container has access to all previously recorded sensor data and thus can report obstructions that might be invisible for the current robot. The data structure thus can guide the robot around existing yet invisible obstructions. The exact same argument holds true if one of the robot’s onboard sensors completely fails: this sensor stops reporting new data, so the current container has no information regarding this particular sensor and will ignore it when comparing with previously recorded data - however, the container still remembers old spatial information and takes this information into account when reporting free space. Furthermore, when adding a new sensor to an existing system, old spatial information maintained in a place agent’s containers is still perfectly well suited for navigation, and will incrementally get enriched with data obtained from the new sensor.

The presented container not only easily adapts to modified sensing hardware, but also is flexible regarding computation requirements: the number of bins in a container positively correlates to required computation resources and precision of spatial representation: The higher the number of bins in a container, the more precise the representation of a given environment this container maintains; the lower the number of bins, the coarser the representation. But the same relation also holds for computation and memory requirements: the lower the number of bins, the lower the required memory and computing power required for performing basic operations, such as translation of data or comparison of containers. For a given environment (e.g. fine detailed structures vs. open field) and given computing resources (a PC on a desk is very different to a microcontroller on-board the robot), we can flexibly adapt the data structure. We can also imagine maintaining containers of differing spatial precision for different tasks or different places in the environment: all those requiring high precision get represented by a computationally expensive container with many small bins, whereas places and task that require only low spatial precision get represented in a different type of container with fewer coarser bins.

Throughout this thesis we put our emphasis on representing stationary environments, but we also consider two types of perturbations of a local environment: moved objects, e.g. chairs, and reversible structural changes, such as a surprisingly closed door. Regarding the first, robots today face difficulties deciphering objects in their environments, such that they typically do not detect a moved chair as the identical object at a different place. The container maintained by the robot always represents the current situation, so wherever the chair happens to be the robot's container will report obstructed space. The place agent (chapter 1) representing the chair's local surrounding, however, will build up its spatial knowledge based on multiple fused containers provided by the robot whenever visiting the place. Each such container represents the chair at a different position, so the summation of all containers results in a blurred representation of obstructed space - just as the chair moves. Reversible structural changes, in contrast, are harder to detect and to represent. We only consider a special case in which connections to other place agents are occasionally available or not available. In such a situation the place agent representing space captures multiple different representations in different containers and applies knowledge from the one container that matches best with the spatial signature currently reported by the robot.

Improvements

We have implemented a simple mechanism to maintain data in existing containers when receiving updated sensor information e.g. from a moving robot (chapter 2.5). Instead simple addition and normalization, we can rephrase data maintenance in a Bayesian fashion, taking translated and rotated previous data as prior and current sensor reading as updates. In such a framework we can maximize the likelihood of a container's data to represent the current local environment. We are currently investigating in such a Bayesian optimal representation of data; although the existing simplistic approach serves our purposes well.

The container presented in this chapter currently is implemented for 1d and 2d sensor events, such as a heading direction (1d) or a landmark in a distance and an angle (2d). For simplicity, we currently represent 3d events - such as a target position (2d) combined with a desired target orientation (1d) - by splitting the 3d event into a combined 2d and 1d representation. A wheeled robot on floor only operates in such a three dimensional space, but other robots might have higher degrees of freedom. We can directly extend the concepts presented in this chapter to design a higher dimensional data container, which allows e.g. a flying robot to operate in space by representing 6 dimensions: 3 for translation and 3 for rotation.

3 A Mobile Agent to Explore Unknown Environments

In this chapter we introduce the mobile robot used for acquiring spatial information from real world environments. We design our system such that the robot is a passive actor in the system. The single one currently active place agent (PA_C) sends motion commands to the robot and receives integrated sensor perceptions about the local environment from the robot. The robot does not act on its own, except for triggering primitive behaviors such as obstacle avoidance or emergency backup motion when hitting an obstruction. Later in this chapter we introduce a simulator that allows running a large number of experiments in virtual environments simultaneously.

This is a very technical chapter, which explains details of a robotic system with various sensors that we use to show that the developed navigation algorithm works well in the real world instead of in simulation only. Readers not concerned about technical implementations can skip this chapter and only read the summary.

3.1 Desired Functionality of a Mobile Agent

The mobile agent we include in our system performs only a few basic functions autonomously. It receives motion-commands to be performed from the currently active PA_C, and delivers local environmental knowledge to the same PA_C. In this subchapter we briefly discuss the autonomous functionality.

3.1.1 Maintaining an Integrated View on the Current Local Environment

Software running on the robot permanently receives sensor information from various on-board sensors and constantly adds new information to an instance of the data structure described in chapter 0. This data structure updates existing information (such as decay and translation of old data based on the robot's motion), and provides an integrated best view about the robot's current local environment. Environmental information from within a few times the robot's body length is represented with high spatial accuracy, information from further away with lower accuracy as the data structure internally uses a log-polar based occupancy grid to represent knowledge (chapter 2.2). This particular instance of the data structure maintained by the robot only provides information about current sensory perceptions; it does not represent abstract knowledge about the world such as position of place agents (see chapter 1) or previously explored space.

Software on the robot uses information provided by this data structure for avoiding obstacles, backing-up after hitting an obstruction, and correcting ego-motion errors in path-integration (see chapters 3.1.3 - 3.1.5). Additionally, the currently active PA might request a snapshot of the robot's current environment to update its local environmental knowledge or to guide the robot towards itself.

The robot continually inspects its data structure to spot incomplete information: As most sensors only report perceptions from a limited field of view, knowledge from other regions might have decayed or be severely underrepresented. In such a case, the robot decides to spin 360° on the spot, overruling all other driving targets (chapter 3.1.2) to refresh the current environmental knowledge.

3.1.2 Driving Control

The PA currently controlling the robot regularly provides desired driving targets in the robots coordinate frame, such as “target in 2,00m distance at +30° angle”. Software on the robot computes the required wheel motion to smoothly reach such a target, taking the robot’s current motion and constraints regarding acceleration and de-acceleration into account.

3.1.3 Local Obstacle Avoidance

Occasionally, targets provided by a PA are not reachable on a direct path, as objects in the environment might have shifted or the PA might have a small glitch in its representation of the world. To avoid bumping into obstructions, the robot permanently monitors its internal representation of the current local environment (chapter 3.1.1) and computes a detour trajectory if the direct path is blocked (see Figure 14). Starting from the current position, the robot evaluates possible trajectories found by a breadth-first search (Cormen, Leiserson et al. 1990) on a grid of hypothetical free positions in increasing steps from the robot’s current position. The breadth-first search guarantees that the first path found to lead to the target is at least as short as the shortest possible path. If the robot cannot reach the target (because the target is too distant or currently all potential paths are blocked), it will follow along the path that gets closest to the target, expecting that a better path will show in the future after receiving further sensor perceptions.

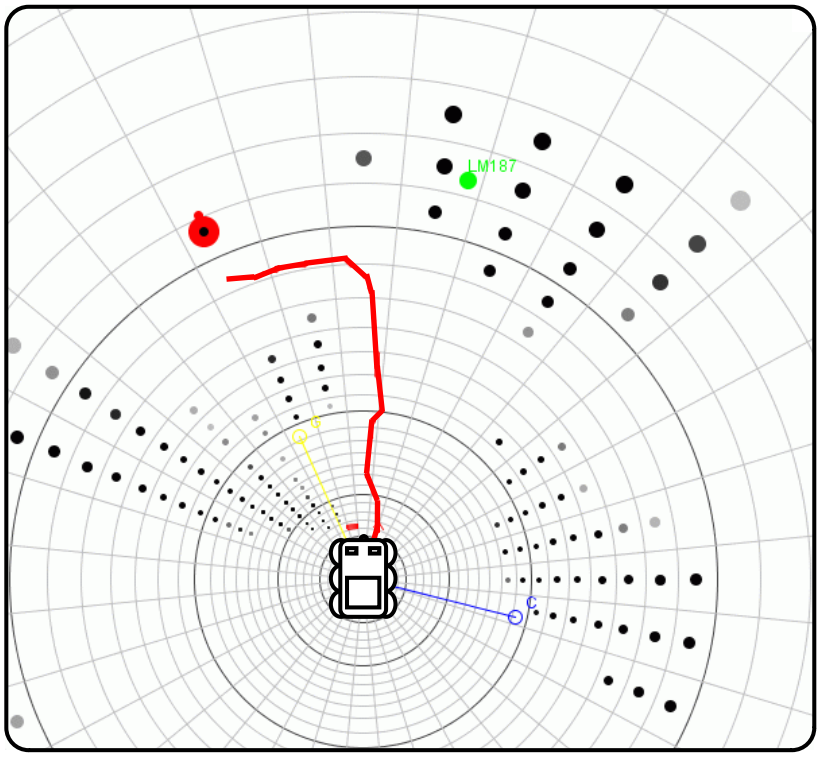


Figure 14: Local obstacle avoidance. The robot received a desired driving target (red circle) represented in its coordinate system. Inspecting the robot’s representation of the current local environment reveals that the direct path towards the target is blocked by obstructions (black circles). The robot autonomously computes a detour to the desired target (red line). Note that the target is beyond a 2m search space; hence the trajectory only gets the robot close to the desired target.

The obstacle avoidance algorithm runs on board of the robot, allowing it to quickly react and continuously adapt the executed trajectory to current sensor perceptions. On average, the algorithm

requires less than 5ms to find a target; in worst case scenarios it only needs about 100ms to find the route leading closest to a blocked target.

3.1.4 Emergency Backup

Rarely the real robot encounters obstructions that it was not able to sense before bumping into them - or only sensed in such a short distance that the detour algorithm described above performs poorly. Few obstructions, such as glass doors or very low steps, are only noticed by an increase in the motor's current consumption when the wheels block, and as such are not available in the robot's environmental representation.

In such a case the robot stops motion instantly, and reports a virtual sensor perception of "obstructed space" at its current position that gets added to the current local environmental representation. After stopping, the robot initiates a backup motion opposite to the direction it went before stopping, which shifts the reported "obstructed space" out of the current position. Restarting the algorithm for local obstacle avoidance (chapter 3.1.3) provides a new trajectory towards the target, which detours the newly encountered obstructed space.

3.1.5 Path Integration

The robot continually updates a 3-dimensional path-integration vector representing distance, direction and orientation traveled. A currently active PA can set and reset this vector, e.g. when directing the robot to a new target. While moving, the robot autonomously integrates desired motion at high update frequencies of about 50 Hz, and corrects accumulated errors using the on-board maintained environmental information (see ego motion error correction, chapter 2.5.3). Computing ego motion on-board allows reacting quickly on changed motor speed. The robot provides path integration information to the currently active place agent, which might use this information to estimate distances towards its neighbors.

3.2 A Real Robot

3.2.1 Motivation

We want to show that our design principle - representing an agent's environment as a collection of independent behaviorally significant active local patches - works well for navigation in real-world scenarios. We can argue and present several graphs of performances of simulated robots, but ultimately people are only convinced when seeing a real mobile agent acting in a real environment, facing all the problems that sensing and acting in the world bears.

When simulating the real world we always face the danger of simplifying problems by providing information or data that is not available in the real world. As an example, a program simulating a robot in a constructed world always knows the robot's true location. Although convenient for debugging and displaying purposes it is dangerous to accidentally miss separating simulator and algorithm, such that the algorithm certainly has no access to such information. Additionally, a "simulated world" is an abstract copy of the real world that usually consists of "boxes", whereas reality consists of tables and chairs with legs that get pushed around regularly. Such an abstraction is an important process of making the world manageable in a simulator - otherwise we can as well use

the real world. However, no one can guarantee that this abstraction keeps the important aspects represented well and only simplifies non-important aspects. For example, in a detailed simulation the four legs of a table might be modeled as individual small “blocked spots”; but in reality they might be metallic and reflecting, such that vision-based sensors hardly see them. Only a very advanced simulator replicates such details; most likely it simply treats table legs as visible obstacles that the robot easily senses and avoids. Furthermore, the real world extends in 3 dimensions, whereas most simulators assume the robot driving on a planar surface and simulate the world in 2 dimensions. Such a simulation avoids problems that robots face in the real world, e.g. hitting a table plate with tall onboard electronic, despite all sensors reporting free space ahead underneath the table. Yet another aspect is sensor variance based on external influences: a vision sensor such as a camera typically reports very different images in bright sunlight compared to inside a dark room. In the real world a robot has to face such changes, as window blinds might be open or close and artificial light might be turned on or off, depending on time of day. A simulator typically does not reproduce such changes. All these real-world problems are not directly related to the algorithm we developed; however, they’re challenging problems and need to be addressed for real-world performance. Our algorithm needs to be tolerant with respect to these kinds of changes.

On the other hand, a simulator also provides advantages, especially during the development. A major advantage is that initially one does not have to care about the above “real-world” issues. While developing the concept, we do not want to worry about a robot getting stuck and possibly damaging itself; we simply want it to move and collect data. Another strong advantage is that we can replay a particular scenario several times, receiving identical sensor readings to tune parameters - or to detect improper behavior. Thirdly, a simulator optionally provides sensor data at higher frame-rates compared to real sensors, allowing speeding up execution of the algorithm by a large factor. Furthermore, we can execute multiple instances of simulators and algorithms under investigation on multiple computers simultaneously, collecting a larger amount of data compared to a single robot in the real world. Using such a simulator reduces waiting time during development and simplifies tuning the algorithm significantly.

Seeing advantages and disadvantages of both, real robots and simulators, it is clear that we want both to initially develop the concept and ultimately prove it working in real-world environments. In this chapter we will first present our developed robot, its actuation, its on-board sensors, and discuss its limitations. In the second part of this chapter we will present a developed simulator that is mimicking the existing robot in an abstract 2d world that allows simple tests.

3.2.2 Requirements for a Real Robot

We had various requirements in mind when thinking about a robot to explore our navigation ideas in the real world: mainly regarding long during battery live (overnight continuous exploration), extended range of action (at least within the whole institute of roughly 100x30meters) and a variety of different on-board sensors. Looking at these desired features we realized that we either have to purchase a huge and expensive robot that is potentially dangerous for humans; or chose a medium sized mobile robot and customize it extensively. Especially the option of running long autonomous explorations without safety concerns convinced us to work with a medium-sized mobile robot and investigate a significant amount of time in customizing it.

Conceptually, the mobile robot shall only receive simple motion commands (eg. a nearby target location in terms of direction and distance), and in return continuously report sensor data. We do not consider the robot an “intelligent” agent in the system, but rather a passive device that collects information from various places in the environment or performs an elementary task at the current place (such as recharge batteries). The robot itself does not understand the environment’s spatial structure, but it is guided by place agents (see chapter 1) within space.

However, the robot directly performs low-level (reactive) behaviors, such as avoiding obstacles. If it is told to drive forwards 1m but its on-board sensors report blocked space in front, it tries to compute a detour based on its current best knowledge of the local environment (refer to chapter 2.5.4). Also, if it accidentally bumped into an obstacle or got stuck (indicated by the motors’ current consumption), it reacts by backing-up and waiting for new commands. Nevertheless, all the robot’s actions are purely reactive based on external stimulus, without further knowledge about the environment.

Note that the robot continuously provides updated sensor information about its current environment for the currently active controlling node (a place agent, chapter 4X), such that this agent can update its knowledge and its perception about the world.

This chapter describes all details of the mobile robot: starting from its basic hardware setup, continuing to all on-board sensors and the kind of data they provide up to the parts of software that run directly on-board of the robot preprocessing data for speed reasons.

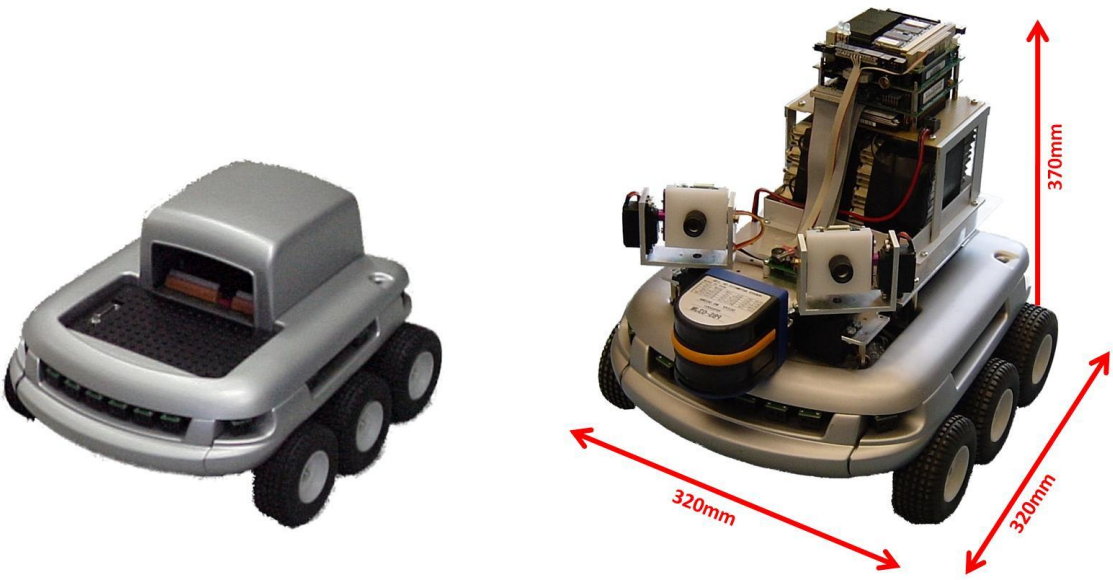


Figure 15: Koala Robot. Left: factory-configuration, as sold by K-Team, SA. Space under the gray “hood” in the back and underneath the black metal cover plate in the front is free for custom add-ons. Refer to Table 3 for technical specifications. Right: final modified version including various sensors (front), a large battery pack and on-board computation (back).

3.2.3 Robot Base

We have good previous experience with mobile robots from the Swiss company “K-Team” (www.k-team.com, Lausanne), so we decided to purchase a mid-sized mobile robot type “Koala II” (see

Figure 15 left, Table 3). The robot is 320mmm long and wide, can carry several kilograms of payload, and offers an easy to use command interface via serial port. It has two independent motors that each power three aligned wheels on either the left or the right side of the robot's body. This configuration essentially allows steering the robot like a tank:

I. Both triplets of wheels rotating in the same direction at identical speed results in forward or backward motion with a resulting velocity proportional to rotation speed of wheels
II. Both triplets of wheels rotating in different directions at identical speed generates rotation on the spot, again proportional to angular speed of wheels
III. Both triplets of wheels rotating at different speeds and possibly different directions results in various curved trajectories, which can be computed as a superposition of translation and rotation (cases I. and II.)

Although these options for motion seem quite rich, the robot is not holonomic: it cannot translate sideways. In order to translate perpendicular to its main body axis, it needs to perform a 90 degree rotation on the spot, translate forwards, and counter-rotate to face the previous direction. Such functionality is desirable when a control algorithm determines a target sideways of the robot. Conveniently, the Koala robot is designed such that getting stuck during rotation on the spot is extremely unlikely to happen, as its body length equals its body width.

Looking at the robot's technical specifications (Table 3) it is immediately clear that the on-board computational resources are too limited to perform substantial computation. We decided to add external computing resources on the robot (see Chapter 3.2.5) and use the robot's build-in CPU only to execute elementary driving commands.

On-board computation	Motorola 68331 @ 22MHz, 1MByte RAM, 1MByte ROM
Communication	Serial port command protocol up to 230.400 bps Various digital IO lines, 6 analog inputs
Motion	2 DC brushed servo motors with integrated incremental encoders (roughly 19 pulses per mm of robot motion)
Speed	Max: 0.38 m/s using factory default PID speed controller Min: 0.005 m/s using factory default speed controller
Acceleration	Max: 0.075 m/s² using factory default PID speed controller
Slope traversal	Max: 43 degrees
Ground clearance	30mm
Sensors	6 Infra-red proximity and ambient light sensors Motor torque and global power consumption
Size	320 x 320 x 200 mm length x width x height
Weight	3.6kg without battery
Autonomy	Between 4-6h (unpowered equipment)

Table 3: Koala Robot Specification as from K-Team, SA (www.k-team.com)

3.2.4 Power Supply

Similar to the on-board computation, the standard supplied power option (NiMH battery pack of 12V, 4000mAh) is too limited to keep with our requests regarding running time. We decided to advance to Lithium-Polymer (LiPo) battery technology that currently provides the highest energy density per weight. We supply a large stack of 3 battery packs connected in series that each contains 10 batteries in parallel. Such an assembly provides a total of 11.1V, 56.000mAh, at around 3.5kg in a

box of roughly 160x280x100mm. When fully charged such LiPo batteries provide about 4.2V, when almost empty they discharge to slightly below 3V, resulting in operating voltages from 12.6V to about 9V. This variation in operating voltage is larger than for standard NiMH batteries, therefore we decided to use active switching capacitor voltage regulators (Traco Power, DC/DC converter), to generate stable 12V for the Koala robot and stable 5V for various additional sensors described below. Such switching capacitor regulators achieve a significantly higher efficiency (typically around 85%) compared to standard voltage regulators that convert extra energy in heat. This increases running time of our robot significantly; our current estimate (although not yet verified) is that the robot can operate around 10h autonomously.

3.2.5 On-Board Computation

During initial discussions of our algorithm for navigation it became clear that a reliably prediction of required computing power will be very difficult, if not impossible, until very late in the project. Therefore, we decided that most of the required computation shall happen on standard computers off the mobile robot, with a wireless link for data exchange between PCs and our robot. However, preprocessing raw sensor data needs to happen on the robot as we cannot provide a real-time high throughput wireless link to transmit large amounts of raw sensor data (such as video images at high frame rates) to PCs. We quickly realized that the on-board computational resources that Koala II provides (see Table 3) are non-sufficient even for simple preprocessing, so we investigated in external hardware that provides additional ob-board computing power.

We decided to implement an industry-standard PC104 computer with 800 MHz Crusoe processor and 256Mbytes of internal RAM, running a Linux operating system. Such a configurations allows a simple wireless link (WLAN, wireless Ethernet at up to 54Mbps data rates) that is available everywhere in the institute to exchange data between programs running “on the robot” and programs running “on various external PCs”. Additionally, such a standard on-board PC offers a broad range of extensions that we can use to interface to sensors on the robot. In our setup, the PC104 is equipped with the following extension boards:

- A WLAN 802.11g 54Mbps wireless Ethernet interface card
- A serial port extension board providing additional 4 serial ports (a total of 6) to communicate with on-board sensors (refer to Chapter 3.2.7).
- Two video frame grabber boards to continuously capture video images from stereo pan-tilt cameras (refer to Chapter 3.2.7.1).

Initially, we designed the system without moving parts (no fan for the low-power Crusoe-CPU, no hard-disc but Flash memory instead) such that the robot’s motion wouldn’t cause damage to these parts. Unfortunately, at the time of system design we did not manage to get the flash-based storage working properly, so we are running the system from a laptop’s 20 GB hard disc mounted on the robot. We have not seen performance problems or an unusual large number of bad sectors on the hard drive. Having 20 GB total disc space in contrast to 2 GB flash memory also allows recording much longer traces of data.

3.2.6 Communication Interface

The communication interface between the robot’s on-board PC and external PCs running the main software is a TCP/IP based protocol over our institute’s wireless local area network (WLAN 802.11g),

providing guaranteed packed transmission on the cost of non-guaranteed timing. However, under practical considerations the timing delays in a fast network are in the range of milliseconds and cause no problem for transmitting information between the robot's PC and external PCs. The main advantage of using WLAN is that our whole institute is covered, so the robot can explore large environments.

We implemented a simple human-readable protocol for exchanging messages in which each data line consists of a timestamp in millisecond resolution, followed by a string to identify the type of information and optional parameters, terminated with a "Carriage Return" <CR> character. Examples of such are listed below:

[1120790233:597]:COMPASS A= 35.000<CR>

[1120790233:597]:SETTARGET D= 2.000, A= -30.000, O= 0.000<CR>

The first line shows data from the on-board compass reporting the current magnetic north at 35 degrees in robot centered coordinates, whereas the second line sets a desired driving target send to the robot's on-board PC104, directing it to a target in 2 meter distance at -30 degrees. The protocol is simple, highly flexible, and easy to extend for further sensors. It is not optimized for throughput, but instead for clarity and convenience. We can easily record transmissions in a file and inspect or even replay recorded data to investigate in case the system behaves in unexpected ways.

3.2.7 On-Board Sensors

The robot's on-board PC104 needs to communicate with all on-board sensors and with the robot. The connection to the Koala robot happens through a standard serial port interface, allowing transfer of data at up to 115.200 baud. But only one of the sensors we decided to add (an expensive commercial LED range finder, see Chapter 3.2.7.2) also offered a serial port to connect to the PC104 directly. In contrast, most typical sensors for mobile robots, such as a magnetic compass or a gyroscope, do not directly plug into PC104, so we had to develop interface circuits. We build several copies of a small electronic board containing a microcontroller that communicates with PC104 over a serial port, and offers flexibility to interface directly to a single sensor, e.g. through A/D converters or pulse-width-modulated (PWM) signals. This approach allows extending PC104 with a larger number of serial ports and adding sensors independently of each other, thus increasing the overall tolerance against failures.

Refer to Figure 16 for a sketch of the on-board communication between PC104, robot, microcontrollers and sensors. A detailed discussion about the available sensors happens in the following chapters.

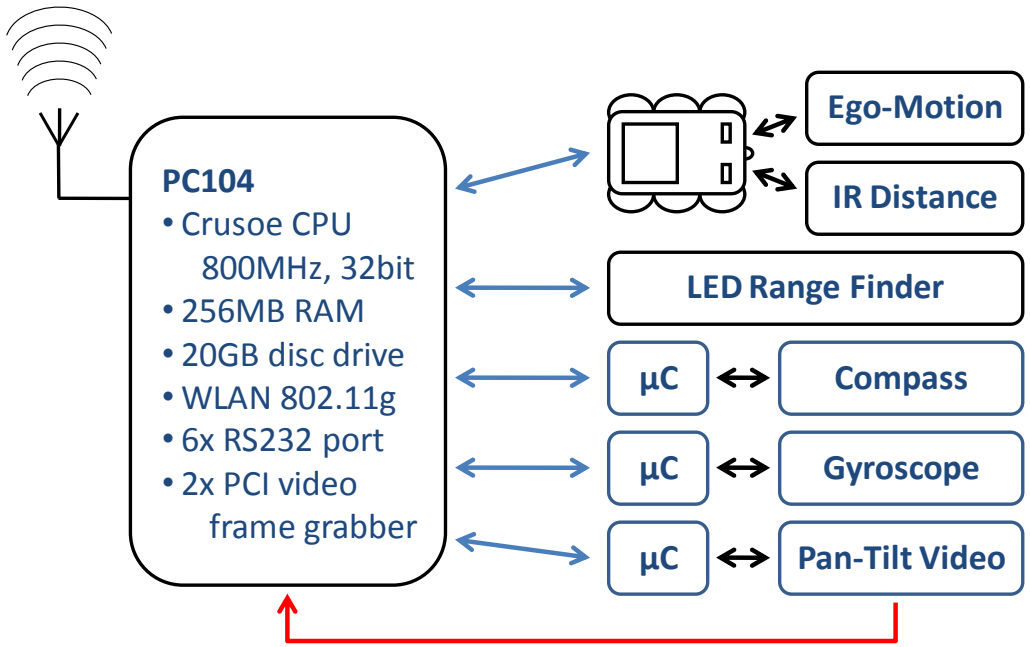


Figure 16: Overview of on-board computation, sensors and communication channels. All existing on-board sensors are shown on the right side. Blue arrows indicate bidirectional serial (RS232) ports; black arrows are specialized connections depending on sensor type. The red arrow at the bottom shows analog video signals from cameras to on-board video capture cards (frame grabbers). All blue boxes indicate own hardware development, whereas black boxes show purchased components.

3.2.7.1 Vision

Vision is a rich source of information about the local environment. In fact, humans can easily navigate on visual information only, whereas we have to move much more carefully when relying on auditory or tactile information solely. However, vision also is very complex to decode: In the human brain large areas of cortex are devoted to processing visual signals (Kandel, Schwartz et al. 2000). This thesis is not concerned about vision: we are not investigating in techniques that allow a mobile robot to interpret visual information and extract relevant information for navigation. Researchers have spent decades on this problem and it is yet far from being solved. Still we would like to add some sort of 'vision-capabilities' to our robot, enabling it to obtain richer information from larger distances as compared to most sensors described below. We decided to go for a compromise: It is reasonable to assume that in the very near future computer vision algorithms will be sufficiently advanced to detect and label particular objects in snapshot views, such as a "red sofa", a "gray desk", or a "green flower" (Evolution-Robotics 2008). Giving such labels to visual blobs does not imply that the vision system understands these objects: it does not know that a sofa is a place humans use to relax or read a paper. It only assigns a label to that object; such a label could as well be "a red blob with a size of roughly 2x1x1m" for the sofa. All we demand from our vision system is labeling an object with the same label when seeing it again. Note that the robot might recognize multiple red sofas in different rooms, which will all receive the same label. But also humans typically rely on context information (top-down feedback) when distinguishing multiple red sofas from one another, e.g. knowing in which room a sofa is.

For our navigation system, we reduce the computer vision problem of object detection in complex scenes to assigning a repeatable label to significant visual patches. Still, this is a challenging problem that we do not want to address. Therefore, we decided to place easily recognizable stickers at various places in the environment (see Figure 17). We determine these places by human visual

inspection of the environment, searching for places of large uniquely colored blobs, such as desk drawers, file cabinets, or doors. Each of these places receives a sticker that has an easy-to-detect red-blue outside border and a unique black-white binary pattern on the inside (see Figure 17, top left). We chose these colors because they turned out to be most insensitive to changes in ambient light in the video images. The outside red-blue pattern allows a program running on the robot's PC104 to detect stickers in video data provided by on-board video frame grabbers. Once position and extend of a sticker within a snapshot image is determined, the algorithm parses and decodes the inside binary pattern of this sticker, decoding a 12 bit identity and a 6 bit checksum. If the checksum verifies the binary pattern, the visual preprocessing algorithm knows about an object in a given angular direction, depending on the sticker's location within the camera image.

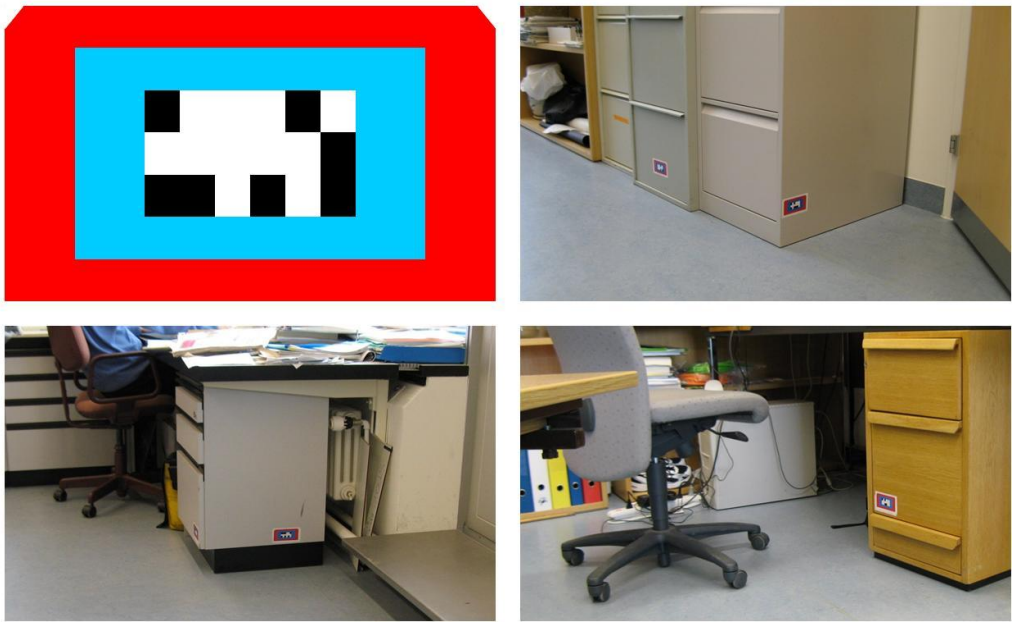


Figure 17: Top left panel: a "landmark" sticker, original size 90x45mm, with easy-to-detect red-blue outside boarder and binary-coded identity inside. Remaining panels: examples of objects labeled by landmarks, which all have a large area of unique color.

Initially, we planned to use stereopsis to determine distances to objects: A second camera mounted at an offset of 100mm records a second image of the same scene, in which we search for the same sticker once it is detected in the primary image. The relative spatial distance between occurrences of a sticker in the two images provides distance information: the more offset in the two images, the shorter the distance between cameras and sticker. We need a large range of overlap between the two images to increase chances of finding the sticker in both images and receiving a precise distance estimate. Initial tests, however, showed poor performance for the following reason: we used a tele-lens to have a sufficiently high spatial resolution to detect and decode stickers reliably in reasonable distances of a few meters. Currently, the cameras detect stickers in up to 5m distance. However, using tele-lenses for the cameras significantly limits the field of view, in our case to below 12 degrees; chances of seeing stickers with such a small field-of-view are tiny. We implemented a pan-tilt system providing rotation of the camera around pan and tilt axes (refer to Figure 18) covering about 180° horizontal and about ±30° vertical field-of-view by taking multiple images over time. However, sufficient overlap of both images only occurred at up to 120° horizontal view due to image occlusions by the other camera, limiting stereopsis to that range. Furthermore, the rotational joints

introduce uncertainty in viewing angle: only a few degrees due to mechanical slack in the gears yield distance-errors of several meters. Seeing such poor distance estimates we decided to implement a different principle to extract distances from visual information. Given that we know the true size of the stickers, the apparent size of the stickers in the video image represents distance: the further away the smaller the appearance in the image. Humans heavily rely on the same principle, e.g. when estimating distances to a car.

Despite not needing two cameras for distance estimation we kept both cameras on the robot active, which doubles the searched area per time: each camera independently scans a horizontal field-of-view of 150°, with about 120° of overlap between the two cameras (see Figure 18 inset). The upper servo that generates tilt motion turns the cameras only slightly about ±30°, as we placed all stickers roughly on horizon level. The on-board algorithm controlling both cameras keeps track of detected stickers and their positions: While they stay in a camera's field-of-view, each camera periodically returns to that direction and searches again to refine the sticker's position estimate. Stickers are relatively sparse in the environment; therefore it is important to initially scan and search for them, but also remember and re-fine detected instances.

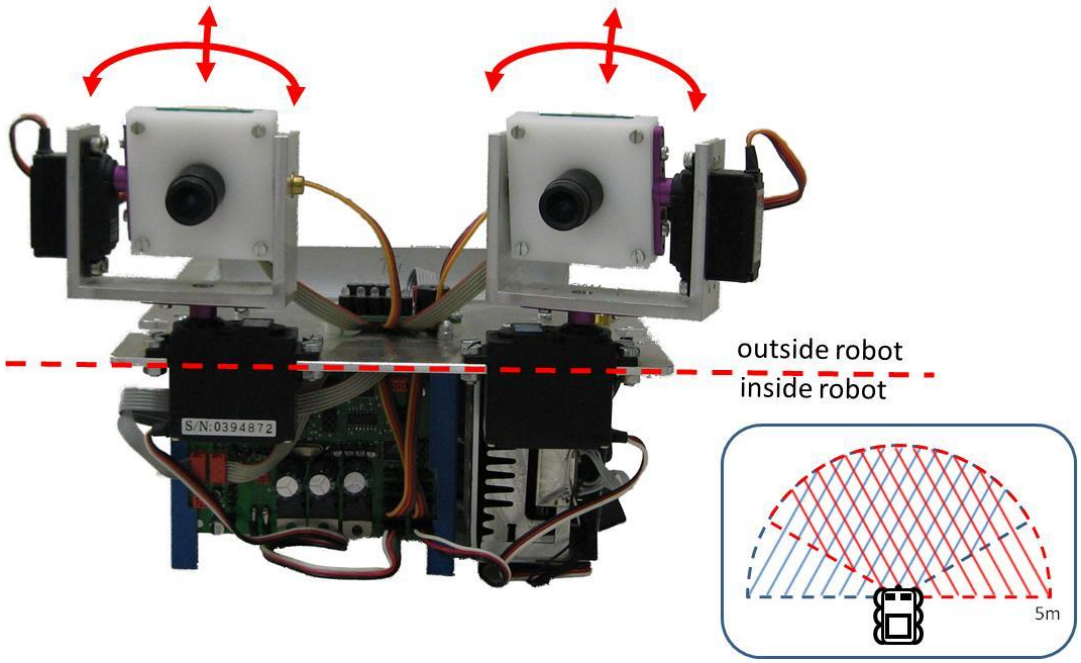


Figure 18: The on-board pan-tilt video system removed from the robot. Left and right video cameras get rotated by two perpendicularly mounted servo-motors. Red arrows indicate direction of rotation. Inset shows 180° field-of-view for both cameras, left (blue) and right (red), with about 120° of overlap.

Although the encoded identity of a sticker is unique, we translate this identity to an object name, such as "red sofa". This removes uniqueness of visible objects in the world, as multiple sticker identities map to the same object "red sofa". We call such a non-unique identity a landmark, and transmit a triplet consisting of landmark-identity, distance and angle to the remote PC running our navigation algorithm. For simplicity, we call the object identities "LM<number>" instead of "red sofa".

Note that the time-consuming visual pre-processing of images happens in the robot's on-board PC104, which significantly reduces the amount of data transmitted over the wireless link: only landmark-identity, angle and distance of each detected landmark get transmitted, instead of full images that might or might not contain landmarks. The on-board PC104 receives images from each camera at around 5Hz, resulting in 10 images per second to search for landmarks. Transmitting 10 color images at a resolution of 640x480 pixels per second over wireless link to another PC for processing is possible, but it limits data transfer for other sensors and often introduces significant delays. Analog wireless video transfers images in real-time, but provides significantly lower image quality and often suffers from interference with other wireless systems, such as WLAN. Therefore, we decided to process video images on-board and only send pre-processed data (landmark identity, angle and distance) to the off-board PCs.

3.2.7.2 Distance Sensing

The robot has two different types of on-board distance sensors: an LED range finder and multiple IR reflectance sensors. This chapter will briefly introduce both these distance sensors.

The first sensor (see Figure 19, left, and Table 4) estimates distances to objects located from within several cm to a few meters. It emits a single directed light beam (invisible infra red light, wavelength 880nm) and measures time until the light's reflection from an object is received at the sensor. The time past is proportional to distance traveled, therefore the LED range finder can estimate distances to reflecting objects. As the time needed for light traveling within a few meters is very short, the sensor applies various tricks to achieve higher resolution (Alwan, Wagner et al. 2005), such as sensing interference patterns of light or modulating the light's phase. Inside the sensor a small disc spins at 10 Hz, rotating both the emitting LED and the receiving sensor. Such a setup records distances in 100 directions within a 180° forwards field-of-view. The rear 180° are covered by electronic and support structure, blocking the sensors outreach. The minimal distance reliably measured is 20cm; the longest guaranteed distance in which objects are received is 3m (Figure 19, right). However, in our experiments the LED range finder typically reports objects at distances up to 5m (Figure 20). Note that the sensor is limited to operation within a horizontal plane: mounted on the robot it only detects objects close to floor level. It will not detect elevated objects (as e.g. a table-top), nor will it detect see-through objects reliably (such as a door with a large glass inset).

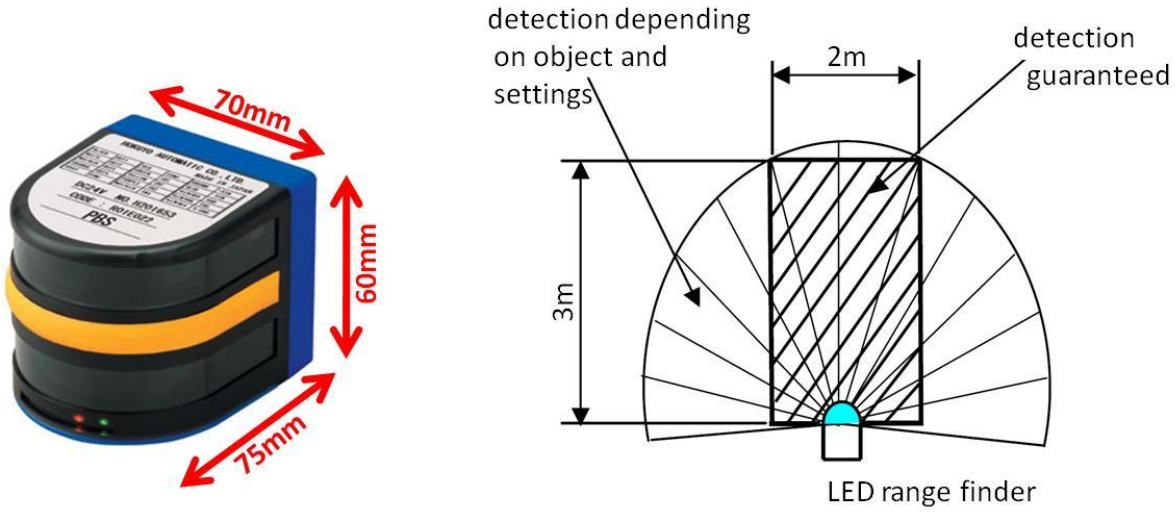


Figure 19: Left: Hokuyo LED Range Finder model PBS-03JN. Right: top-down schematic view of guaranteed operation range (from Hokuyo data sheet (Robotshop-Canada)).

Manufacturer and Model Nr.	Hokuyo Automatic Co. LTD, Japan; PBS-03JN
Operating Range	180°, min. 3m long x 2m wide rectangle
Operating Frequency	10 Hz
Detection Precision	±10cm below 1m, ±10% above 1m distances
Angular Resolution	100 scanning directions in horizontal field of 180°
Operating wavelength	880nm
Size (l x w x h), weight	75 x 70 x 60 mm, 500 gram
Interface	RS232 (serial) port at 57,6kbps, 7N1, no flow control
Power Supply	24VDC, 250mA max

Table 4: Specifications of Hokuyo PBS-03JN LED range finder

The LED range finder is a complex sensor that already implements signal preconditioning. It communicates with the robot's on-board PC104 through a serial port, providing a block of 100 distance estimates for every rotation of the spinning disc. For an evaluation of the sensor's accuracy refer to (Alwan, Wagner et al. 2005).

Many traditional robots used for navigation employ laser range finder (LRF) that work use a laser instead of the LED for emitting a thin beam of light. Such LRF are more precise compared to the Hokuyo LED range finder; however, they are also significantly more expensive, larger and heavier. Here, in contrast to most engineering approaches for spatial mapping, our primary concern is not accuracy of distance estimates. Our navigation algorithm stores sensor data in a binned representation (see Figure 20, and Chapter 2.2), which does not maintain high spatial precision. We are preliminary concerned about extracting a characteristic signature of features in the local environment, which is already possible with lower-precision distance information.

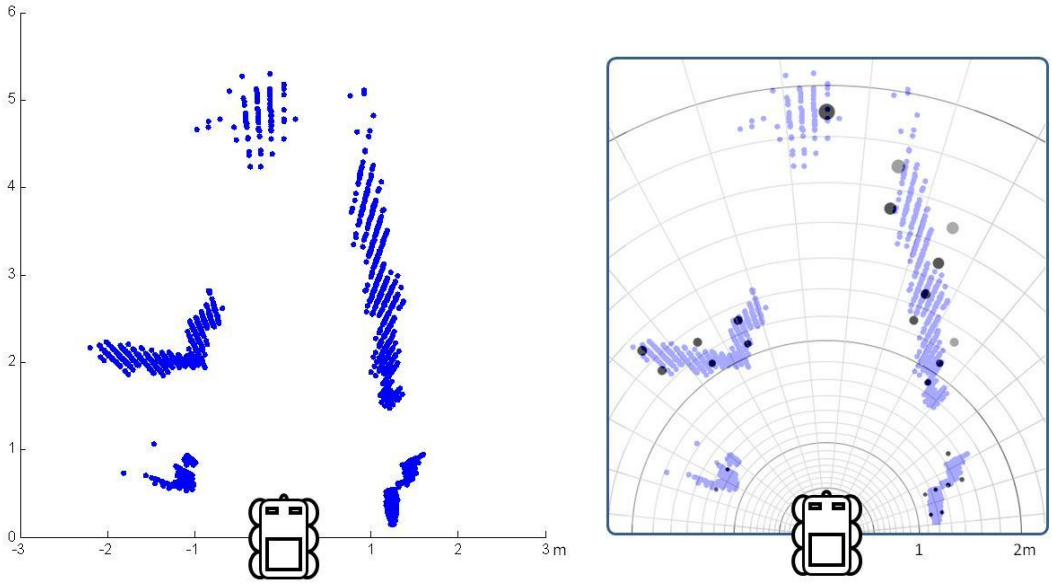


Figure 20: Distance estimates from the robot's on-board LED range finder. Left: 5 seconds of data (corresponding to 50 vectors of 100 data points each) recorded from a stationary robot. A large open space in front of the robot and two open passages to the left and right are clearly visible. Right: The same data stored in a binned representation (original blue dots superimposed). Black dots represent occupied bins; the darker a dot the more certain a bin is blocked.

The second kind of distance sensors also emits infra red light, but measured the brightness of reflected light instead of time of travel. The reported brightness strongly depends on reflectance properties of the objects reported, as shown in Figure 21. In prepared environments, such as a room designated to robot-experiments, the reflectance properties of walls and obstacles can get well characterized a-priori, so that these sensors report reliable distance estimates. However, in unprepared environments such as our institute, different objects have vastly different reflecting surfaces. Therefore, this sensor is only little suited to estimate precise distances to objects for us. However, it is still useful as a sensor to detect obstacles blocking the way that are not detected by the LED range finder.

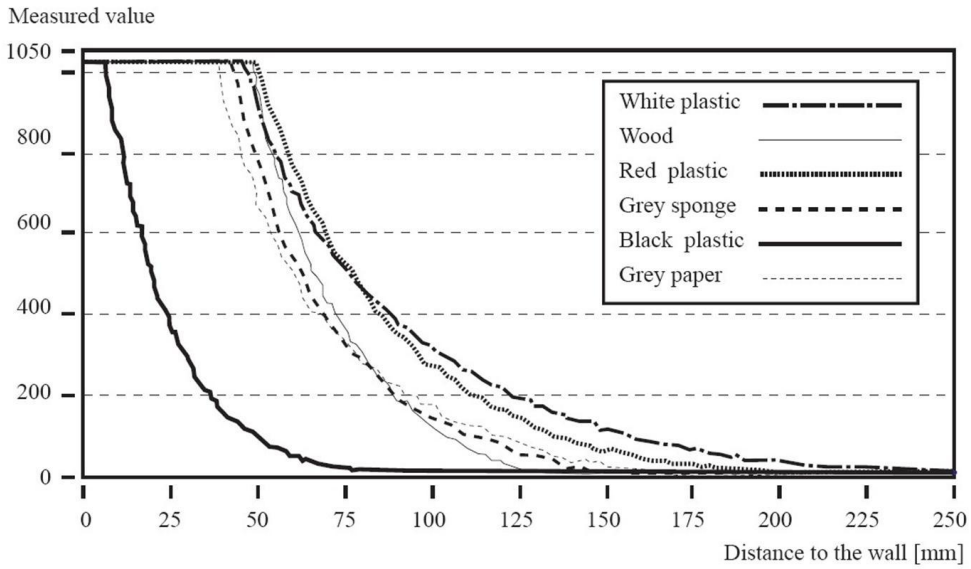


Figure 21: Reflectance values sensed by IR distance sensors for various materials at given distances. Note that white plastic in 150mm distance returns the same value (around 100) as black plastic in only 40mm distance. Figure taken from (K-Team-SA).

The base robot as provided by K-Team contains 16 such IR-distance sensors, arranged around the robot. Our office environment, however, contains several obstacles that are too low to get detected neither by the standard IR arrangement nor by the LED range finder, such as legs of office chairs. We decided to remove the six sensors originally facing left and right of the robot and reposition them as a second row of sensors facing forwards, but very close to floor level (refer to Figure 22). We are using all 14 sensors facing forwards and the two sensors facing backwards to report "emergency stop" events only. Each sensor either report open space or a large "blob" of blocked space with imprecise distance for its particular angle. Having such information, the robot can backup and detour around the blocked space.

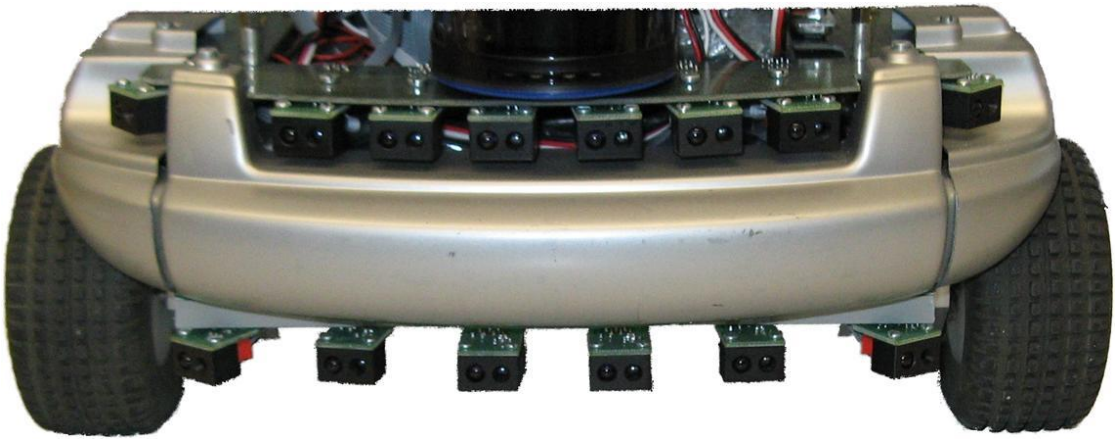


Figure 22: Front view of robot, showing IR distance sensors. Each sensor is a black box with two circular openings for transmitting and receiving IR light. Note two rows of IR distance sensors on the robot facing forwards: top row 8 sensors, bottom row 6 sensors.

3.2.7.3 Orientation Sensing

Two different on-board sensors report the robot's current heading direction. One sensor (Honeywell HMR3300 magnetic compass) measures the magnetic field that surrounds the robot, reporting the

direction towards the earth's magnetic north as seen in the robot's perspective. When rotating the robot's angle towards magnetic north changes, such that it can infer its own rotation relative to ground. However, as the earth's magnetic field is very weak, the sensor also strongly amplifies surrounding magnetic signals, including those from power cables. Initially, we mounted the sensor directly on the robot which turned out to be too close to the floor: All heading direction data was strongly influenced by electric power cables in the floor of the institute (see Figure 23, blue trace). Ultimately, we added a 1m long carbon-fiber tube to elevate the sensor significantly above floor. The red trace in Figure 23 shows the sensor's readings when rotating on the spot. Adding such a carbon tube extends the height of the robot significantly which potentially causes problems when e.g. driving underneath desks. No on-board sensor detects that the magnetic compass stops at a table's desk. Therefore we read the magnet sensors integrated accelerometers to detect the board's tilt level. If the tilt exceeds a small threshold, we conclude that the magnet sensor got stuck and report a blob of blocked space in front of the robot.

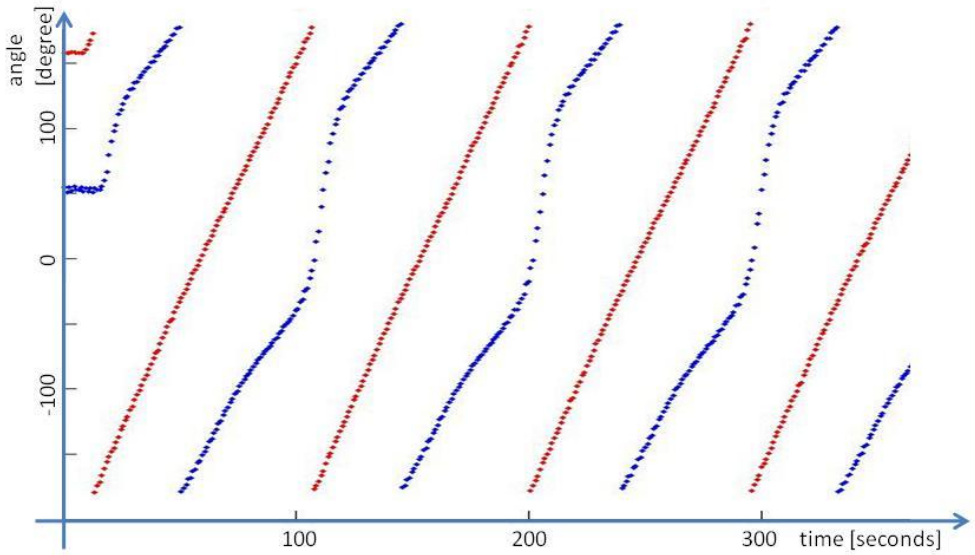


Figure 23: Magnetic angle (-180° to +180°) for continuous rotation on the spot. After power-up, the robot slowly rotates with constant angular speed. The blue trace shows data from a sensor mounted on the robot close to floor level, whereas the red trace shows sensor recordings from a sensor elevated 1m above floor level. Note: both sensors mounted with 180° offset.

The other sensor reporting a heading estimate is a gyroscope (Silicon Sensing, UK; model CRS03-02). This sensor reports an analog voltage proportional to the change in orientation, up to 100°/sec. We are not interested in the current rotational speed but the heading angle; therefore the microcontroller connected between PC104 and the gyroscope integrates all signals from the gyroscope, resulting in an estimate of the current heading angle. Such integration drifts over time, as the correct baseline is unknown or might change with temperature and mechanical stress (see Figure 24). Therefore, this sensor needs a periodical reset to report the correct heading direction.

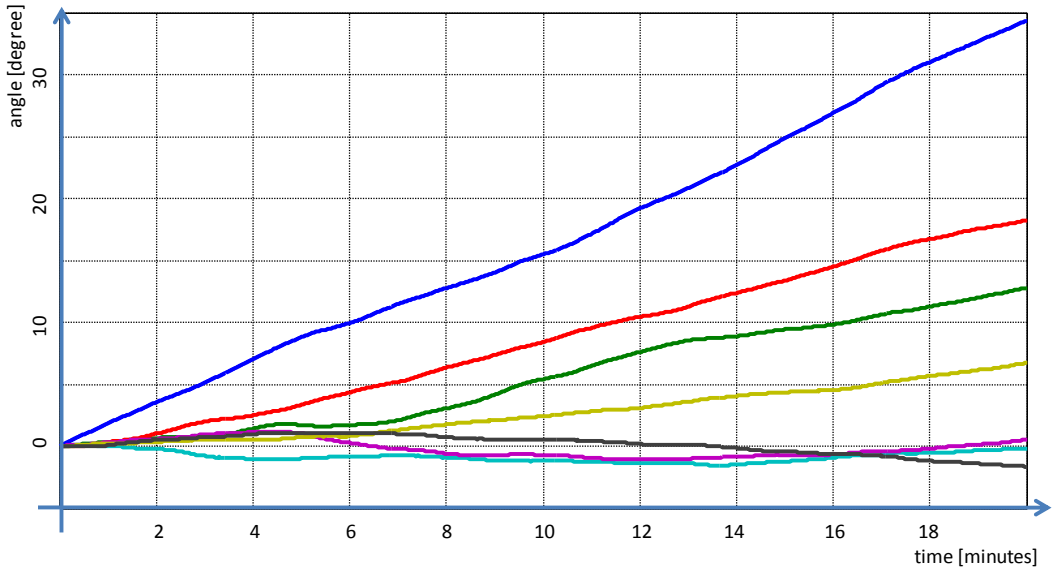


Figure 24: Estimated heading direction recorded from a stationary gyroscope during 7 consecutive recordings of 20 minutes. Note the angular drift over time: upper traces correspond to initial recordings; lower traces got recorded later. Each recording initially captured the gyroscope's current output as baseline. The gyroscope has reached its final working temperature after about 1.5hours, indicated by more constant heading angles. Note that this is data from a stationary gyroscope; a gyroscope in motion shows significantly increased drift.

3.2.7.4 Ego-Motion Sensing

The mobile robot allows estimating ego motion by monitoring wheel encoders attached to the left and right set of wheels. These wheel encoders report incremental steps of wheel rotations, corresponding to 0,045mm motion on the outside of a perfect wheel. Integrating these signals at high frequencies reveals the trajectory the robot followed, assuming perfect wheels, a perfect surface and ideal friction between wheels and surface. On a real robot, errors accumulate quickly and the trajectory needs to get reset periodically.

The data structure introduced in chapter 0 cannot directly represent a trajectory in a meaningful way, so we preprocess information from the wheel encoders and convert the decoded robot's motion into changes of a virtual target's position. During exploration in unknown environments, the robot remembers its starting position as virtual target and moves away from it. In contrast, while it is purposefully moving between two known places, the starting place provides its best estimate of the position of the target place, which the robot internally sets as the virtual target's position. So in both cases the robot maintains an internal vector to a target position that it updates based on signals from wheel encoders. Whenever it starts a new elementary trip the target position is reset within the robot's current frame of reference, thereby discarding previously accumulated error.

Note that the robot's wheel encoders report true wheel motion, thus automatically taking motor errors into account due to fluctuations in battery voltage, slack in gears, or inertia effects for acceleration and de-acceleration. However, the wheel encoders cannot sense wheels slipping over floor or losing ground contact.

3.3 A Simulator

We can substitute both, the robot and its natural environment, by a simulator in software (Figure 25), without the main navigation algorithm realizing that it is exploring a virtual environment. During development periods of the Ph.D. project such a simulator allowed monitoring and recording a large number of repeatable trials easily for tuning parameters of our system and evaluating its performance. In contrast to the real robot we can run the simulator unattended without facing the danger of severely breaking itself, and we can run several instances of explorations with a separate simulator for each on multiple computers. The simulator allows time-warping, thereby reducing time to explore large environments or replaying sequences slower to investigate closely in unexpected results. The simulator offers logging position of the simulated robot in a file for later debug, or alternatively recording snapshots of the robot in its environment in regular intervals to create movies of the moving robot.

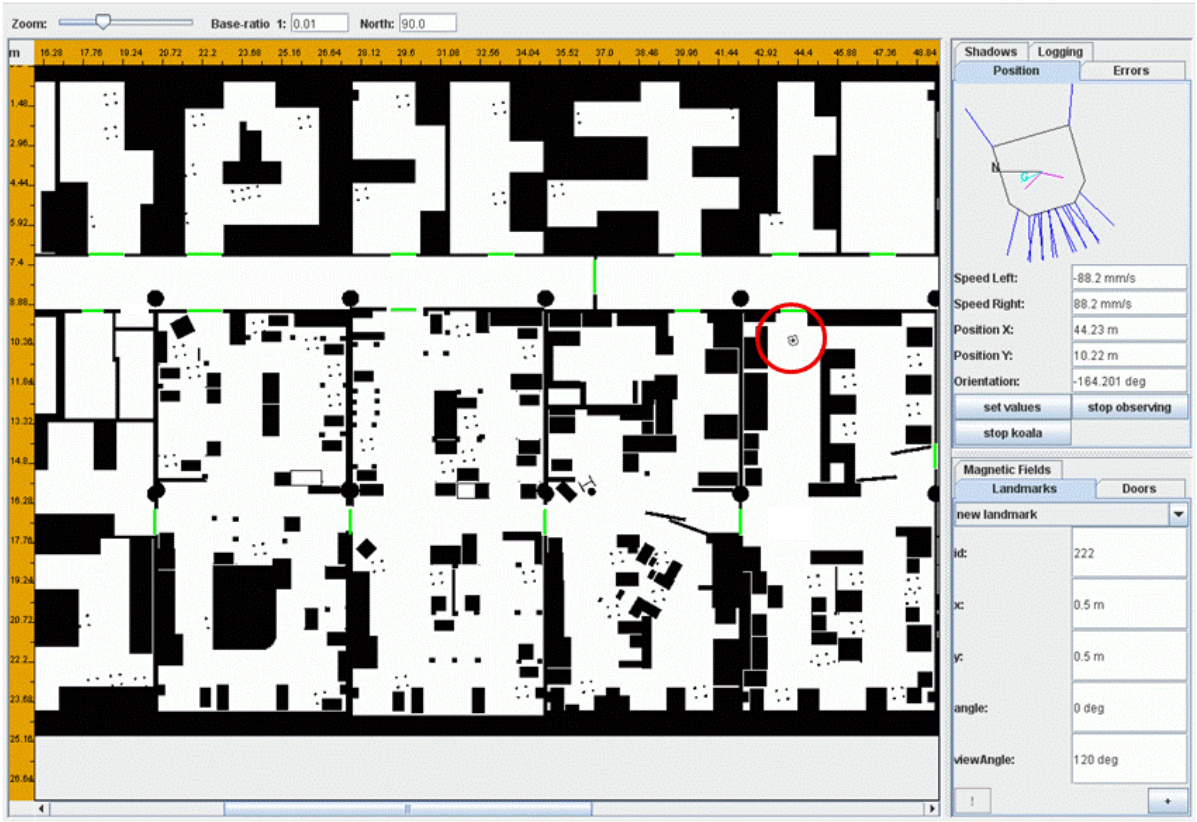


Figure 25: A screenshot of the software simulator in operation. The red circle indicates the robot's current position in its simulated environment. The panel at the top right shows status information about the robot (including its simulated true position in the world), the panel at the right bottom shows information about objects in the environment such as landmarks or doors. Note the different spatial details of the environmental representation in the upper and lower rooms, leading to a sparser or denser coverage of space with place agents respectively.

The software simulator was programmed as a Masters-project in the HSR Rapperswil by Thabo Beeler during nine months of work in close collaboration with us. The simulator mimics the physical robot closely in terms of available sensors and their respective uncertainty; but also simulates random motion errors and distorted environmental stimuli. A black-and-white 2-dimensional plan-view image of an environment with a resolution of 1cm per pixel serves as spatial input to the simulator, with black pixels denoting obstructed space and white pixels traversable space. Green colored bars indicate doors that we can open or close at runtime, possibly depending on a particular

task. For any such imported environmental map, we can add landmarks (chapter 3.2.7.1) and local distortions of the simulated magnetic field on a comfortable graphical user interface. The simulator saves a created artificial environment on hard disc that we can reload to start a new exploration run on identical initial parameters.

A TCP/IP connection provided by the simulator accepts the same commands and reports the same kind of sensor information as the real robot does. For the main navigation software, the communication interfaces to the real robot and to the simulated robot are identical, as are the available commands. Usually, we run the simulator on a separate computer than the one the main navigation software runs on, to introduce similar delays in TCP/IP communication as on the real robot.

For further details about the simulator refer to (Beeler 2004).

3.4 Summary

Our system for building a cognitive map of an environment needs a mobile agent that moves in space and reports sensor perceptions from various places. We have designed and build a mid-sized mobile robot that accepts motion-commands from our system, autonomously performs basic reactive behaviors such as obstacle avoidance, and constantly reports on-board sensor perceptions. We chose a mobile robot of size 32x32 cm, which is sufficiently large to operate in office environments, but still is not dangerous for humans. The size of the robot allows adding special purpose hardware, such as a small on-board computer, a large variety of sensors and a huge battery. Software running on the on-board computer performs primitive behaviors and sensor-preprocessing autonomously, but also communicates over a wireless network link with the institute's computer network. Software on the robot constantly receives sensor information and uses these to maintain a representation of the local environment in the robot's frame of reference. Based on this representation the robot avoids obstacles and plans local detours to reach nearby targets.

At any time only a single place agent is interacting with the robot; typically the PA that represents space at the robot's current position. This active PA can update memorized information about its place by inspecting the robot's perception of the current environment. Or it can use differences between memorized and currently perceived environment to attract the robot closer to its own place (refer to chapter 4.3.2). In addition, the currently active PA provides near-by driving targets in robot-centered coordinates, as it interprets the local environment and determines the robot's actions in this environment. The robot, upon receiving such a target position, autonomously computes a path to this target, avoiding obstruction as seen in its representation of local space. This principle leaves the details of getting to a target for the robot; the place agent doesn't know how a particular robot executes locomotion, whether it drives, runs, jumps or flies.

We can replace the mobile robot with a simulator that provides an identical interface, such that the main system is unaware of it interacting with a simulator. This allows collecting large amounts of data in short time by running multiple instances of our system in parallel, and parameter tuning by replaying recorded trajectories in slow motion.

4 Place Agents

Our system decomposes global space into behaviorally relevant small patches that are each represented independently of all others. We refer to such a representation of a local place as a "Place Agent" (PA), because it is implemented as autonomously acting computer program (agent) with its private memory. Each of these PAs independently takes decisions based only on individually acquired knowledge about its local place.

Such PAs are the only elements in our system maintaining spatial knowledge. Ultimately, all behaviorally relevant places, e.g. those with a battery charger or an intersection of paths, get represented by an individual PA. Only the PA representing a particular place knows why this place is relevant, knows what the robot can do at this place and how a particular action can be performed at this place. Each PA represents its local environment in its local coordinate frame, including directions and distances to neighboring PAs. None of the PAs is aware of spatial structure beyond distances and directions to its nearest neighbors; neither does a PA know about a network of places. In fact no actor in our system is aware of a network of places, as will be discussed in chapter 5. As a PA is the only element in our system that maintains knowledge about a particular place, it additionally remembers possible actions that a mobile robot can perform at its place and instructions how to perform such actions. All such information is stored with respect to the PA's own coordinate frame; all information a PA provides to its neighbors are relative to its own coordinate frame. There is neither a global coordinate frame nor a global reference in our system.

Any one of such Place Agents can create copies of itself, which start as "unspecified blast" PA not representing a particular place. Once such a blast PA receives control of the mobile robot, it guides the robot to explore unknown space. Upon detecting a behaviorally significant position, this blast PA settles to represent the new place and transforms into a full PA. Repeating this process generates a collection of PAs representing behaviorally relevant places of large environments; each only knowing its local place and its direct neighbors represented in its own coordinate frame. The system started without prior environmental knowledge building up a distributed spatial representation from only a nucleus PA.

We implement Place Agents as active Java threads that run independently from each other, possibly on different computers. This chapter describes the above outlined process in detail.

4.1 Creating Place Agents

4.1.1 Creating a Nucleus PA

Our system starts without any knowledge about space. It needs an initial PA as active component, which triggers exploration of space and serves as nucleus to build up a network of independent PAs representing the environment.

Upon program start we artificially create such a nucleus PA (PA_{Nuc}) and give it control over the robot. Not representing a place and without any neighboring PAs, this nucleus commands the robot to rotate on the spot to obtain a rich signature of the current local environment. Inspecting that signature, PA_{Nuc} determines the center of nearby free space and guides the robot to that position.

Once in free space, PA_{Nuc} captures the robot's current spatial knowledge, memorizes this as its place signature, thereby turning into a regular PA representing this initial place (Figure 26). Note that a PA's reference frame is not aligned with the world reference frame as shown in the image. Instead a PA's zero-degree orientation (indicated by a large green spot in Figure 26) is aligned with the direction the robot happened to face when creating the PA. As we do not require a global reference frame, all PAs memorize their knowledge in their own arbitrarily chosen coordinate reference.

When capturing current environmental knowledge provided by the robot, a PA removes all data from relative sensors (refer to chapter 2.2.5). Current data from such sensors, e.g. a heading direction that got reset previously, is meaningless for a stationary place signature. Such sensors only provide relevant information while the robot is traveling between places, as we will show in chapter 4.3.2. In contrast, a PA captures all data the robot provides from absolute sensors (chapter 2.2.5), and merges such information with possibly existing previous information stored in its memory of its local environment.

The place represented by the initial PA does not necessarily have direct behavioral relevance, besides being the system's place-of-birth: it is wherever the robot happened to be after power-up. However, this nucleus PA starts the systems spatial exploration mechanism by creating additional relevant PAs (refer to chapter 4.1.2). In addition, PA_{Nuc} remembers that it is the robot's home, such that the robot can later return to the initial place.

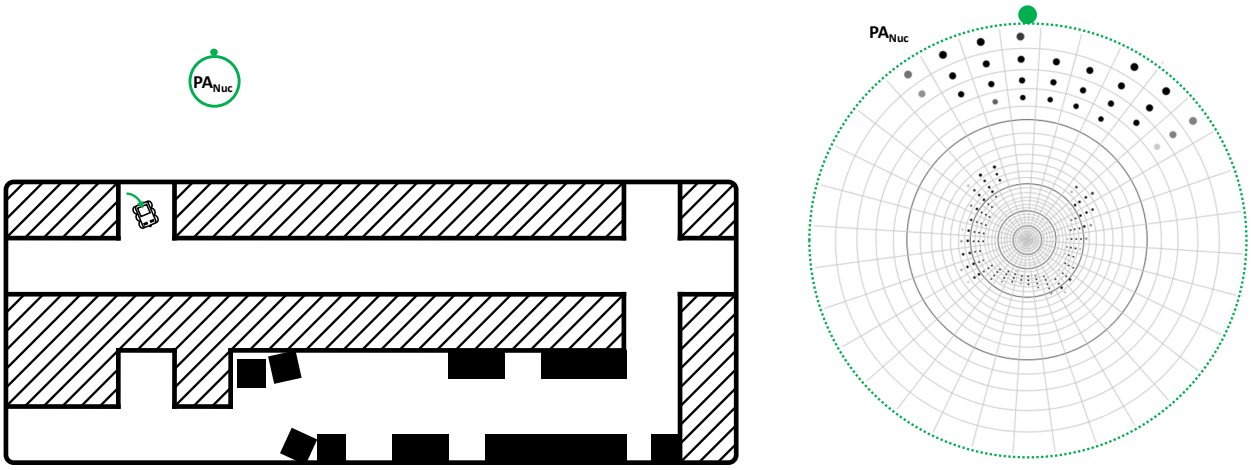


Figure 26: left: plan view of a "toy-world" environment of size 25x10m for a mobile robot. Note that there is no loop in this environment (handling of loops will be discussed later in chapter 5.2.4). An artificially created nucleus PA_{Nuc} (shown above by solid green circle) directed the robot from its starting position in a corner along the short green trajectory into free space. Once in empty space, PA_{Nuc} captures and memorizes a snapshot of the robot's current environment (shown in the dashed green circle to the right), thereby transforming into a full PA. Note that the orientation of the PA's memory - shown by green circle on outside - is aligned with the robot's current view of the environment, not with a global map orientation.

4.1.2 Creating Offspring PAs

Each existing PA represents a local environment, typically limited to an area within sensor view. Being an autonomously active program, such a PA can inspect its spatial knowledge and determine nearby unexplored regions. "Unexplored" here denotes traversable space for the robot that is not represented well by this PA, and that is not represented by at least one neighboring PA. To determine such open space, the PA searches from its center outbound in radial directions for

traversable straight paths (see Figure 27, gray lines). In addition to starting such paths from its center, the PA also tests starting positions shifted by ½ the robot's body length in all directions, such that it detects small detours around corners.

Figure 27 shows the above created PA_{Nuc} searching for free space in its own environment. The light gray lines extending radially from the PA's center denote free space, with superimposed blue lines pointing to local maxima. If those maxima exceed a distance threshold arbitrarily set to four times the robot's body length, the PA regards such directions as traversable paths for the robot. In the example shown in Figure 27, PA_{Nuc} detects a large region of unexplored space at about -15° orientations in its own reference frame. The other blue maximum at -145° does not exceed threshold; all other directions are blocked by walls. Note that there is no maximum at +125° due to limited resolution in the data representation and sensor noise.

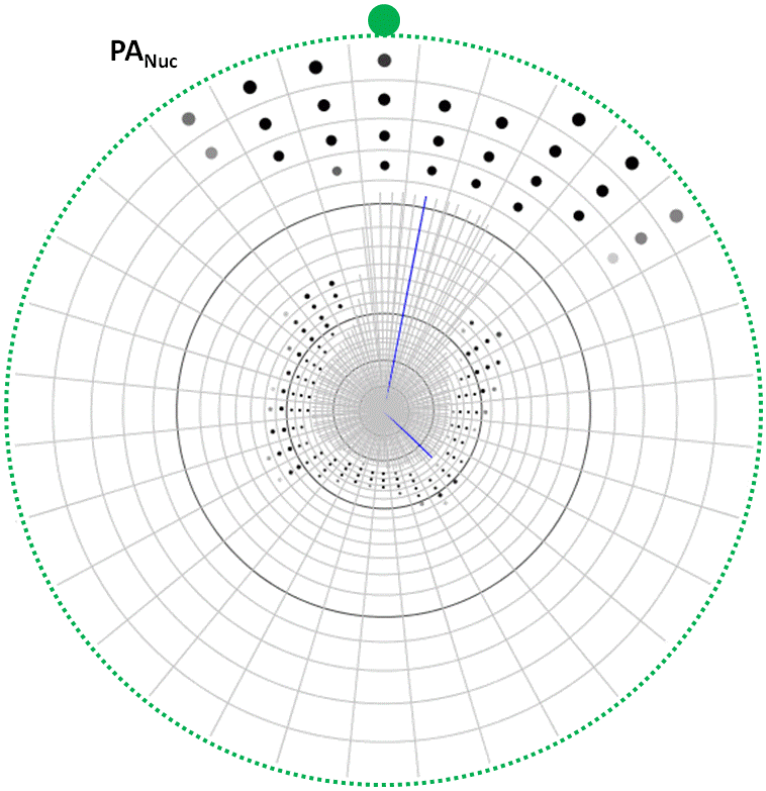


Figure 27: PA_{Nuc} inspecting its local environmental knowledge (shown in top view, black circles denoting obstructed space). Free radial paths starting from the center shown as light gray lines, local maxima superimposed in blue.

All such maxima show directions in which a robot can move, assuming it starts from the center of the current PA. This PA, however, can only provide coarse information about space in these directions as such space far away is in regions that are only poorly represented in the log-polar binned data structure. Ultimately the system has to represent such traversable space by an additional PA, so upon detecting such free space the current PA (in our example PA_{Nuc}) creates a child (here PA_1) as a duplicate of itself, running as an independent program (thread). We refer to such an offspring PA as "blast-PA" in reference to undifferentiated blast cells. The blast PA does not have any information about the environment; all it inherits is a reference to its parent (PA_{Nuc}). The parent keeps a reference to its child labeled with the angle towards its child in its own coordinate system. Figure 28 shows such a situation in which PA_{Nuc} has created a blast child PA_1 that does not

yet represent a place. In PA_{Nuc}'s knowledge about the world, the child PA1 is indicated as red dot at the correct angle but at an arbitrary distance of 4x the robot's body length.

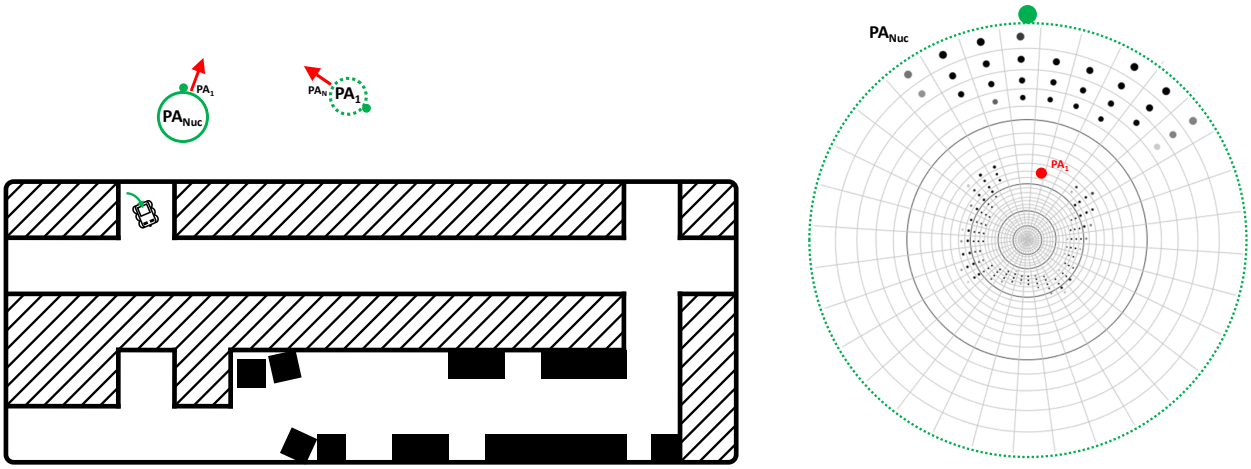


Figure 28: PA_{Nuc} detected open space and created an offspring blast PA, called PA_1. The new PA_1 does not yet represent an environment; it only knows the direction in which it shall explore: forward in its own coordinate frame, away from its parent which is exactly backwards in PA_1's coordinate system. In contrast, the parent PA_{Nuc} knows the angle to PA_1 in its own coordinate system, as shown by the red dot in PA_{Nuc}'s place signature (large green circle to right).

Such searches for directions to explore happen at a PA whenever its knowledge about its local environment updates. In the example above, PA_{Nuc} captured a place signature and created an offspring blast PA. This blast PA will - after exploration, refer to chapter 4.2 - capture its own place signature and possibly create further children, which then repeat the process. Later updates of a PA's local information happen when the robot returns to a previously visited PA and provides an updated "current place signature". In such a case the PA merges old and new spatial information and searches for additional directions to explore. If a PA's spatial environment changed, the PA will thus detect new open paths.

Note that a PA does not create blast neighbors in directions in which it already knows of a neighboring PA. In our example, PA_{Nuc} initially has no neighbors; so all traversable paths exceeding the distance threshold result in a blast PA. Later, when the robot revisits and updates PA_{Nuc}'s environmental information, the same open space would give rise to another blast PA; but as PA_{Nuc} already has a neighbor (PA_1) in this direction, it suppresses creating another blast. PA_{Nuc} assumes that PA_1 and its children represent this space. Furthermore, a PA creating a child internally remembers the direction of this child as already explored, such that if the child decides to represent a place in a different direction the parent still remembers that it already initiated exploration in this direction.

4.2 Exploring Space

After a PA created a child, this new blast PA exist as a standalone computer program - a new Java thread - just like its parent. The blast PA does not yet represent an environment and does not know what it will represent later. Initially, its only task is to explore unknown space as determined by its parent. Once a blast PA receives control over the robot, it guides the robot forwards until it detects a behaviorally relevant place (such as an intersection or a dead-end), or a behaviorally relevant object (such as a coffee machine or a battery charger). Upon such an incidence, the blast PA memorizes the

significance and captures a spatial signature of the current place, thereby transforming into a full PA that represents this place. Each new full PA again inspects its knowledge about its local environment to determine new directions for exploration. Upon detecting unexplored space, the PA creates its own children (new blast PA) which later continue exploring. Repeating this process, a network of PAs grows that finally represents all traversable space. Figure 32 at the end of this chapter shows an idealized sketch of a network that represents "toy world", with Figure 33 showing various resulting networks generated by a simulated robot in toy world. The rest of this chapter explains the exploration process in detail.

Note that initially we do not consider the special case of closing loops in an environment, i.e. surprisingly returning to a previously captured place along an outbound exploration. The current toy world example in this chapter does not contain loops; exploring more complex environments with loops will be discussed in chapter 5.2.4.

4.2.1 Finding New Places

A new-born blast PA (PA_B) wants to explore unknown space and determine which significant place to represent; but being a computer program it cannot do that alone. It needs access to the mobile robot for exploration, so it waits to receive control over the robot. Upon birth, PA_B's parent PA_P has control of the robot and decides which one of its neighboring unexplored regions - each represented by an individual PA_B - to explore first. In our initial example shown in Figure 28, PA_{Nuc} has determined only one unexplored region and thus created only one child PA_1. Typically, however, a PA_P creates multiple PA_Bs of which it picks the one at the position that is closest aligned with the robot's current orientation. Following this criterion minimizes the required changes of robot orientation before continuing exploration. Any other selection mechanism, including randomly picking a PA_B, also continues the exploration process. Carefully designing a criterion has impact on the order in which an environment is explored: e.g. always picking the left-most PA_B results in finding cycles in an environment quickly, whereas selecting the PA_B most opposite of this PA's parent results in fast forwards exploration of large environments, initially ignoring local loops. In our system ultimately all PA_Bs receive control of the robot and explore space. After deciding which PA_B to chose, the current PA commands the robot to rotate on the spot until it faces in the direction of PA_B (refer to chapter 4.3.1 for a detailed explanation), and hands over control to its child PA_B by sending a message (refer to chapter 5.1).

A blast PA_B starts exploring space upon receiving control of the robot. Initially, all such a PA_B knows about the world is a reference to its parent (its parent's name) and the direction to its parent: as parents create children such that they face towards unexplored space, the parent must be at 180° (backwards) direction in the child's coordinate system. Additionally, PA_B can assume free space in front which it shall explore.

PA_B's goal in exploration is finding a relevant place that the robot can reach directly following a simple forward trajectory starting from the current place - or alternatively, if no such place exists, the most distant place that the robot can reach following such a trajectory. During exploration, PA_B directs the robot to slowly drive forwards, while monitoring the robot's current spatial knowledge (refer to Figure 29). PA_B continuously updates the driving direction and speed - and ultimately decides to stop exploration - based on the following criteria:

- Open Space in Front of Robot: PA_B permanently checks free space in front of the robot up to 1m distance and ±45° angle to update the robot's driving direction. The desired direction is the average between the initial direction for exploration (as determined by its parent) and the center of current free space. Continuously adapting the driving direction detours around small obstacles while keeping the robot exploring space in the direction determined by PA_B's parent. See Figure 29, green triangle in front of robot and red driving vector. When free space in front of the robot is too small to continue, PA_B stops the current exploration.
- Avoiding Too Strong a Path Detour: the initial direction for exploration was provided by PA_B's parent; however, constraints on free space might ultimately lead the robot to detour. When the current robot position exceeds a predetermined threshold of ±20° with respect to the original direction of exploration, PA_B stops exploration. This keeps the robot trajectory directed forwards with respect to the initial direction of exploration.
- Avoiding Too Strong an Orientation Detour: if the current robot orientation exceeds ±45° of the original robot orientation, PA_B stops exploration. The robot might have to change its orientation abruptly due to an obstacle, but still be well within the ±20° corridor monitored above; especially once the robot is further away from its starting place. PA_B would like to avoid such sudden change of orientation which would complicate the trajectory between its parent and itself. This criterion denies such kinks in the trajectory.
- Detecting Spatial Geometric Pattern: As most human-designed environments are structured the robot detects particular geometric arrangements in its environment that are relevant for navigation. Examples of such arrangements are intersections of aisles, an aisle opening to the side, or a dead-end of a corridor. During exploration PA_B constantly runs the algorithm presented in chapter 4.1.2 and Figure 27 to determine current directions of open space. When these directions resemble a pre-coded pattern up to an error margin, PA_B directs the robot towards the center of the arrangement and stops exploration.
- Detecting Behaviorally Relevant Objects: In a future version of the system we expect PA_B to receive feedback from other components of the software (similar in function to higher brain regions) that detect behaviorally relevant objects in the current environment. Upon such information, PA_B guides the robot close to that object and stops exploration. Currently, the simulator only provides a small number of hardcoded objects; the real robot does not recognize behaviorally objects itself but only by human intervention. Detecting such objects in real-world environments remains a challenging research question, mainly in the area of computer vision.

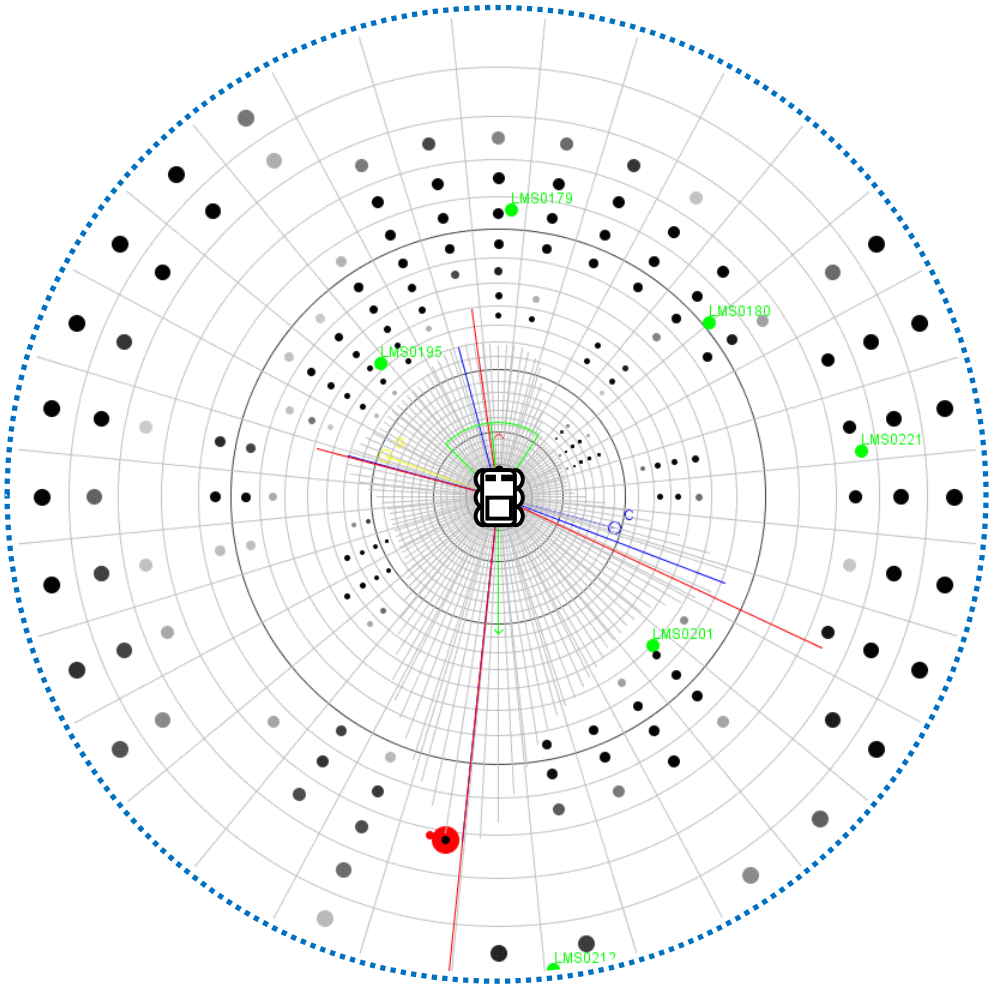


Figure 29: Exploring space: A top-down view of the robot while exploring space. The green triangle shows immediate free driving space, the small red vector the current direction for exploration. Gray lines show free space as seen by the robot, red lines possible aisles for exploration. The green vector pointing backwards shows the PA's estimate of its parent's position.

4.2.2 A Name for a New Place

During this documentation, names for new place agents follow simple natural numbers for clarity: PA_1, PA_2, PA_3, ... Implementing such a naming scheme in the running system, however, requires that all place agents have access to a global counter, which violates our local-knowledge-only paradigm. Looking into our system, however, we can see that the PAs do not need to have global names; it is only for our convenience when analyzing the system that we refer to a PA with a global name. Running the system, each PA only has to distinguish its local neighbors; a task for which it can apply its own a local naming scheme. Multiple neighbors of a single PA can internally reference to that single PA using different names, as long as each PA for itself keeps a unique name for that neighbor. So each PA can use its own local counter to label children and neighbors.

However, for our convenience when inspecting the system, we implemented a mechanism that allows finding globally unique names: A parent keeps its name for all its children, but adds an individual identifier to each, e.g. A, B, C, ..., resulting in unique place agent names. Unfortunately such names quickly grow inconveniently long. Shortening the names, when a child transforms from blast to a full PA, it changes its name to the current system time. Only one PA in the system can

perform such a transition from blast to full at any instant of time - so the PA's name must be unique. Furthermore, this PA needs to have control of the robot to perform the transition, so the apparent global system time can be provided by a "counter-sensor" on the robot to just this one PA, making it a currently only locally accessible measure. No other PA knows about the time while a blast PA transforms into a full PA.

4.2.3 Establishing a New Place's Knowledge

After PA_B stopped its exploration, it is roughly at the place that it will later represent[3]. It directs the robot to rotate on the spot to increase the richness of the robot's sensory signature representing the current local environment. Afterwards, PA_B behaves identical to a nucleus PA explained in chapter 4.1.1: it centers the robot in nearby free space, captures the robot's current spatial knowledge and memorizes this signature as its own place signature, thereby transforming into a regular PA that represents this place. As for the nucleus PA, the coordinate frame of this new PA is not aligned with any other permanent reference in the system - and not aligned to a global origin - but only with the orientation the robot happened to have after exploration. Again, as we do not have a global reference frame, all PAs memorize their local knowledge of the world in their own arbitrarily chosen coordinate reference.

The captured local signature constitutes the PA's basic knowledge about the world. In addition, the PA might have received information about behavioral significance at this place from other components of the system, e.g. from a vision program that detected coffee. This information might be provided as key word "coffee", as a picture of the coffee machine, or any other abstract data. The new PA does not understand the content of the data, and simply memorizes the provided data as a feature of its place, exactly as provided by someone else. This stored data will be useful for goal-directed navigation, as explained later in chapter 5.3.1. Note that the PA does not inspect or interpret the data; it will only memorize the information.

Another important piece of information that a new PA has about the world is distance and direction to its parent in its own coordinate frame. During exploration from its parent to the current place, the new PA directed the robot, and therefore had permanent access to driving information. This allowed the PA to integrate the path the robot took, resulting in a vector towards its parent. The new PA enters the position information about its parent in its place signature. In addition, the new PA sends a message to its parent (refer to chapter 5.1.1), notifying it about the transition from blast to regular PA and about the distance and angle it directed the robot. When receiving such a message, the parent PA modifies its own knowledge, adjusting PA_B's position in its own coordinate frame. Note that this information exchange happens completely based on local information: The new PA reports distance and angle information relative to the start of the exploration; it does not know its parent's reference frame or a global coordinate frame. The parent does not treat the received information as distance and angle in its coordinate frame or in a global coordinate frame; instead, the parent sees the information relative to the orientation it has originally created its child in. For the parent nothing else changed, only the position of one of its neighboring PAs; it does not remember if a neighbor is a blast or a full PA.

[3] In those rare occasions when a blast PA ends exploration close to its parent (e.g. because of a previously unseen obstacle), the blast PA decides to return the robot to its parent and decease. The parent PA remembers that it has previously sent a child for exploration and does not attempt to not re-explore this direction.

Note that solely the fact that a PA_B found a trajectory from its parent to its current position established the neighborhood relation between these two PA, such that they know of each other and can exchange messages. This implements the main principle of connectivity in our network of place agents: Every place agent knows only about those other PAs to which the robot can travel directly from this PA; but it does not know about other PAs in the system. Constraining the knowledge of all PAs as such fulfils our commitment to maintain local knowledge only. We do not have an actor in the system that has access to information beyond its local environment: here the outreach of local environment is beyond the current sensory signature, but it is still limited by each PA's nearest neighboring PAs.

4.2.4 Continuing Exploration

The new PA has turned itself into a full place agent that represents a particular place in space and knows about its parent and the spatial arrangement between itself and its parent in its own coordinate frame. Just as its parent did the new PA inspects its spatial knowledge of the local environment to search for unexplored space and creates children in appropriate directions (refer to Figure 30). Note the modified lengths of arrows between PA_{Nuc} and PA_1 compared to Figure 28: Initially, the distance between PA_{Nuc} and PA_1 was unknown, so PA_{Nuc} placed its child PA_1 at some predefined distance; initially only the direction was important. After exploration, PA_1 settled to represent a particular place and determined the distance to if parent PA_{Nuc}. Figure 30 shows PA_{Nuc} and PA_1 (above the top-down view of the environment) after they updated their knowledge about their respective neighbor. The two new blast PAs shown next to PA_1, called PA_2 and PA_3, are children from the new place agent PA_1 created to explore the two open directions PA_1 sees in its environment.

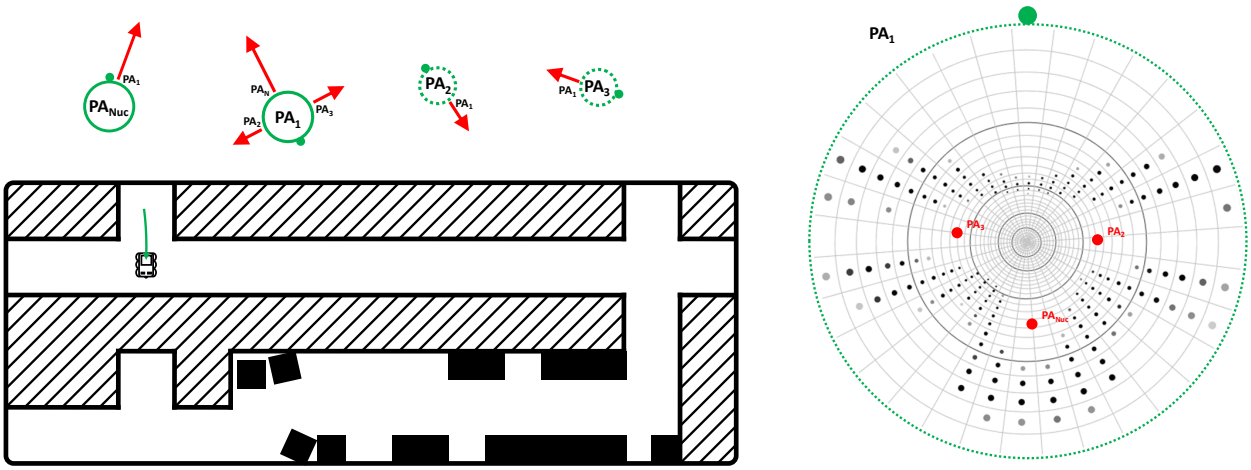


Figure 30: The environmental representation after PA_1 turned from blast into a full place agent. Note the new length of the arrows connecting PA_{Nuc} and PA_1. Inspecting its signature of the local environment (shown as large green circle on the right), PA_1 decided to create two children (new blast PA) to explore along both open directions of the aisle.

Note that in Figure 30 the place agents seem to float somewhere in space. This notation closely resembles what really happens in the system: none of the PAs is aware of its true global position in space. They all know distances and directions to their specific neighbors only; no actor in the system is aware of a particular arrangement of the collection of place agents. Figure 31, in contrast, shows the current situation with four PAs that have been hand-arranged as overlay over the existing map of toy-world to reveal the underlying spatial structure. Such a display is helpful for human observers

to understand how the PAs represent space; however, the system does not require such knowledge for navigation, and no actor in the system maintains such a representation. Later, with cycles in the environment, it might not be possible to arrange all PAs and their individual knowledge correctly due to limited sensor accuracy during exploration. For our reading convenience, most displays of place agents in this thesis are arranged to resemble the underlying spatial structure as closely as possible (refer to chapter 4.4 for a detailed discussion about displaying PAs).

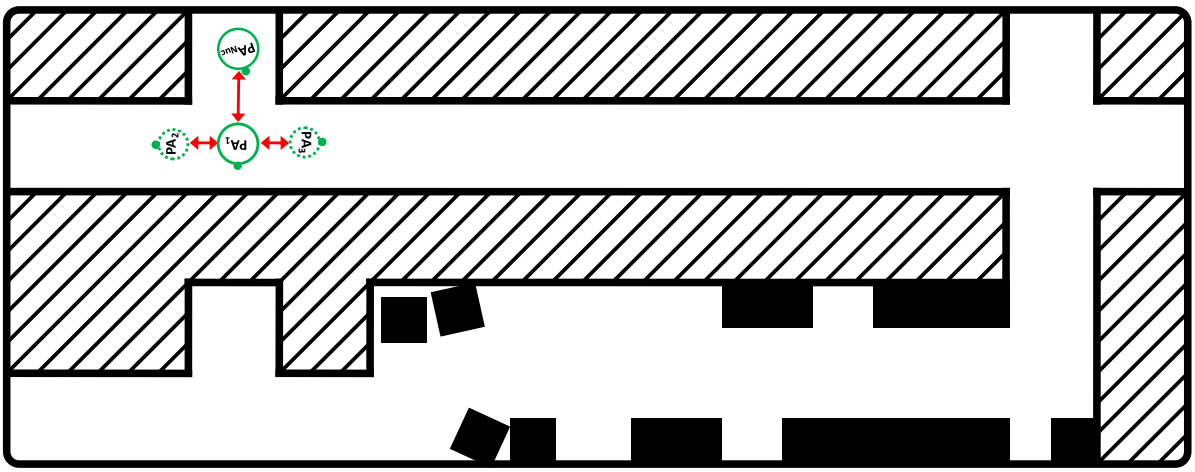


Figure 31: The four independent PAs spatially arranged to reveal the underlying spatial structure. Note that this representation is only for display such that humans can recognize the spatial structure. The system is not aware of this global arrangement: all agents "float in space" as shown on the top of Figure 30. Refer to chapter 4.4 for a more detailed discussion about the network display.

After creating children, a new PA hands over robot control to one of its children to continue exploration of unknown space. Alternatively, without children, the new PA returns control to its parent, which initially directs the robot back to its place (refer to chapter 4.3.2). After the robot returned to this PA, it hands over control to another child or its respective parent. Iterating this process until no PA has blast PA children finally results in a distributed representation of the toy-world environment as shown in Figure 32. Note that this representation is an idealized sketch for clarity; initial true exploration results of the simulated toy-world environment are shown in Figure 33. Although the current toy-world does not contain loops, sometimes multiple PAs each detected a path into a single dead end, such that a group of three PAs created a small loop, as will be discussed in chapter 5.2.4. As can be seen, these three results differ in complexity and spatial arrangement of nodes. Most obviously, the bottom network shows a significant skew between the two parallel corridors. Although the three results seem to be of different quality regarding the spatial representation, the basic spatial layout of toy-world (a straight long aisle on the top, a more complex aisle at the bottom, various intersections) can easily be recognized in all three networks. For a detailed discussion about the quality of such graphs refer to chapter 6.1.4. Remember that the spatial arrangement of PAs does not exist as such in the system; the display in Figure 33 shows an arrangement of all PAs such that the error with respect to all PA's individual knowledge about neighborhood relations is minimized (refer to chapter 4.4 for details about the display). Such a display is only useful for human observers when interpreting the network - but then again it is very helpful for humans.

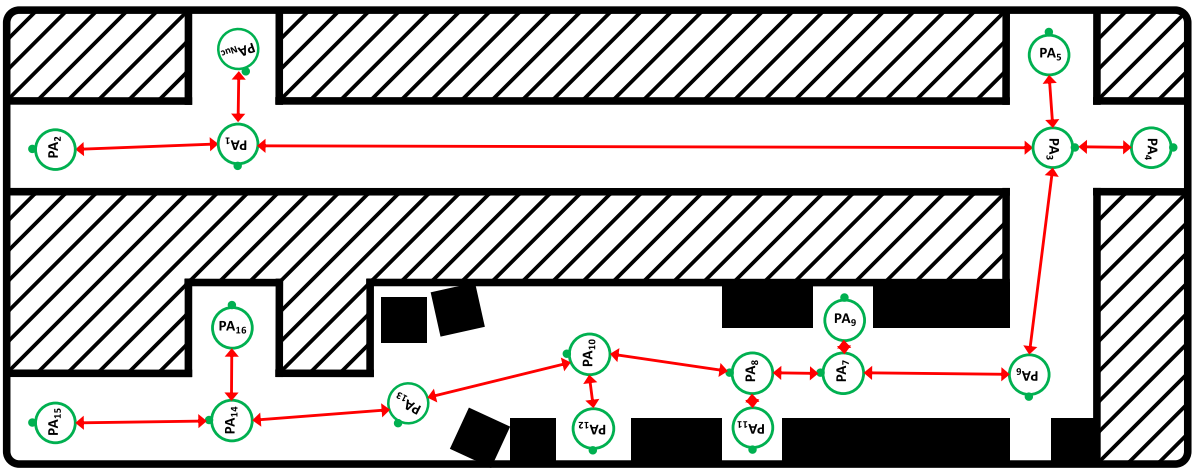


Figure 32: A final network after exploring "toy world" - an idealized example for clarity. Remember that the spatial arrangement of PAs does not exist as such in the system; it is only overlaid on the true map for our convenience. PA_6 at the right bottom might have created a child to explore free space below it; but the child - upon closer inspection of the free space - decided that it is too small to fit in and deceased.

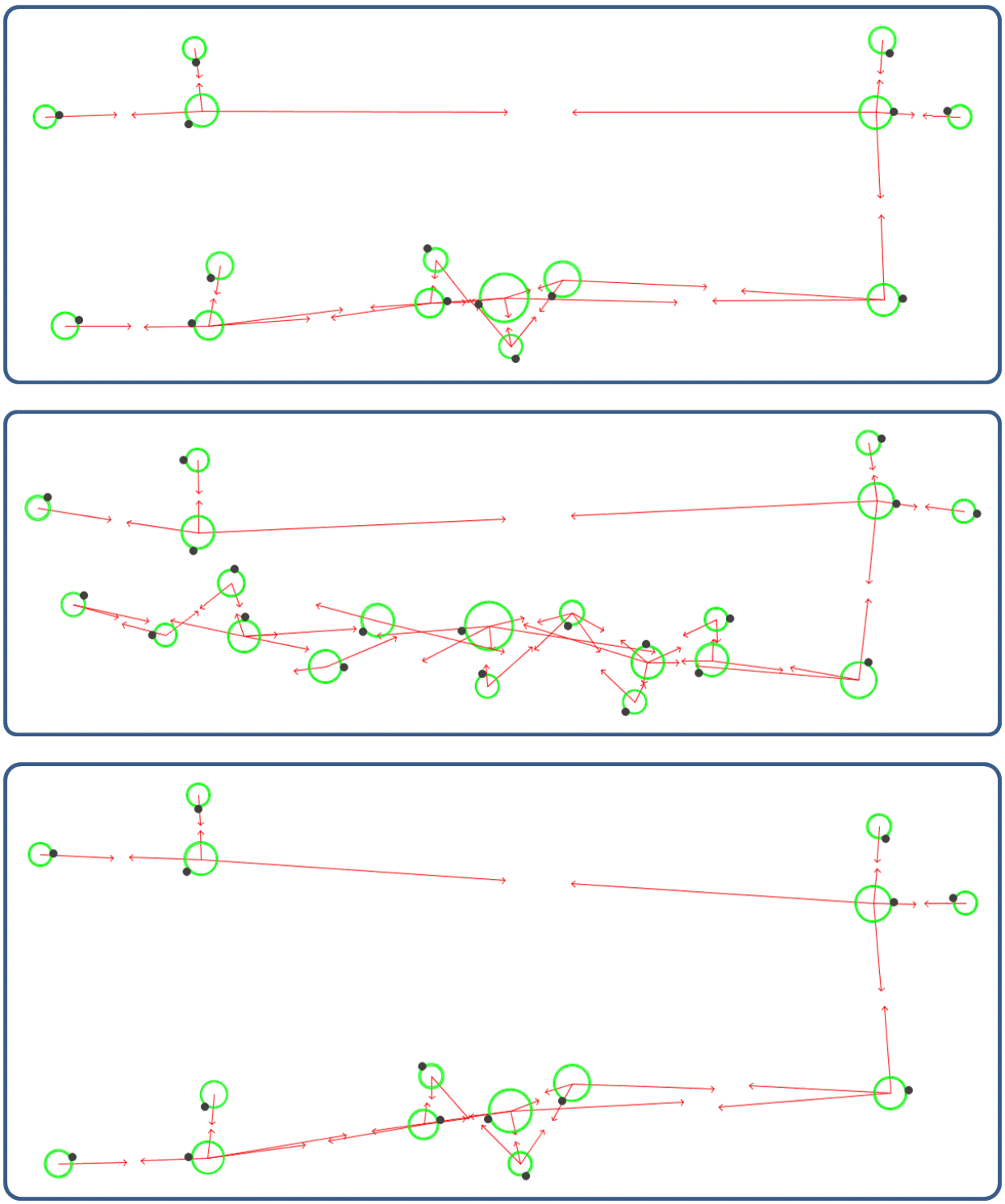


Figure 33: Resulting networks of place agents from three independent exploration runs in “toy world”, captured by the PA network GUI (refer to chapter 4.4). Green circles denote place agents, red arrows show neighborhood relations (length of an arrow corresponds to 45% of the total distance to their respective neighbor).

4.3 A Single Active PA Controlling the Robot

At any point in time only a single PA communicates with the robot and directs it to perform particular actions. We call this PA the currently active PA, PA_C. In this chapter we will outline actions that PA_C can perform while controlling the robot. In general, elementary interaction between PA_C and the robot is restricted to two primitive operations:

- Fetching a current environmental signature from the robot as seen in the robot's current coordinate frame (the robot's current view of the local environment)
- Sending a desired driving target to the robot, which consists of position and orientation in the robot's current coordinate frame

Iterating these two basic actions and computing appropriate driving targets based on the current view, a place agent can perform various complex actions.

4.3.1 Guiding the Robot at a PA

Assuming that the robot is close to the center of the place PA_C represents, PA_C can direct the robot to a position and orientation with respect PA_C's own coordinate frame. Such an action is initially required when PA_C sends the robot to one of its neighbors - it needs to orient the robot facing the neighbor; but also when the robot shall approach a particular object that PA_C memorizes at its place.

While guiding the robot, PA_C continuously fetches the robot's current environmental signature (the currently perceived environment) and compares that information against its own memorized place signature (ref to chapter 2.6). The result of such a comparison is a vector showing an offset in position and orientation between the robot's current place and the center of the place represented by PA_C. Assuming PA_C wants to guide the robot towards its center, applying simple PD-control with low gains computes desired driving targets for the robot, such that the offset in position - and possibly orientation - decreases. Repeating this process slowly gets the robot closer towards the center of PA_C's place signature. When below a distance threshold, PA_C only sends further rotation commands, ultimately making the robot face towards zero degree at the origin of PA_C's coordinate frame.

Assuming PA_C instead wants to send the robot to one of its neighbors PA_N, the robot does not need to be aligned to zero degree orientation, but instead towards PA_N's position in PA_C's place signature. Adding an angular offset (PA_N's angle) to the desired target angle for the PD-control modifies the robot's target towards PA_N. This is a simple solution, but conceptually not very elegant. Instead, we stay within the existing framework of place signatures and develop a method allowing rotating the robot to an arbitrary object memorized in the existing signature: Initially, PA_C creates a copy of its place signature, and rotates this copy such that the target object faces forwards in the new signature's coordinate frame (see chapter 2.5). Afterwards, PA_C follows the same principle outlined above based on the copied and rotated place signature. This method allows orienting the robot towards arbitrary objects stored in the place signature, including neighbors, but also those objects that the PA does not understand because they were provided as abstract behaviorally significant targets by someone else, e.g. a battery charger. Furthermore, not only allows it rotating towards objects but also driving towards objects represented in the stored environmental signature: Assuming PA_C wants to guide the robot to a target object that is represented in its signature, it copies, rotates and translates the signature such that the object is in the center of the new "target

signature". Following the same algorithm using simple PD control techniques on the difference between current and target signature, PA_C guides the robot towards the local target.

In such a scenario the robot might have to start from places further away to the center of a target signature - even when the robot currently is in the exact center of the original place signature. A potential target might be several robot body lengths away from the original center of the place signature, such that the distance the robot needs to travel grows. Each target, however, will still be well within the outreach of this place agent, as we are currently considering local targets only. Comparing the robot's current view, which continuously changes while in motion, against the computed target signature often provides a fluctuating estimate of the robot's position and orientation. Such poor estimates can trap the robot on its way in local minima, e.g. by infinitely turning left and right. To reduce the likelihood of such a problem occurring, PA_C maintains a low-pass filtered estimate of the robot's position: PA_C initially finds the best estimate of the robot's position relative to the target place signature, and stores this estimate in a position vector P. Future robot motion commands that PA_C sends to the robot get integrated and update P, such that in an ideal world no further position estimates are required and the robot reached its target. However, due to uncertainty in the initial position, actuator nonlinearities and sensor noise, further comparisons between current view and target signature are required. Each such future comparison results in an error vector, which only updates P by a small fraction of its value. We chose an update rate of 5% which turned out to be sufficiently high to correct for initial errors over time, but still saves PA_C from large jumps in its position estimate regarding the robot's current position.

These simple techniques allow a single PA controlling the robot in its vicinity: guiding the robot to the PA's center, guiding it to memorized objects in a PA's environment, and orienting it towards one of its neighbors.

4.3.2 Guiding the Robot from a PA to one of its Direct Neighbors

The last chapter discussed how the currently active PA_C directs the robot to a target that is represented in its spatial signature. Alternatively, PA_C might decide to send the robot to one of its neighbors as a next step towards a long range target (refer to chapter 5.3 for a discussion about long range goal-directed behaviors). This is a task much more difficult, because it involves interactions between two neighboring PAs, two independent place signatures and it might require moving the robot through possibly unknown space in-between the two PAs. Still, guiding the robot from a starting place agent PA_S to a target place agent PA_T is a straight forward process that slightly extends already introduced principles. Note that a currently active PA can only send the robot to any one of its neighbors, since PAs are not aware of the existence of other PAs beyond their direct neighbors. Additionally remember that - by the way the place agents constructed the neighborhood relations (chapter 4.2) - a simple trajectory exists between the currently active PA and all of its neighbors.

Starting a transition from PA_S to PA_T, the PA that currently has control over the robot ($PA_C = PA_S$) uses the mechanism described in the previous chapter to orient the robot towards its neighbor PA_T. When done, ideally the robot is exactly at the center of PA_S, facing exactly towards its neighbor PA_T; however, in real-world situations the robot will have a position offset ΔP and an orientation offset ΔO. If these offsets are expressed in PA_S' native coordinate system only PA_S can interpret them. However, PA_S can express both these values in the rotated coordinate system that it used to orient the robot towards its neighbor PA_T. Hence, the position offset ΔP and the orientation offset ΔO

describe offsets relative to the desired starting position and orientation when traveling from PA_S to PA_T. Refer to Figure 34 for a sketch of such a situation showing the offsets in blue.

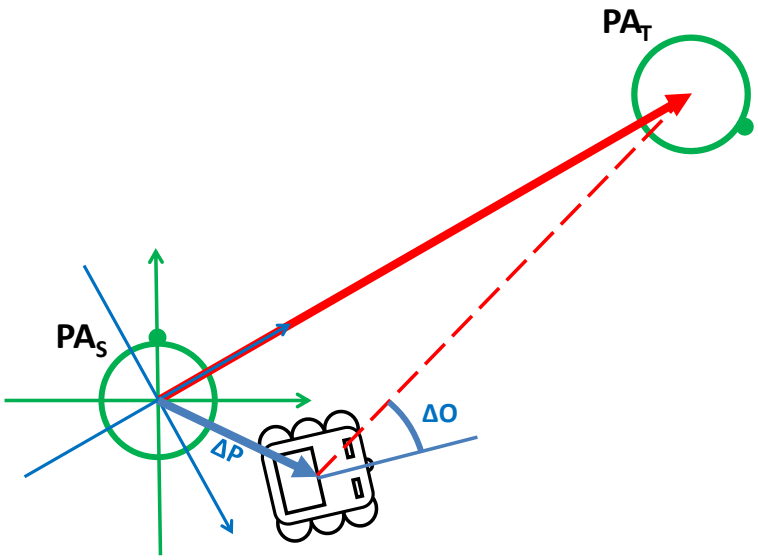


Figure 34: PA_S oriented the robot towards its neighbor PA_T, leaving an error in position and orientation. Expressing such offsets in PA_S' coordinate system (green) does not provide information for PA_T (PA_T is unaware of PA_S' coordinate system). PA_S expresses offsets in position (ΔP) and orientation (ΔO) in a temporary coordinate system (blue) that is aligned with the desired direction towards its neighbor PA_T.

Once the robot faces the target, PA_S releases control and signals its neighbor PA_T to take over robot control (refer to chapter 5.1.1 regarding message exchange between neighboring PAs). In addition to this take-over signal, PA_S submits the robot's position and orientation offset with respect to the ideal starting position. Upon receiving such a message, PA_T knows that it can attract the robot from the one of its neighbors that sent the message (PA_S).

For PA_T, attracting the robot to its own center is similar to sending the robot to an arbitrary target in its local environment; here the desired target is at the origin of PA_T. PA_T therefore needs to prepare a copy of its own place signature that suits the task: coming from its neighbor PA_S, upon arrival the robot will face in the direction opposite of PA_S. Therefore, PA_T rotates a copy of its place signature such that PA_S is at the back (180° orientation) of the new place signature's coordinate system. In this frame of reference, PA_T can also compute a vector to the robot's current position, given its locally available information about the distance to its neighbor PA_S and the received robot offsets in position (ΔP) and orientation (ΔO) from the ideal starting position, as shown in Figure 35. Such a temporary coordinate systems allows the place agents to communicate the robot's true position, without knowing each other's coordinate frame. PA_S and PA_T have shared knowledge about the distance between them, but no other shared information. Each of the PAs operates in its own coordinate frame, based on its own information about its neighbor. PA_T uses the computed position vector to initiate its low-pass filtered estimate of the robot's current position, which is required when guiding the robot to a local target as explained in the previous chapter.

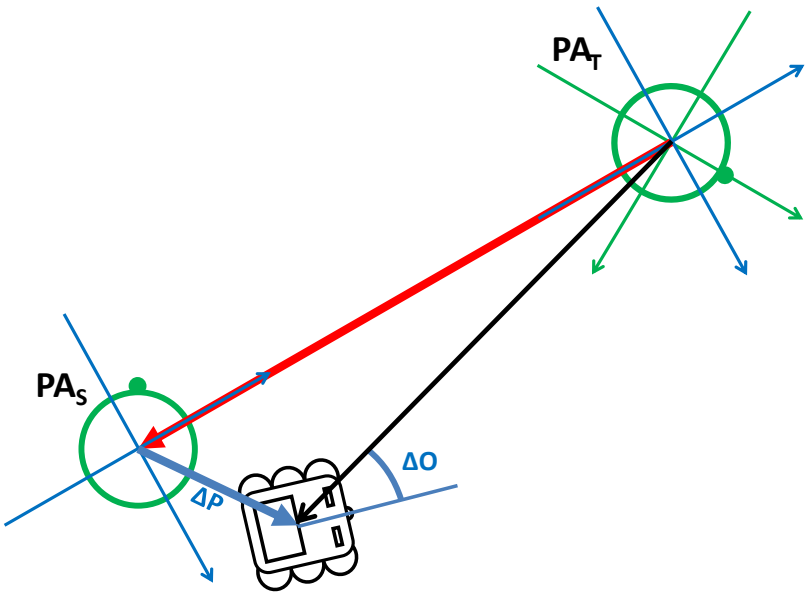


Figure 35: PA_T received a message, signaling to take over robot control from its neighbor PA_S. PA_T rotates a copy of its own place signature such that its neighbor PA_S is at the back (blue coordinate system inside PA_T). Computing the robot's starting position, PA_T adds the provided offset in position (ΔP) and orientation (ΔO) to the place it assumes its neighbor PA_S to be, using its temporary coordinate system (blue). This temporary coordinate system is identical to the coordinate system used by PA_S for computing position and orientation offset. This results in an estimate of the robot's current position and orientation in PA_T's temporary coordinate system, as shown by the blue arrow.

When attracting the robot from a known starting position, PA_T can use additional sensory information provided by the robot, besides those already stored in its signature about its local environment. The robot maintains a variety of sensor signals that report changes relative to an occasionally reset baseline (refer to chapter 2.2.5), e.g. a gyroscope reporting changes in heading direction or a virtual target sensor monitoring ego motion based on integrated wheel rotations. Data from such sensors is not captured in PA_T's quasi stationary place signature, since such sensors report different values every time the robot reaches PA_T, subject to their last reset. However, having an estimate of the robot's current position, PA_T can reset such sensors on the robot to its own advantage: after reset, PA_T knows their current baseline and thus can predict the data they will report at the time the robot arrives. Assuming a perfect starting position, PA_T resets the robot's on-board gyroscope and predicts a heading direction of zero degree when the robot arrives from its neighbor. For a known offset in orientation ΔO, PA_T can initialize the gyroscope with a value of ΔO instead of zero, such that the on-board gyroscope takes the robot's current offset into account. Similarly, PA_T will initialize the robot's virtual target sensor that reflects ego motion based on wheel integration: for a perfect starting position, PA_T sets the robot's virtual target at zero degree in a distance equal its distance to its neighbor PA_S. As PA_T knows about the robot's position and orientation offset from the perfect starting position, it can adapt the robot's virtual target appropriately. In an environment without wheel slip or sensor errors, the robot's on-board motion integrator will shift the position of the virtual target to zero when the robot arrives at PA_T. After setting up these additional sensors on the robot, PA_T adds the expected target values of those relative sensors to its target signature (the rotated copy of its local spatial signature). Having such sensor representations in both signatures - the robot's current signature and the target signature - provides additional information about the robot's current position when comparing both signatures while the robot approaches PA_T.

For a set of two place agents PA_S and PA_T that represent nearby places - within a few times the robot's body length - the problem of guiding the robot from PA_S to PA_T is solved: knowing that the robot's current position and knowing it is well within PA_T's local environment, PA_T can guide the robot to its own center, using the mechanisms described in the previous chapter. In most environments, however, various neighboring PAs are significantly further apart than a few times the robot's body length; for an extreme example see PA_1 and PA_3 in Figure 32, representing both ends of the long aisle in the top of the environment. In such a situation, when PA_T compares the robot's current view against its target place signature, various equally likely matches will occur at divergent positions because of missing overlap of the two signatures. PA_T will fail attracting the robot to its own center, because the path between the two PAs is too long and available information is too sparse. PA_T uses the following two tricks to compensate for the poor knowledge about the track between PA_S and itself:

PA_T continuously maintains its estimate of the robot's position in its own coordinate frame based on the robot's current motion. PA_T uses this estimate to projects its own target signature such that it matches the expected current view of the robot. As an example, image what a robot senses that is 1m away from its target: It will perceive an object that is exactly at the target to be in 1m distance. The target signature represents the same object exactly in its center; hence the two place signatures are shifted by 1m with respect to each other. PA_T needs to shift its target signature by 1m to align those two signatures, in the direction away from the approaching robot. Such a shift moves the object at the center in the target signature to 1m distance, just as the robot currently perceives it.

While guiding the robot, PA_T constantly computes an updated target signature by shifting its initial signature. PA_T uses its maintained low-pass filtered estimate of the robot's position vector for such a shift. Comparing the robot's current signature against a shifted target signature can happen within a much smaller search range, typically limited to a circle of less than 1m diameter. This significantly reduces potential false matches compared to a search radius the size of the robot's current distance. Note that for initially large distances, very little or no overlap will occur between data in the robot's current view and the shifted target signature. However, no overlapping data reports an "unknown position" in the comparison, so the result does not corrupt PA_T's position estimate, but false matches would. PA_T's position estimate is still updated by robot motion commands that PA_T sends to the robot. Figure 36 shows a sketch of the target PA_T attracting a robot.

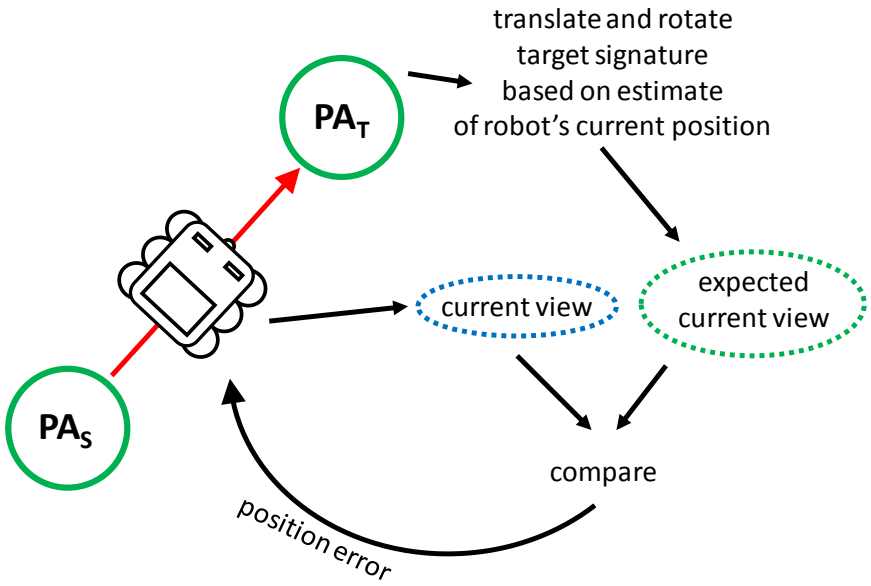


Figure 36: Sketch of target PA_T attracting the robot from its neighbor PA_S. PA_T translates and rotates its internally maintained target signature such that this signature (green dashed circle) matches the robot's current view (blue dashed circle) as close as possible. It compares the two signatures and computes short-range driving targets based on the difference of the two signatures.

The second trick PA_T uses to help the robot crossing large regions with possibly limited overlap in sensor information between current view and target signature is adjusting the significance of different sensor signals according to their reliability along the way. As an example, a gyroscope is highly reliable initially, but decays in performance over time and distance traveled. An absolute sensor in contrast, such as a compass, reports equally reliable all along the way; however, local disturbances of the magnetic field might have a small impact along the way. Sensors reporting stimuli that are only visible at the target, such as local landmarks, initially perform very poor but report highly useful data close to the target. Figure 37 shows graphs of the quality of contribution for selected sensors depending on distance to the target. Note that clearly two types of sensors exist: Those that show increasing performance closer to the target and those showing decreasing performance. Refer to the figure caption for a detailed explanation about the individual graphs' shape. PA_T continuously estimates the robot's current position and orientation for each sensor independently, but weights these estimates with the sensor's current significance, based on PA_T's own estimate of the robot's current position. Summing all sensor predictions and dividing the sum by the total sensor contribution for normalization provides a single estimate of the robot's current position that is weighted according to sensor significance. Using this trick, PA_T initially guides the robot by using dead-reckoning sensors only, such as the gyroscope, the virtual target, or the compass. All these sensors are well suited for an initial approach to PA_T through places that are not represented well in PA_T's place signature. However, as the quality of other sensor signals gets better, the dead-reckoning-sensors reduce their contribution, which ultimately leads to a precise estimate of the robot's position close to the target.

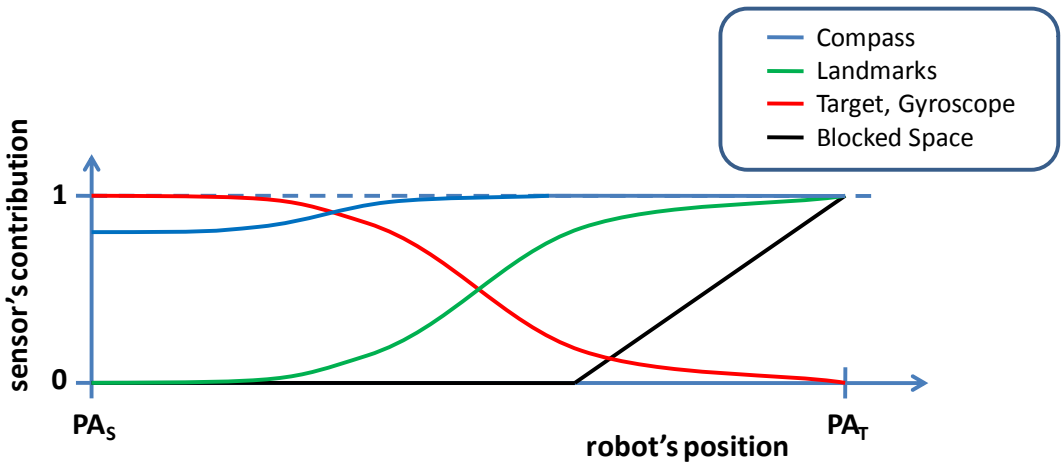


Figure 37: Sensor contribution with respect to target distance. Blue: a compass sensor that reports reasonably precise data everywhere, but highly precise data close to the target. Green: Landmark sensor that does not recognize landmarks in large distance - or only with large uncertainties about position and identity. As the robot approaches PA_T, its position estimates get precise, such that its contribution increases. Red: Relative sensors, such as a virtual target or a gyroscope, reduce their contribution over time, ultimately decaying to zero close to the target. Black: blocked space sensors are likely to report false matches while far away from PA_T; therefore their contribution is neglected until a fixed distance before the target, at which point it increases quickly.

Applying such techniques allows PA_T to guide the robot from its neighbor PA_S to its own center, independent if PA_S is nearby or far away. We implemented a simple random walk for the robot to show transitions between place agents: every PA sends the robot to an arbitrary one of its neighbors after centering the robot at its own place. This simple principle results in an infinite trajectory for the robot in explored space. Figure 38 shows the trajectory of such a random walk in "toy world", as generated by a network of independent place agents, which are all ancestors automatically created from a single nucleus PA.

Inspecting Figure 38, one can immediately see that the green trajectory the robot took does not follow the exact same positions on multiple repetitions of a path. This is due to two different effects: On one hand, sensor and actuator noise in the simulated robot causes initially identical trajectories to diverge quickly. On the other hand, all PAs in the system update and refine their estimates of their neighbors' positions when the robot travels between them (explained in detail in the following chapter). This results in different starting directions when traveling between the same two PAs over time. However, the green trajectory clearly shows a large number of "hot spots"; i.e. places visited by the robot over and over again. Those are the places represented by place agents, which remember the local spatial context and manage to guide the robot close to their own center. Following a single trajectory between to PAs it is often clearly visible that the path initially diverges from an optimal path, but later settles down in the center of the target PA. Typically, such strongly diverging trajectories occur only in an early stage after exploration, when the individual PAs have not yet consolidated their knowledge about their neighbors. A more detailed analysis of such networks of place agents and resulting trajectories will be provided in chapter 6.2.2; this figure only provides a first impression of what kind of trajectories a network of place agents generates.

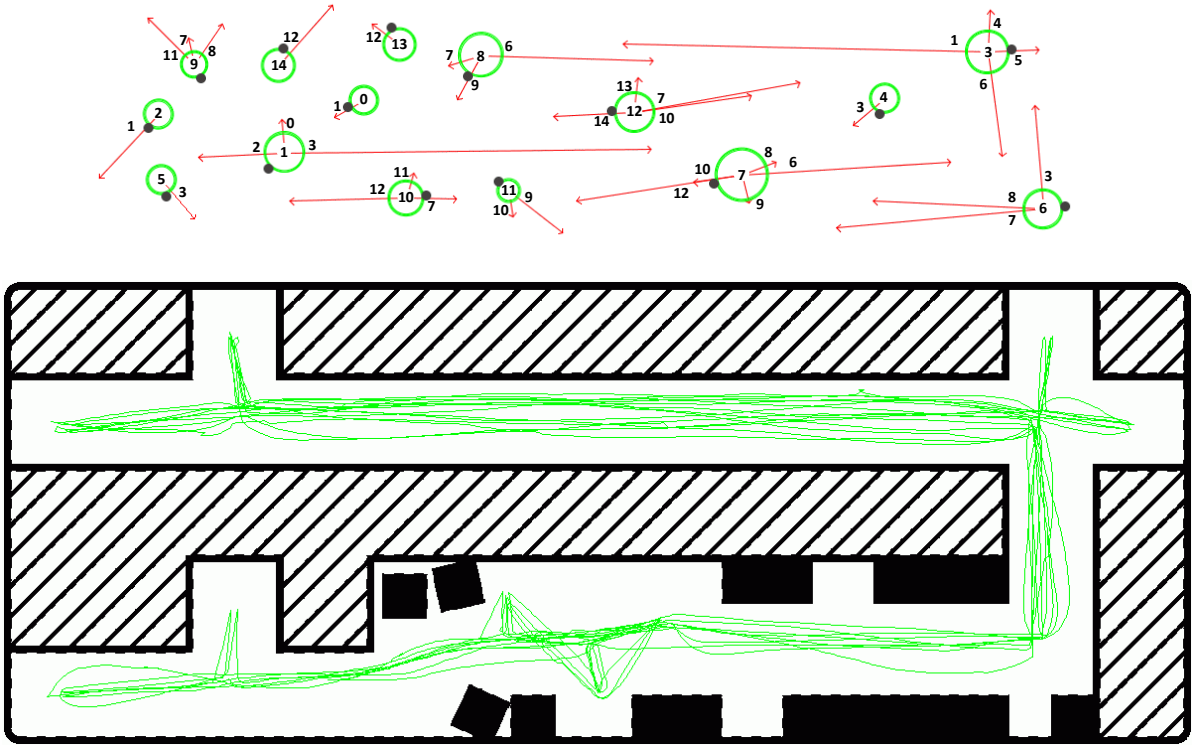


Figure 38: Resulting network of place agents and a trajectory of random walk in the 25x10m "toy-world", guided by the collection of place agents shown above (details in text). The network of place agents is shown as it is maintained in the system: a collection of independent place agents that each has no global reference, but only knows its neighbors' names and positions in its own coordinate frame. Readers finding it difficult to interpret such a collection are invited to refer to the top panel in Figure 33, which displays the same network of place agents, but arranged to reveal the underlying spatial structure.

4.3.3 Maintaining a PA's Spatial Information

When turning from blast to a regular PA, each place agent captures the current place signature as its base knowledge about the environment it represents. Similarly, the PA establishes its knowledge about each of its neighbors based on only a single position estimate: for its parent based on the trajectory it took while exploring space; for its children when they themselves explored space. Needless to say that such knowledge might be imprecise due to sensor noise during the single recording.

However, when the robot later returns to a place agent, this PA has control over the robot and can capture a current - updated - place signature. Ideally, this current place signature is identical to the PA's stored signature; but under real-world circumstances a new signature provides different information about the local environment. Hence, the PA can use the current view to update its own spatial knowledge. The data container introduced in chapter 0 - that maintains the PA's place signature - handles the details of such an update: the PA only needs to provide the current view, which it has received from the robot. The container inspects its currently stored data, estimates the relevance of the new data relative to existing data, and updates the maintained spatial knowledge appropriately. Repeating this process whenever the robot returns to a place agent consolidates the PA's knowledge about its local environment over time.

Alternatively, when parts of the local environment significantly changed, the PA might capture a completely new signature and memorize that two different constellations of this place exist. A good

example of such a scenario is a PA representing space close to a door: if the door is open the robot reports a completely different signature compared to when the door is closed. Refer to chapter 5.2.1 for details. In such a severe case, or even for just a small modification in the local place signature, a PA can re-evaluate if unexplored open space exists in its vicinity - and in case create a new child PA to explore such open space.

Similarly to updating spatial information in its local place signature, a PA regularly obtains updated distances and directions to its neighbors: whenever the robot reached this PA from one of its neighbors PA_N, or travels from this PA to its neighbor PA_N, one of these two PAs has integrated the trajectory the robot has taken. This PA can immediately update its knowledge about position of its neighbor: knowledge about neighbors is handled as a layer in the data container; so again the container handles the update. The PA only needs to communicate its current estimate of position and distance regarding its neighbor to its own place signature. Additionally, when the PA that received the robot sends a message to the sending PA notifying it about the distance traveled, the sending PA can update its information about its neighbor as well. Repeatedly traveling between two neighboring PAs, these PAs consolidate their respective position estimates.

With PAs updating knowledge about their local environment and about distances to their neighbors, the following problem might occur: a PA that is only ever visited from one side might slowly shift the center of space it represents towards its neighbor, especially if the robot repeatedly stops early. As an example, the nucleus PA in Figure 31 will later only ever get visited with the robot starting from its single neighbor PA_1. Because of a tolerance margin when comparing the robot's current position with the target position, the robot is likely to stop early on each visit. Updating PA_{Nuc}'s environment and the distance to its neighbor on every visit might get PA_{Nuc} to slowly drift closer to PA_1. In practice, however, this does not seem to be a problem. Due to sensor and actuator noise the robot's final positions when approaching PA_{Nuc} are equally distributed around the original center of PA_{Nuc}, such that after a few visits a stable position has emerged which only gets corrected marginally on consecutive visits. To be certain that we do not ignore potential problems, we additionally simulated an extreme example in which we update the PA's signature at every visit with only ½ of the true distance traveled. In such a scenario PA_{Nuc} indeed moves rapidly closer towards PA_1, but a least one of the following two correction mechanisms triggers: either PA_{Nuc} while moving downwards detects free space above itself, which is sufficiently large to create a new child for explore. Or PA_{Nuc} and PA_1 detect sufficient similarity to merge into one place agent (refer to chapter 5.2.4) - and thus again detect free space above to explore. Either of these mechanisms ensures that explored space does not get lost, but might be represented by a different place agent.

4.4 The Place Agent Network GUI - a Tool for Human Convenience

This subchapter describes a tool that is not required in a running system - in fact this tool stops the running system to comply with our constraint on local information processing only, and hence it will ultimately not be allowed in a running system. However, it is extremely valuable for debug and tuning purposes.

As explained throughout this chapter, all PAs exist as individual programs (Java threads) inside a computer, each of them only knowing about a small subset of other PAs. In the running system no actor is allowed global access to the system's spatial knowledge, which is distributed amongst all

individual PAs. For globally consistent operation, no actor in fact requires access beyond its local knowledge, as will be demonstrated in the upcoming chapters.

However, for a human observer developing the system it is highly convenient to inspect the collection of place agents, and investigate in their behavior: both, by monitoring individual agents and their knowledge, but also by pinpointed manipulations of individual PAs. Applying external disturbances to individual PAs, we can inspect the systems behavior more thoroughly and detect and fix existing problems. For this purpose we have developed a tool called “the Place Agents Network GUI” (PANGUI). If such a process exists on the computer hosting the PAs, each newborn PA registers itself at the PANGUI, and keeps the GUI up to date about its neighborhood relations in its own coordinate system: for each of its neighbors PA_N, every PA submits PA_N’s identity and the distance and direction in which it assumes PA_N to be.

PANGUI is an independent program that knows about all currently existing place agents and can - upon user request - generate statistics about the current collection of place agents. It can e.g. print a list of all currently existing PAs and their individual neighbors. Or - even more convenient - PANGUI can generate a consistent graphical representation of all PA’s spatial arrangement. Most of the network displays in this thesis are generated by the PANGUI using a simulated annealing technique to relax place agents to positions that best satisfy their individual neighboring constraints (for details refer to chapter 4.4.1).

Figure 39 shows a captured snapshot of the PANGUI displaying a growing network of place agents and their neighborhood relations. Green circles represent full PAs with the black dot pointing to their internal zero degree orientation, the diameter of such a green circle corresponds to the free space a PA detects at its center. Small circles in purple denote blast PAs that will explore space in the direction away from its only neighbor: its parent. Red arrows between two neighboring PAs indicate their neighborhood relation. The length of these vectors is scaled to 45% of the assumed distance between two neighboring PAs - such that for an ideal arrangement of PAs the tips of two opposing vectors just do not hit each other. Note that the misalignment of red arrows in the network is due to inconsistencies in all PA’s estimates of their neighbors’ positions and orientations, as will be discussed in detail in chapter 6.2. A relaxation technique applying simulated annealing (chapter 4.4.1) has found a good arrangement of place agents such that all neighborhood relations are clearly visible. The small purple blast PA at the very top showing two additional rings in blue and red currently has control over the robot, hence it is exploring open space in that direction. All the buttons on the right side for user interactions will be explained in the chapter 4.4.2.

Remember that the PANGUI is a tool for display and debug purposes only. It does have access to all spatial knowledge at once by maintaining references to all individual PAs that are distributed in the system. However, the tool is absolutely not required for operating the system. We have run several exploration trials without the PANGUI, and only afterwards inspected the resulting network.

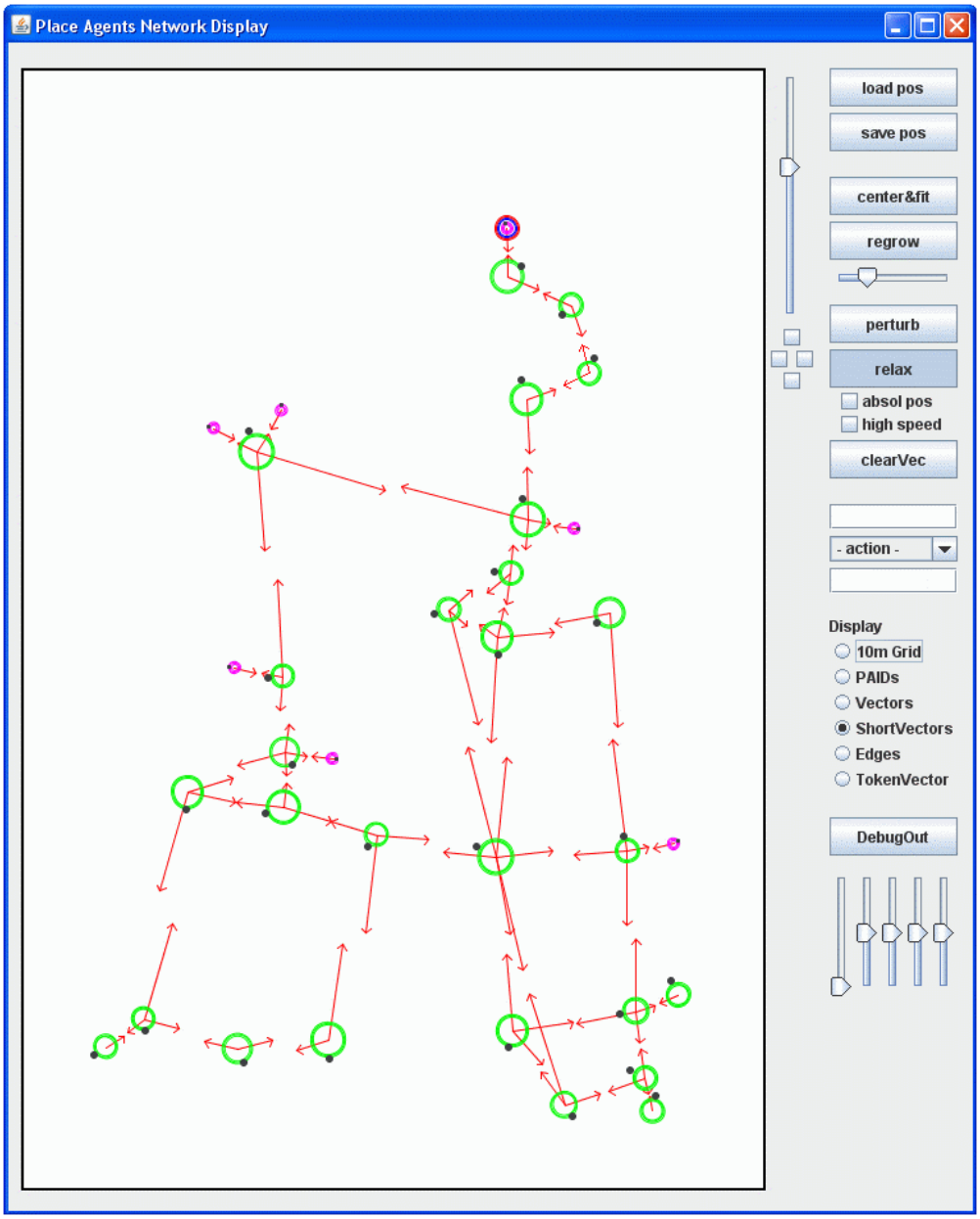


Figure 39: A screenshot of the place-agent-network GUI. The large panel shows all place agents (green and purple circles) with their neighborhood relations (red arrows). Buttons on the right-hand side allow inspecting the system and performing various manipulations to individual PAs. The PANGUI is a convenient tool, but not required to run the system.

4.4.1 Finding a Place Agent's Position on the Display

All place agents represent a true place in the robot's environment, but they are unaware of their position in absolute (world) coordinates. As an exception to this statement, place fields recorded by the simulator can in exceptional cases get tagged with their simulated position. However, we are ultimately concerned about a robot interacting in a real world environment in which such position information is not always accurately available.

This generates difficult task for the Place Agent GUI to find appropriate positions for PAs on the display window, such that all each PA's individual information about its neighbors is reflected in the arrangement. This is easy as long as we have no loops in the system: a simple algorithm picks one of

the agents and places it at a random point on the display. Afterwards, it adds all the PA's neighbors according to the PA's estimate of where they go in its coordinate system. Repeating this procedure for each new PA that is placed on the display, this algorithm "unfolds" a non-loopy network in a tree-like structure from an arbitrarily chosen root to its leaves. It does not matter which of the PAs serves as the root; ultimately a unique structure (except for an unknown orientation of the overall network) appears on the display.

Typically collections of place agents contain internal loops, as already visible in the tiny toy world example shown in Figure 33. How PAs establish such loops will be explained later in detail in chapter 5.2.4. Assuming all angles and distances between PAs are recorded correctly, a loop does not cause problems for the dispatch algorithm described above; it will simple stop expanding a branch if one of a new PA's neighbors is already placed on display. As angles and distances match perfectly well, the neighbor already exists at the correct position. However, as we will see later in the thesis, the assumption of correct angles and distances is not justifiable for real world scenarios. The PANGUI needs a more sophisticated technique, which iteratively reduces errors in positions and angles of the network, to discover appropriate places on the display for all its place agents.

For this technique, PANGUI initially puts all PAs at random positions on the display. Afterwards, PANGUI repeatedly picks a single place agent PA_R at random, estimates its display errors for position and orientation with respect to its neighbors' current positions on the display - and performs a tiny incremental correction of PA_R's position and orientation. Repeating this process reduces inconsistencies in the network display, but is likely to end in a local minimum: a situation with a suboptimal solution that does not improve because of counteracting forces. A technique called "Simulated Annealing" (Arbib 2003) helps to avoid those local minima by initially applying a relatively large noise term to the incremental correction, which later decays towards zero. Such a technique does not guarantee to stay out of local minima, but empirically showed that it is significantly less likely to get trapped in minima. The full algorithm is outlined in detail in Figure 40. Remember that we are not concerned about a perfect display of places and positions. This is a research problem on its own, and various solutions to relax a network of connected notes have been suggested (Duckett, Marsland et al. 2000; Hafner 2000; Golfarelli, Maio et al. 2001; Filliat and Meyer 2002). We are only interested in obtaining a display that allows us to identify neighborhood relations between groups of individual place agents. Even more, we do not use a consistent arrangement of the network for any navigational task, as we will discuss in detail later in chapter 1.

- Distribute all place agents at random positions on the network display
- Repeat:
  - Pick a random agent PA_R.
  - For this update, pick a random subset of PA_R's neighbors.
  - Define n := number of PAs in PA_R's random subset of neighbors.
  - Clear total position and orientation error for PA_R: $\Delta o_{tot} = \Delta p_{tot} = 0$.
  - For each neighbor $PA_N(i)$ in the subset of PA_R's neighbors, i = [1..n]:
    - Compute Δo_1 as the difference in orientation between PA_R's estimate of $PA_N(i)$'s position and $PA_N(i)$'s current position on display.
    - Compute Δo_2 as the offset in orientation between the vectors PA_R-$PA_N(i)$ and $PA_N(i)$-PA_R, based on PA_R's and $PA_N(i)$'s current position, orientation, and their respective position estimate of their neighbor.
    - Compute Δp_1 as position error between PA_R's assumed distance to $PA_N(i)$ and the true distance on display between both PAs.
    - Compute Δp_2 as the position error between PA_R's estimate of $PA_N(i)$'s position and $PA_N(i)$'s current position on display.
    - Compute Δp_3 as the position error between $PA_N(i)$'s estimate of PA_R's position and PA_R's current position on display.
    - Weight each of the error contributions Δo_1, Δo_2, Δp_1, Δp_2 and Δp_3 by their respective normalized significance, as adjusted by sliders in the PANGUI.
    - Sum the contribution of $PA_N(i)$'s position and orientation error to PA_R's total error estimate: $\Delta o_{tot} = \Delta o_{tot} + \Delta o_1 + \Delta o_2$, $\Delta p_{tot} = \Delta p_{tot} + \Delta p_1 + \Delta p_2 + \Delta p_3$.
  - Divide the total position and orientation error by the number of contributions: $\Delta o_{tot} = \Delta o_{tot} / (2 \cdot n)$, $\Delta p_{tot} = \Delta p_{tot} / (3 \cdot n)$.
  - Add noise to the total errors Δo_{tot}, Δp_{tot}; the amount of noise decreases correlated to according to progress in the simulated annealing.
  - Adjust position and orientation of PA_R relative to a tiny fraction of the computed noise-contaminated total errors Δo_{tot}, Δp_{tot}; we chose a fraction of 1/1000.
- Finish, when remaining global error in position and orientation saturates.

Figure 40: A diagram of the algorithm for incremental relaxation to find appropriate positions for all PAs on the display.

The above outlined algorithm performing slow relaxation for positions of PAs on the display works very well on small networks, and typically reaches equilibrium close to optimum in large networks, provided the initial display of such a large network is already in a reasonable configuration. This holds true for a growing network: If the network display starts together with the system consisting of only a nucleus PA that incrementally builds up a network, PANGUI constantly maintains a suitable representation. However, after saving and reloading such a network on hard disc, the position configuration can get lost and the PANGUI often has difficulties, relaxing a network with multiple loops into a final configuration. For solving such a case, we implemented a mechanism called "re-grow", which initially removes all agents from display except one randomly selected agent. Note that only the display of place agents is hidden - and hence their contribution to the relaxation task. The place agent itself does not notice such a change. Afterwards, PANGUI repeatedly picks an agent on display, and adds all of its neighbors to the active display, giving each new agent some time to relax to its position before adding further neighbors. Based on empirical evaluation, this is a highly successful algorithm for finding a globally consistent layout of place agents' positions. The re-grow mechanism allows finding an appropriate arrangement of place agents even for a running system

with a complex internal structure containing many loops as various examples later in the thesis show. The re-grow mechanism can reconstruct all examples of networks shown later in the thesis.

4.4.2 User Interactions with the Network of PAs

During development - and possibly later due to curiosity - we want to inspect the system and interact with individual place agents. Each of the buttons on the right side of Figure 39 allows triggering a unique function as explained in the following list:

- **load/save** stores and retrieves the current configuration of place agents on hard disc.
- **center&fit** adjust size and position of the current graphical representation of all place agents to fit inside the visible window.
- **regrow** allows re-computing all PAs' display positions from a single start as described in the previous chapter.
- **perturb** applies a small random noise to position and orientation of all current PAs on display, which might be sufficient to get the position-relaxation algorithm out of local minima.
- **relax** enables the relaxation mechanism described in the previous chapter. The tick-boxes below and the five vertical sliders at the bottom of the panel adjust parameter of the relaxation algorithm.
- **clearVec** wipes PANGUI's memory of temporary information that PAs communicated to the GUI. As example, a PA might have learned which of its neighbors takes the robot closer to a target. It communicates such insight to the GUI for display; over time, different targets will point towards different neighbors, such that for clarity we would like to wipe the display occasionally.

The "**action** selection" box below allows triggering actions that involve one or two place agents, which we can select by clicking on their representation in the display window. Their names appear in the box above or below the action selection box respectively. We can choose between the following actions, which require (1) or (2) place agents to be selected:

- **compare (2)** evaluates the similarity between two place agents' signatures, providing an estimate of their spatial offset and the likelihood of them representing an identical place only based on the stored information (independent of their position in the display).
- **grant / withdraw robot control (1)** hands control of the robot to a particular place agent, or removes robot control from all agents to stop the system.
- **connect / disconnect (2)** allows to artificially introduce or remove connections between the two selected place agents. In the case of creating a connection, the GUI calculates the position for each PA's new neighbor in the PA's coordinate frame based on both PA's current positions on the display.
- **search fusion partner (1)** network consolidation (such as closing loops) requires a process in which a place agents determines if another place agent in the network represents the same place, and potentially fuses/merges with the other PA (refer to chapter 5.2.4). Selecting this action triggers such a process by hand.

The section headed "**Display**" allows selecting the visibility of different pieces of information:

- **10m Grid** displays a regular grid on the background, allowing a quick comparison of a place agent's distance estimates to its neighbors against possibly available ground truth.
- **PAIDs** labels each PA on display with its identity.
- **Vector / short Vectors** displays red vectors of full length or 45% length respectively between all PAs and their respective neighbors, showing neighborhood relations and revealing inconsistencies in the network display.
- **Edges** draws blue lines between the centers of any set of two neighboring PAs, showing neighborhood relations.
- **Token Vectors** displays PANGUI's memory of temporary information that PAs communicated to the GUI (refer to "clearVec" above).

The last button on the right side "**DebugOut"** triggers a function in the Java program that we frequently update to collect information of a running system: since the PANGUI has access to all place agents, it can easily provide information that is distributed in the system to generate statistics. We use PANGUI for simple tasks such as counting the number of PAs or edges in a running system, but also for complex tasks such as detecting inconsistencies in angles and distances within the network.

The final interactive elements on the GUI, the long vertical slider to the top and the four tick-boxes below, help fitting the network display into the visible windows: the slider adjust the current zoom-level, whereas the tick boxes allow rotating the whole display to align it to one of the four compass directions.

4.5 Summary and Conclusion

This chapter introduced the main components in our system that provide a representation of environmental knowledge: Place Agents (PAs). Each of such a PA is an independent active computer program (here a Java thread) that represents a local small place of the robot's environment.

Each PA captures a signature of the local spatial structure at its place, which allows it to guide the robot in the vicinity of its particular place. Whenever the robot revisits, the responsible PA updates its spatial knowledge according to the robot's current sensory perceptions. In addition to local spatial structure, a PA memorizes abstract behaviorally relevant information for its local place, such as the existence of particular objects (e.g. a battery charger) or nearby open space the robot can traverse. A PA is the only instance in the system that knows what the robot can do at the place it represents and how the robot can perform particular tasks at its place.

All PAs in our system are descendants from a nucleus PA, which created copies of itself to explore unknown areas it detected in its own local place signature. Each such descendant PA in turn takes control of the robot to explore its assigned free space. During exploring, the new PA guides the robot into free space while carefully analyzing the robot's current local environment for behaviorally relevant occurrences. Detecting such, the new PA settles to represent the current place, and creates its own offspring to explore unknown areas found in its new spatial signature. Repeating this process incrementally builds up a complete network of PAs, in which each PA represents a behaviorally relevant place in an environment (see Figure 32).

Although a PA knows its local environment in its ego-centered coordinate frame well, it does not know where its place is within the global environment. There is no global origin, nor a global manager that aligns individual PAs in a common reference frame - in fact there is no single component of our software that knows about the existence of all PAs in our system. The network GUI (refer to chapter 4.4) allows to display and inspect all individual PAs and thus knows them all; but this tool is only required for debugging convenience; it is not required to run the navigation system.

Instead of global positions, each PA knows all its direct neighboring PAs and their spatial positions represented in its own coordinate frame. All such neighborhood relations between two PAs in the system are established by PAs that create their own descendants to explore nearby open space: by definition, these children of a PA - after taking the robot to explore and settle at their own place - stay the PA's neighbors and vice versa. This implies that a direct traversable path between any two neighboring PAs in the system exist, as the exploring child has guided the robot from its parent to its own place. The topology of the network of distributed place agents thus reflects traversable space in the robot's environment.

A PA guides the robot whenever it happens to be at or nearby the particular place represented by that PA. The robot is a passive element in our system; it provides sensor information to a current PA and receives driving commands from the same currently active PA. The PA compares the robot's currently perceived environment with its stored representation of its place, thereby computing appropriate local driving targets in the robots coordinate frame to perform elementary task such as centering at the place. Furthermore, a PA can communicate with its direct neighbors (refer to chapter 5.1.1) allowing it to send the robot to any one of its neighbors and hand over robot control.

The PAs introduced in this chapter each represent a local behaviorally relevant patch of a possibly large environment. They receive preprocessed behaviorally relevant spatial information provided by the robot (see chapter 2.7), and compute commands to produce local behavior for a robot at their local place. With the functionality discussed in this chapter, however, these PAs cannot generate globally consistent behavior such as guiding the robot to a remote target elsewhere in the network, simply a currently active PA typically does not know about another PA elsewhere in the network that represents the desired target. We will extend the local PAs' functionalities in the following chapter, allowing them to produce globally consistent behavior still only using their local knowledge and local communication.

5 Globally Consistent Behavior in the Network of Place Agents

In the previous chapters we have discussed individual place agents (PAs) that each represents a local place in an environment without loops. Each such a PA knows only its directly neighboring PAs, such that the collection of them implicitly establishes a network that reflects the topology of the underlying environment. In this chapter we will extend the PA's functionality by introducing a signaling mechanism between neighboring PAs. This allows representing more complex environments (e.g. those with loops), and ultimately allows the PAs applying their spatial knowledge for solving navigation problems: they need to produce globally consistent behaviors to be useful. We introduce and discuss communication methods between individual PAs in the network to achieve globally consistent behavior.

5.1 Communication in the Network of Place Agents

5.1.1 Local Communication between Neighboring PAs

In our system, a place agent knows about the existence of only a few other PAs: its neighbors. We define a small number of PAs to be "spatial neighbors" of a particular PA, because - during exploration - the robot traveled from one to the other on a simple trajectory, thereby establishing a neighborhood relation. As each PA has connections to its entire neighboring PAs, it can communicate with all its neighbors; but also only with its neighbors. It does not have references to other PAs in the network - in fact all PAs are unaware of all other PAs in the system, except their direct neighbors. This adheres to our concept of each actor in the system being restricted to local knowledge only.

In a simple scenario one might imagine a wire connecting two PAs, such that they can send a "wake-up" signal to each other, which forwards control of the robot. In our more complex Java implementation we introduced a method in which a PA can wrap an arbitrary message in a "message container" and send this to its neighbor of choice. The neighbor can un-wrap the original message, inspect it and take appropriate actions.

We have seen implementations of such message exchange earlier in chapter 4.2: during exploration, a parent PA sends a signal to one of its children, signaling to it that the robot is prepared to explore space. After exploration, the child (that turned itself into a full PA) replies with a message, notifying its parent about distance and angle it took the robot, such that appropriate neighborhood relations between child and parent can be established. Later, in the example where the robot traveled randomly in the network, PAs continue to communicate handover-of-control and distances and angles to their respective neighbors. Later in this chapter we will see how the simple mechanism of local neighbor communication is used to consolidate knowledge that is distributed in the network of place agents.

5.1.2 Communication beyond direct Neighbors

For various navigational tasks a robot might want to perform, communication with only a nearest neighbor is insufficient: e.g. finding a route to a distant target defined by some remote object such as a battery charger cannot typically be solved by local information only. The currently active place

agent PA_C might not know anything about the object in search, and its neighbors - with which PA_C can communicate - might not know about that object either. This chapter introduces two special types of messages, which travel through the network and - while on the way - updates their internal state based on locally available information, to find a globally consistent solution to the initial request. Yet, such messages at no point have access to or generate global information.

5.1.2.1 Token Finding a Globally Optimal Solution

We present the concept of tokens in the context of a simple behavior. Such behaviors get discussed in detail later in chapter 5.3; however, it is much easier to understand the concept of tokens when looking at a concrete example. The behavior "hide" would like to find a nearby dead-end in the network, take the robot to that dead-end and hide it; similarly to someone hiding in a corner.

Conceptually, a token is a container to propagate an evaluation function through the network, and - after some time - provide the best evaluation result found. Finding the local value of such an evaluation function at a particular PA typically involves local information stored at that PA together with information the token accumulated on the way while it travelled in the network.

Creating a Token

Two independent sources exist that create tokens: either a PA creates a token on its own initiative, which then typically performs network maintenance (chapter 5.2). Alternatively, an external active behavior (chapters 5.3) creates a behavior-token, which it gives to the currently active PA_C, in order to collect information that allows exhibiting the behavior properly. In either of these two cases, a single initial token exists at only a single PA. This PA evaluates the token, and - based on the evaluation result - replies to the token's creator immediately or propagates the token to all its neighbors. We denote such a propagated token an "outbound" token, as it travels away from its origin. When another PA receives such an outbound token, it again evaluates the token's function, and replies to the sender or propagates the token to its neighbors. Each PA that propagates outbound tokens ultimately receives a reply for each propagated token and - after receiving all replies - itself generates a reply for the neighboring PA that originally provided its outbound token.

Propagating Tokens

In our simple example where we want to hide the robot in a dead-end, the behavior "hide" creates a token that searches for a dead-end in space. A PA receiving such an outbound token can reply immediately, if it recognizes that it already represents a dead-end in space, i.e. when it only has a single neighbor - the one that sent the token. Alternatively, if the PA cannot fulfill the request, it propagates copies of the token to all its neighbors except the one that sent the token, to continue the search for a dead-end.

In a simple environment without loops (such as the toy-world presented in chapter 4.2), a nucleus token starts at PA_C and creates outbound tokens that travel to all terminal nodes in the existing network. Every PA that propagates tokens to its neighbors keeps a count of how many such outbound tokens it has sent, and awaits as many returning tokens. As soon as all such replies arrived, the PA can itself generate and send its reply to the neighbor from which it has received the token. If a PA cannot hide the robot itself, but one of its direct neighbors can, this PA replies "success in 1", stating that it offers to hide the robot in one token-hop distance, if the robot went to this place. Similarly for all other PAs in the network: if one of a PA's neighbors replies "success in X", this

PA itself will reply “success in (X+1)”, where it picks X to be the smallest value it has received from all its neighbors. This principle is illustrated in Figure 41.

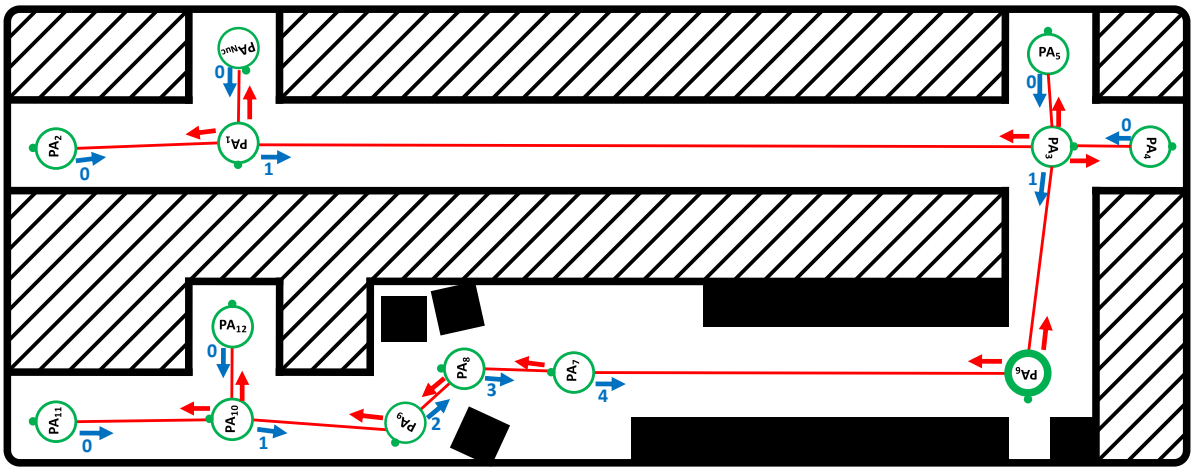


Figure 41: Searching for a place to hide in our toy-world example, starting at the current PA (as an example here PA6, indicated by a solid green circle in the bottom right corner). Note that in this chapter toy-world is slightly modified to explain token behavior in a simple test scenario. Red lines connecting PAs represent their neighborhood relations, red arrows denote outgoing tokens. Blue arrows show returning tokens, the numbers accompanying blue arrows display locally known smallest penalty regarding the evaluation function.

Ultimately, the PA that issued the initial token will have received all replies it waited for, and can itself find the best solution to the evaluation function. In the example in Figure 41, PA_C (=PA_6 at the bottom right corner) itself cannot hide the robot, but after some time receives two replies: “success in 1” from its neighbor PA_3, and “success in 4” from its neighbor PA_7. According to the token’s evaluation criterion - here distance in hop counts - “1” is better than “4”, so PA_6 knows that sending the robot to its neighbor PA_3 is what the behavior “hide” prefers to do at the current place.

Note that PA_6, after receiving all expected replies, still does not know how to send the robot to a hiding place: it does not know which PA the token found as target; neither does it know a path to that target. PA_6 only knows that the token selected to send the robot to its direct neighbor PA_3. Similarly, no other PA in the system knows a route to a target, except those PAs that identify themselves as target. Each PA only knows which of its neighbors the token determined to send the robot to. This communication method using tokens that travel in the network does not rely on global knowledge. All PAs and all tokens only have access to local information and incrementally build up distributed knowledge that solves the initial request (e.g. find a hiding place). No single token knows a path to a distant goal, no PA knows a path to a distant goal; but instead each token at a PA ultimately knows how to get the robot a little closer to a distant target.

Maintaining a Token’s Travelled Distance

In most real world scenarios, the hop count for tokens traveling in the network is a poor evaluation criterion since it does not reflect true distances the robot needs to travel. Assuming PA_7 as start in the same scenario shown in Figure 41, PA_7 will receive the smallest hop count from its neighbor PA_6, whereas the path to the nearest hiding place clearly leads towards PA_8. Instead of accumulating hop count, the token needs to accumulate distances between place agents that it traveled along the network in order to provide valid results for the request. Such information is locally available while the token travels in the network: whenever it moves from PA_X to PA_Y, both PAs know each other as

neighbor, and hence know their physical distance. We implemented a mechanism such that each token arriving at a new PA inspects this PA's knowledge about its neighbors, and internally accumulates the distance traveled. Figure 42 depicts a token's travel starting from PA7, returning accumulated distances instead of hop count. With the updated evaluation criterion PA7 clearly selects to send the robot towards PA8.

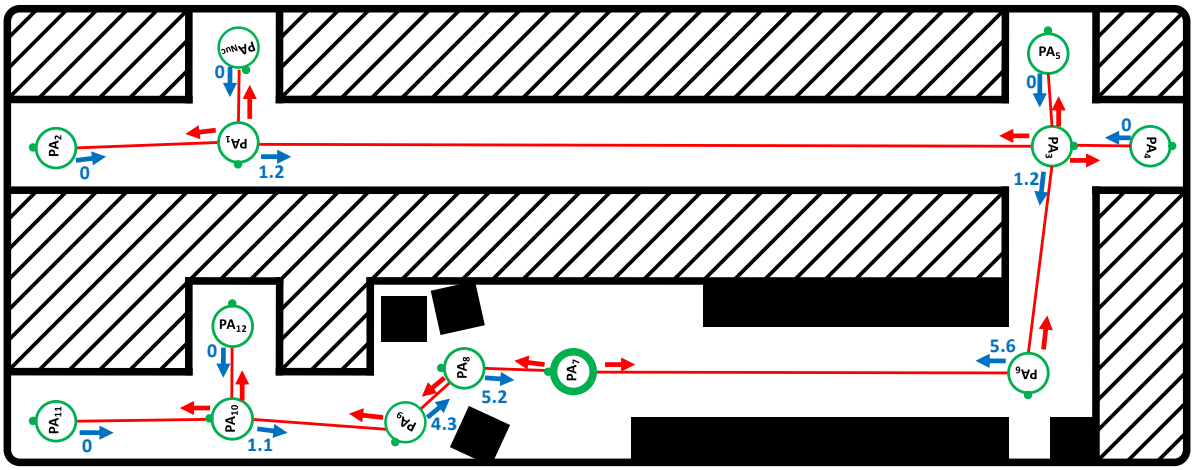


Figure 42: Searching for a place to hide in toy-world using physical distances as evaluation criterion instead of hop count. The numbers accompanying blue arrows show the evaluated distances to a target from the sending PA's perspective: PA_7 still needs to add the distances to its neighbors PA_6 and PA_8 to their respective answers before comparing the two returning tokens.

Tokens in Environments with Cycles

In a network that represents an environment with cycles (chapter 5.2.4), outgoing tokens will explore along several paths and surprisingly collide somewhere in the network. Figure 43 shows an exemplary network representing an office room in which the current PA_C initiates a search for "coffee" (instead of a place to hide), which will ultimately be acknowledge by a place agent representing the kitchen, PA_K. Two outgoing tokens shown by red arrows collide between the two PAs indicated by a large red exclamation mark. "Colliding" here means that these two PAs each propagate an outgoing token to their respective neighbor before receiving their neighbor's propagated outgoing token. Due to the asynchronous operation of PAs in the network we cannot make any assumption about where in a loop such a collision will occur, but we know that for every loop exactly two neighboring PAs in the loop will receive the same token from two directions. Simply put these PAs have to ensure that they notify each other about the duplicate token and determine, which of the two paths the token came from is more favorable. A special case occurs when one of these PAs already propagated outbound tokens to other neighbors, but later learned about a more favorable path to the token's origin. Such a PA needs to notify all of its neighbors about the new option, such that they all can re-compute all their tokens' evaluation functions.

A detailed explanation regarding the implementation of such tokens, the handling of exceptional situations, a pseudo-code implementation and a proof that such tokens find an optimal path with respect to their evaluation function can be found in Appendix B.

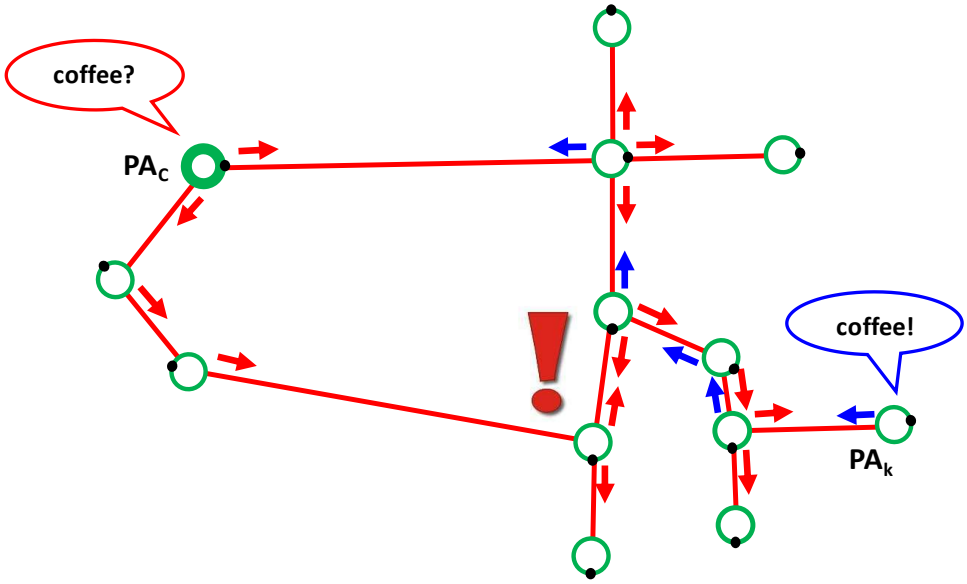


Figure 43: A network with a loop, showing a "collision" of outgoing tokens (red) to the right of the exclamation mark. Blue arrows show the optimal return path regarding this search for coffee.

Note that a place agent that receives a token does not itself understand the token. A PA only grants the token access to its local knowledge and executes the token's evaluation function; but it does not care if the token searches for a place to hide, tries to maintain the network's integrity (chapter 5.2), or evaluates anything that the PA has never heard of. Only a token's creator knows this token's purpose - and can ultimately perform appropriate behaviors based on the token's result.

Further note that such a token provides a result that is valid only on all PAs along the optimal path to the detected target. Nodes not participating in the optimal path might have non-optimal information with respect to this token's evaluation function, or might have no information about a target at all (see nodes without blue arrows in Figure 43). We need exactly such functionality for network maintenance (chapter 5.2), where PAs might decide to send tokens to selected neighbors only and thereby constrain the resulting path to start towards a particular neighbor (chapter 5.2.2). However, for a robot targeting request it seems beneficial that after such a search process all PAs involved have gained potentially applicable insight about the search. We present a different method of querying the network in the following chapter that generates target-directed information in each node of the network.

5.1.2.2 Local Invitations Establishing a Gradient in the Network

This chapter describes a completely different approach allowing PAs to reason about information that is only represented in remote locations within the network. In contrast to tokens described above this method generates a gradient in the network of PAs towards one or multiple target locations, which allows each PA in the network to improve the robot's position with respect to the current request by sending it to one of its neighbors. This gradient is generated only by exchanging messages between neighboring PAs, so called "invitations". These invitations include cost for a particular commodity (e.g. coffee), similar to the evaluation function used by tokens. Due to the asynchronous nature of message passing, we cannot determine when a PA received a final solution; but on the other hand a PA will learn about an intermittent solution as quickly as possible. This process of exchanging invitations for a particular commodity is much simpler compared to the tokens from above.

This method only uses one single type of message that we call an “invitation”. An invitation contains in identifier of the commodity it searches (e.g. coffee), an evaluation function, and the current best available result of a local evaluation of that evaluation function. The identifier for the commodity and the evaluation function are designed by the initiator of the search and do not change, whereas the current best available result is updated at each PA.

If any PA would like to find the local gradient towards a new commodity, it simply sends an “invitation” to all of its neighbors which starts the process of building up a gradient. This invitation contains an identifier for the commodity, an evaluation function and a current (initial) best cost. A typical evaluation function returns zero if the commodity is locally available, otherwise returns the best evaluation-value amongst all such known values from the PA’s neighbors incremented by the distance to that neighbor. The initial cost is typically infinite, as the initiator PA does not know about the commodity and has not yet received information (invitations) from its neighbors.

A PA receiving such an invitation re-computes the evaluation function, including the newly gained knowledge from its neighbor, and performs one of the following three options:

- If the re-evaluated cost is better than what it currently knows, it remembers this cost as current best result (CBR) and the sender as best neighbor. It propagates the CBR to all its neighbors, except the sender.
- If the re-evaluated cost is worse than its CBR, it returns an invitation to the sender, offering its own CBR. “Much worse” here denotes that the PA expects its own result to be better for the sender. The PA needs to take the distance between itself and the sender into account to compute the sender’s CBR based on its current CBR. Details in appendix C.
- If neither of these cases applies the PA takes no further action.

Once started, this simple principle establishes a gradient amongst all nodes in the network to possibly multiple targets. The network self-organizes so that each source of the “targets” is the root of a tree covering the region in which it is the closet source. Once the PA that initiated the search receives a valid invitation from one of its neighbors it can start sending the robot along that path. The PA might receive additional invitations to even better targets later and will rearrange its local gradient. In practice, establishing the gradient is substantially faster than sending the robot anywhere, such that the robot will start towards the best target immediately. Figure 44 shows how invitations establish a gradient to solve the same search for coffee as presented for tokens in Figure 43. Appendix C explains the invitations in more details, handles exceptional cases such as changing distances or the topology and shows pseudo-code implementing invitations.

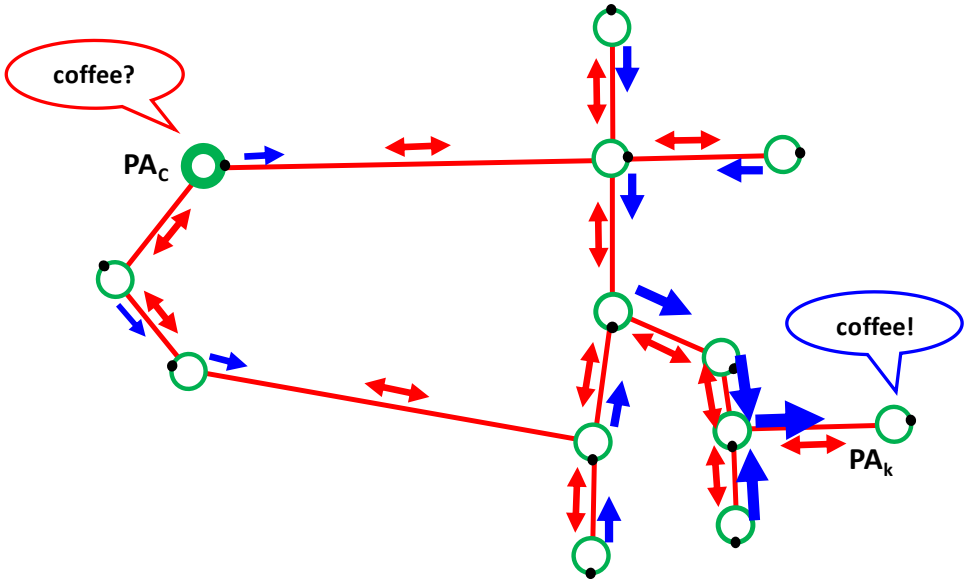


Figure 44: Neighboring invitations (red vectors) setting up a gradient (blue vectors) in the network towards the only source of coffee available.

5.1.3 Limiting a Query's Outreach

Both examples for querying the network above ultimately involve every place agent that exists in the network. For most searches, however, a local outreach is sufficient: if the robot is looking for a battery charger, it can limit the search to those PAs that represent places within reach of the remaining battery capacity. Similarly, if we want a coffee we will typically approach one of the nearby kitchens instead of exploring all options for coffee in the world. In all these cases, the creator of the request is aware of such a limiting factor and can model this - if existing - in the original token or invitation. Further propagation of messages stops after violating such a criterion, e.g. the distance covered, by returning a non-successful reply. This ensures that the search stays limited to within a reasonable area instead of flooding the whole network.

5.2 Network Maintenance

5.2.1 Temporarily Disabling Connections

When a current PA (PA_C) hands over robot control to one of its neighboring PAs (PA_N), but PA_N fails to attract the robot within reasonable time for the distance between the two PAs, PA_C returns the robot to its own place, starting from PA_N's current estimates of the robot's position. Such a failure can happen due to a modified environment (e.g. a closed door or a chair blocking the path) - or because the trajectory discovered initially during exploration is too difficult to repeat. Upon such a scenario, both PAs have several choices how to continue:

If the current place agent has clear evidence that the obstruction is temporary (e.g. as reported by a sensor on the robot detecting doors, or by careful reasoning about changes in the place signature), both place agents can mark the connection between them as temporary unavailable and re-run the current behavior (see chapter 5.3). Tokens and invitations (chapter 5.1.2) that a behavior employs to find a trajectory recognize the temporarily disabled connection and find an appropriate detour by ignoring the connection. The place agents can decide to re-enable such a disabled connection after

some time, or only when the robot revisits the place and detects open space. In a more advanced version (which we have not implemented yet), a PA can maintain two or more independent place signatures of the single place it represents, and apply selectively a signature based on binary attributes of current environmental features: door open or closed.

In most cases, however, PA_C will not be able to detect such a clear binary situation: it will not know a reason why the robot failed to reach its neighbor. In such a scenario, PA_C at first increments a penalty attached to that connection, and afterwards re-invokes the current behavior. The behavior, by issuing a new token or a new invitation, re-evaluates its options to reach the target starting from the current place, but taking the new penalty into account. The behavior might decide to try the same connection once again, or instead favor a detour that might be longer but more save to travel. If a behavior immediately or later tries the same path and the robot succeeds, PA_C reduces and ultimately clears the assigned penalty. In contrast, if the robot fails again, PA_C further increases the penalty for that connection, making it less likely for the current behavior to choose this path once more - unless no other path towards the target exists.

Note that such an assigned penalty for a connection is direction-sensitive: A path might be very difficult to follow from PA_X to PA_Y, but might be very easy the other way. Therefore, only the PA that sent the robot towards its neighbor updates its locally stored penalty for this connection.

After several unsuccessful trials, a PA might determine a path unusable, and prefer to completely remove this path (and thus its neighbor) from its spatial knowledge. Such an action is a severe modification to a grown network of places, which needs very careful examination regarding network integrity as shown in the following chapter.

5.2.2 Removing Connections between Place Agents

Removing connections between neighboring PAs modifies the network structure, such that in an extreme scenario the network might fall apart in two disjoint groups. Once a network has separated in two parts, the group that contains a PA currently controlling the robot continues guiding the robot in space; but no member of this group has evidence of further PAs existing in the other group. Therefore, those PAs can never get rejoined with the group in control of the robot. All PAs in this group will never again have a means of interacting with the world and thus turned useless. Obviously, this shall never happen!

Before two PAs (denoted here by PA_A and PA_B) can agree to remove their connection, they need to verify that the integrity of the network remains intact, i.e. that afterwards still only a single group of connected PAs remains. If one of the PAs happens to be a terminal leave of the network, the connection can get removed immediately with the terminal leave deceasing afterwards. If each of the PAs has at least one additional neighbor, the PAs need to determine if another path exists that connects them besides their direct connection: Either one of them (here PA_A) creates a search token with a cost function that is equally expensive for all PAs the token arrives - forcing the token to explore the whole network for a possibly better solution. PA_A gives this token to all its neighbors except PA_B. The token explores all outbound paths in the network, and might or might not arrive at PA_B. If the token arrives at PA_B, at least one alternative path from PA_A to PA_B exists, such that both PAs can safely remove their direct connection without damaging the network's topology. If the token does not arrive at PA_B, but instead PA_A receives all expected return tokens, the connection

under investigation is the solely link between two otherwise disjoint subgroups. Both PAs need to keep the connection alive, possibly labeling it with a high cost for traversing. Both PAs can occasionally repeat the search, as other PAs might have continued exploring space and might have found an additional connection between the two groups of PAs, finally allowing both PAs to remove their connection.

5.2.3 Re-Exploring Space

Once a connection between two neighboring PAs is established, each of these PA assumes that space in-between is not relevant for behavior, and upon demand only sends the robot all the way to its respective neighbor. However, the blast PA exploring the initial trajectory might have missed an important aspect along the way, or a new object might have appeared in the environment. Therefore it seems relevant to occasionally re-explore space in-between two already connected place agents, which otherwise would be ignored forever.

To re-explore such space, a group of two PAs can create a common blast child; an extension of a normal blast PA (chapter 4.1.2) that is created to explore unknown space. Such a special blast PA has two parents instead of one, but still does not represent an environment upon creation. When receiving control of the robot, this blast child guides the robot from its current position towards the other PA, just as a normal travel between the two PAs would do. While guiding the robot the blast PA inspects the currently perceived environment just as any blast PA does during exploration. In case a behaviorally relevant event occurs before reaching the target, the blast PA transforms itself into a full PA representing this place and the new event. A new PA, already linked to two existing PAs, adds itself to the network and again inspects its local environment for further space to explore. In contrast, if the blast PA arrives at the target without detecting a relevant event, it will hand over control to the target PA and decease, leaving the graph unchanged.

Such a re-exploration might detect an entry to a previously completely unknown area of the environment, e.g. one that was blocked by a previously closed door, or contribute additional relevant objects to the existing network, thereby increasing the overall value of the spatial representation for the robot.

5.2.4 Closing Loops

Detecting and closing cycles in an environment is a difficult challenge for globally organized spatial representations (Thrun 2002). The locally distributed system we present treats this process very differently, simplifying computation drastically.

Imagine a robot explores an environment and surprisingly assumes that it has arrived at a place it has previously visited: its perceived representation of the current environment is suspiciously similar to the perception it had at a previously recorded place. Are these places really identical? In case of unique landmarks in the world - or markers left in the environment by the robot - the system can unambiguously decide if the two potentially identical perceptions are indeed from one place. Typically such help does not exist, so the system is left with uncertainty whether the two perceptions come from one place or two similar yet different places.

A navigation system can use path integration as an additional measure to determine if two places are identical. Unfortunately, such path integration along the trajectory a robot traveled is strongly

contaminated by accumulated noise: even small initial angular errors quickly result in large position offsets between assumed and true position. When arriving at a possibly previously visited place, path integration might give a further indication to whether the two places are identical, but is absolutely insufficient to base a decision upon.

Besides the problematic uncertainty, the real challenge only starts when arriving at a previously visited place: a navigation system maintaining a globally consistent map has to immediately decide whether to treat such possibly identical places as a single place, or to remember them as separate places. In the first case, the system has to perform some expensive computation for map-correction: i.e. relaxing the accumulated position error along the path the robot has taken to maintain a globally consistent environment. Note that several proposed systems (Thrun 2002) store all sensed data along all trajectories taken, and perform offline post-processing of all that data to generate a single optimized representation of an environment. This is a viable solution for a computationally advanced system, but does by no means reflect a biologically way of building up spatial knowledge.

Our concept approaches the problem of loop closing very differently. In fact, due to the distributed nature of local place agents, the concept of a loop does not exist anywhere in our system. No single actor in our system realizes that there is a loop in the representation, because a neighborhood relation between two PAs cannot constitute a loop. Therefore, we do not need to perform corrections of accumulated error along the trajectory.

Multiple Representations for a Single Place

We still need to implement a mechanism to detect that the robot surprisingly revisited a place, and afterwards adapt the network of PAs appropriately. Such a situation only occurs in our system when a blast PA (PA_{NEW}) guides the robot to explore space and decides to represent a place that is already represented by another PA (PA_{OLD}) somewhere in the system. Within the current framework, neither PA_{NEW} nor PA_{OLD} knows about the respective other PA, so PA_{NEW} continues exploring its environment and rebuilds an identical network, just as PA_{OLD} has done before. This chapter introduces a method that allows PA_{NEW} to detect such a situation and ultimately merge with PA_{OLD} into a single place representation that contains the combined spatial knowledge and all neighborhood relations as seen by PA_{OLD} and PA_{NEW} together.

An example to illustrate a loop-closing scenario is shown in Figure 45, where the network of PAs already explored most of a modified toy-world as explained in chapter 4.2. Note that the modified toy-world contains a cycle. PA_{11} is about to hand over control to its child PA_{13}, which will continue exploring unrepresented space upwards along the aisle. PA_{13} currently does not know about PA_1, which already represents the other end of the aisle; PA_{13} only knows its parent PA_{11}. Remember that the spatial arrangement of nodes shown in Figure 45 only happens for our convenience; the network of PAs itself is not aware of such an arrangement, and instead only exists as 14 place agent threads floating somewhere in a computer's memory.

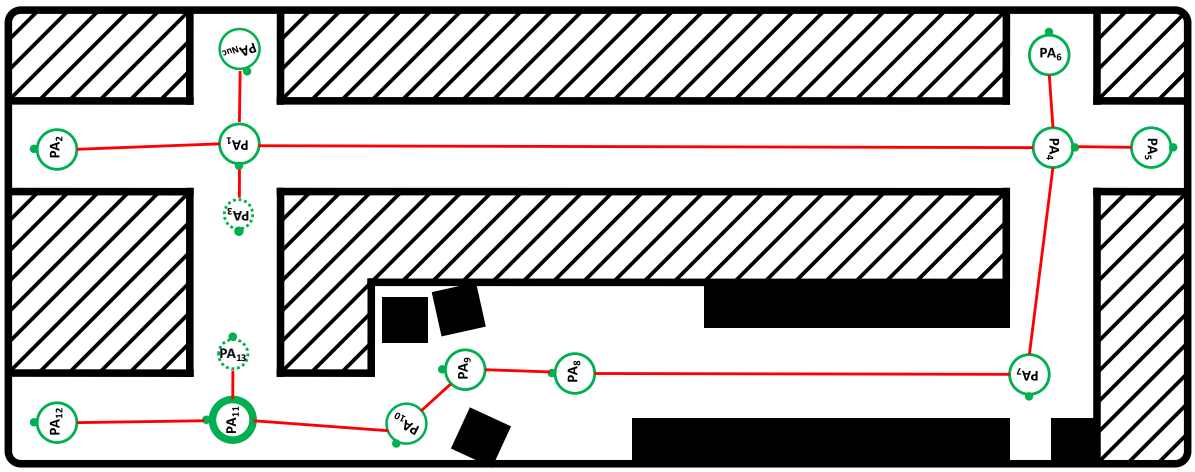


Figure 45: A modified toy world scenario, with a loop in the environment. The network of place agents has explored the world as before, but has not finished yet. PA_{11} is currently controlling the robot, but will shortly hand over control to its child PA_{13} which will continue exploring upwards.

Receiving robot control, PA_{13} explores unknown space in the aisle ahead (upwards in Figure 45). During this exploration, PA_{13} determines the aisle to be behaviorally irrelevant, and thus continues guiding the robot forwards until it reaches the intersection. Reaching the intersection, PA_{13} settles in the center of open space (refer to chapter 4.2.1), turning itself into a full PA that represents this place (see Figure 46). PA_{13}'s exact spatial position might be very close to the place represented by PA_1 or further away, as these two PAs independently of each other decided to represent this place based on the environmental signature reported by the robot at the time of creation. Neither of the two PAs knows about the other PA, so initially the network contains two independent PAs that both represent the same place.

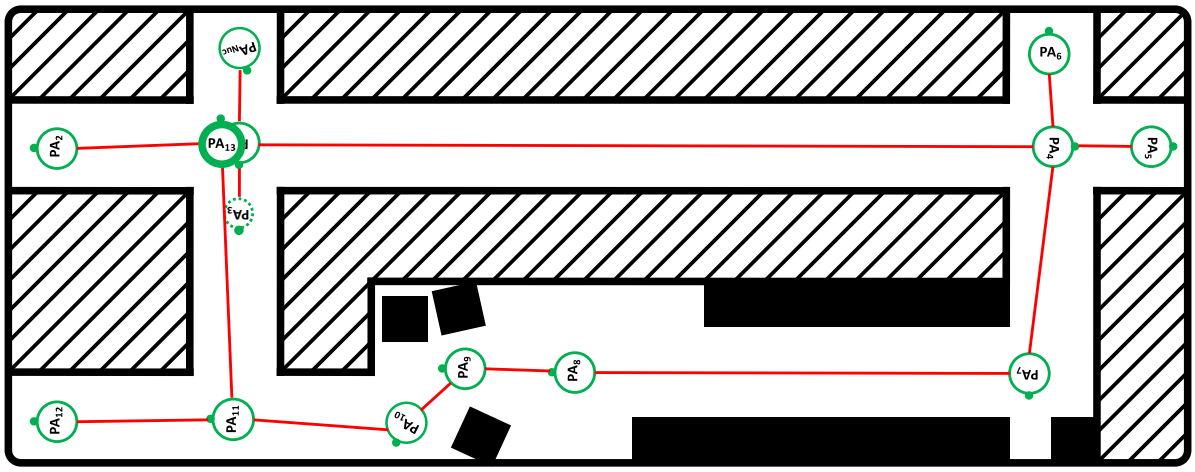


Figure 46: PA_{13} finished exploration and detected a behaviorally significant place to represent. It is unaware that this place is already represented by another PA (PA_1), as it currently only knows its parent PA_{11}.

Detecting Multiple Representations

Only an agent that recently transformed from a blast PA into a full PA contributed a new place to the network, which might or might not have impact on the network's topology. So this agent - PA_{13} in our example - needs to initiate appropriate actions to maintain the network's integrity: upon representing a new place, each PA should inquire in the network if the proposed new place is already represented elsewhere. PA_{13} generates a token (chapter 5.1.2) that contains a copy of its

own place signature, and broadcasts this token among the place agents in the network. Each PA receiving such a token compares its own spatial representation against the provided place signature. Figure 47 shows the propagation of such a token in the network (red arrows); all PAs exceeding a similarity threshold are marked by '!'; those with a low similarity with 'x'. Note that here two place agents, PA_1 and PA_4, see their local environment sufficiently similar to the new environment explored by PA_{13}. All these three place agents represent a place at an intersection of two aisles, so their three environments indeed are very similar. Assuming a lower similarity threshold, PA_7 in addition might see the proposed new place to be sufficiently similar to its own environment. This example shows that estimating the similarity between two PAs solely based on their place signatures is not sufficient to decide if both PAs represent the same place or different places in an environment. PAs that compare their own signature against the signature provided by a token need additional information, e.g. about their spatial distance to the origin of that token.

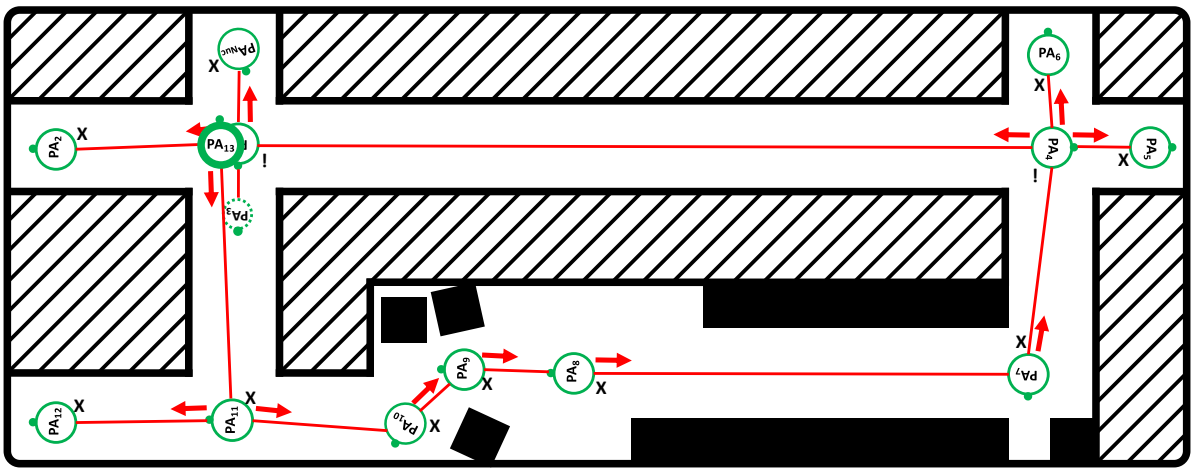


Figure 47: PA_{13} has sent a token travelling in the network, propagating PA_{13}'s view of its proposed environment. All other PAs upon receiving this token compare their spatial representation against the proposed new place, and evaluate similarity. High similarity is indicated by '!', low similarity by 'x'. Note that the horizontal red vector behind PA_{13} shows the propagated token from PA_1 to PA_2; PA_{13} does not know PA_2; so it can not send a token to PA_2.

As we have seen before, tokens can accumulate locally available knowledge while they travel in the network, e.g. integrate local distances between neighboring PAs to maintain an estimate of the total distance they have traveled (chapter 5.1.2.1). Whenever a token arrives at a particular PA, this token can also inspect the PA's local information about angles between its neighbors. Integrating angle and distance information, a token can keep track of its individual "homing vector" - a vector pointing towards the origin of that token. Additionally, such a homing vector can estimate the current PA's orientation offset with respect to the origin PA's orientation, simply by integrating all angles alone that it traveled along the path, independent of distance traveled. Maintaining such a vector only requires adding currently available local knowledge in the current PA's coordinate frame. Although such a vector has much of a global flavor, we argue that - just as accumulating distances - it is only a new piece of local information. If the token instead remembered a trajectory consisting of various waypoints or edges we reject that as global knowledge. Figure 48 shows computed homing vectors of tokens for all PAs in the network.

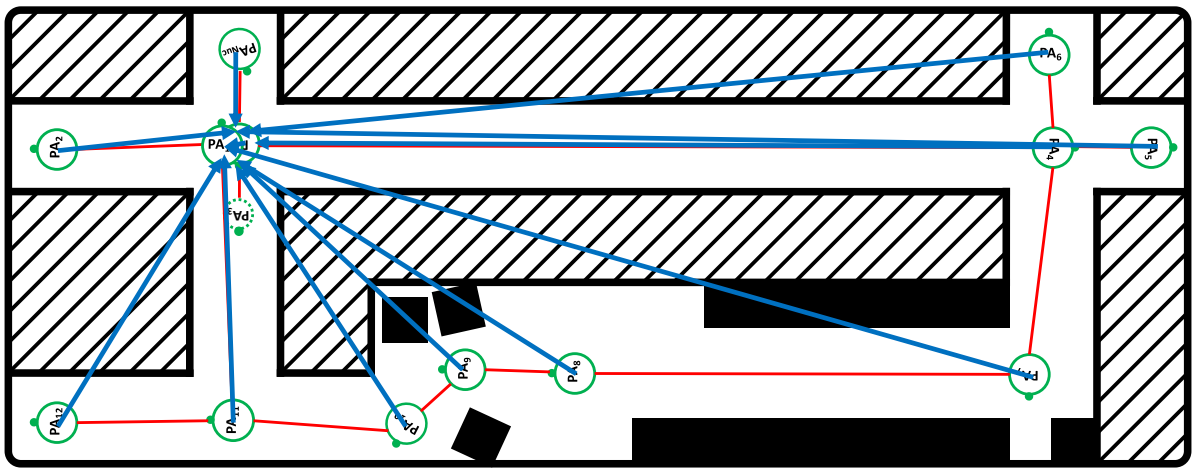


Figure 48: Each token arriving at a PA updates an internal homing vector (blue). Upon arrival, the token represents the old homing vector (computed at the PA's neighbor) in the current PA's coordinate frame, and adds the current PA's angle and distance information about that neighbor. Such an update only requires local knowledge available at the current PA, which results in an incrementally updated homing vector, permanently pointing towards the origin of that token.

The figure above does not display the ground truth, which here is sacrificed for clarity. In a network that does not yet contain loops the arrangement of all PAs in such a red graph is "perfect", meaning that each PA's recorded neighborhood information is reflected correctly in the graph's spatial layout. As a consequence, all blue vectors point exactly towards the token's origin (PA_{13}) - the blue vectors are computed only based on knowledge that is perfectly represented in the graph. However, in contrast to the figure, the network of place agents will be skewed with respect to the underlying true spatial environment. This makes all blue vectors point to a single place that might be far away from the point represented by PA1, although in reality PA_1 and PA_{13} represent the same place. We present a much more detailed discussion about such a network skew in chapter 6.2.4. Later, when an already existing network of place agents implicitly represents loops - e.g. the network represents cycles in the environment without explicitly knowing that it does - the tokens' estimates of their origin differ along various paths they took in the network. Hence, the figure above showing blue vectors pointing at various places near the token's true origin reflects the true situation well for this explanation.

Merging PAs

Any PA receiving such a token - here denoted as PA_X - has two sources of information to determine how similar its own place is compared to that of the remote PA_{NEW}. Firstly, said PA_X can judge whether its place is anywhere close to PA_{NEW}, based on the token's homing vector: only if the length of the vector falls below a threshold, PA_X continues determining its similarity with PA_{NEW}. We empirically found a good absolute threshold to be 10 times the robot's body length (here a total of 3,2m). But uncertainty of homing vectors increases with their token's traveled distance; so in addition we apply a threshold relative to the distance traveled: if the length of a token's homing vector falls below 10% of the token's accumulated path, PA_X continues testing for similarity. Both these thresholds are rather generous, since this first step is a necessary, but not a sufficient condition to determine that PA_{NEW} and PA_X represent a single place.

Note that in a network of PAs without angular or distal errors, a token's homing vector at a PA that represents the exact same place as PA_{NEW} has a length of zero; however, if PA_X happens to

represents a place slightly offset to PA_{NEW} (as shown in Figure 48, PA_1), its homing vector still is tiny in length. In contrast, homing vectors for most other PAs in the system (including the doubtful candidate PA_4) have a length significantly above threshold.

Secondly, if successful in the first case, PA_X compares its own local place signature against the place signature provided by the token (refer to chapter 2.6): it compares the similarity of the two places' representations. This comparison provides two results:

- A scalar value for overall similarity
- A 3-dimensional vector providing the spatial offset ($\boldsymbol{so}$), consisting of distance and angle for the translation difference and an angle for the difference in orientation, represented in PA_X's coordinate frame. If the two place signatures happen to be taken at the exact same position, their translation difference is zero; however the robot might have faced a different orientation, hence their difference in orientation is unpredictable.

These results - together with the earlier obtained homing vector towards PA_{NEW} - allow PA_X to determine whether they both represent a single place with high certainty: If the scalar value of similarity falls below an empirically found level, PA_X stops a potential fusion of itself and PA_{NEW}. Otherwise, PA_X computes the PAs' position and orientation error by subtracting the token's homing vector from the vector of spatial offset ($\boldsymbol{so}$). This position error reflects a possibly different true position of PA_X and PA_{NEW}: in our example shown in Figure 48 comparing the place signatures from PA_{13} and PA_1 shows a position offset of maybe one meter, as the two PAs simply represent slightly different places in space. Both, the token's homing vector and $\boldsymbol{so}$ show this offset, such that the remaining position error approaches zero.

To answer the original question whether PA_X and PA_{NEW} represent the same place, PA_X can take a decision based on absolute and relative position error, absolute and relative orientation error, absolute distance, and overall similarity of the places' signatures. A measure of relative orientation is meaningless, as all PAs are allowed their own coordinate system; such that no matter what relative orientation PA_X and PA_{NEW} have they still can represent the same place. Note that the computed orientation error between two PAs is different to the orientation difference of these two PAs: in the first case, the error reflects the mismatch between the orientation difference predicted by tokens and the orientation difference predicted by comparing both place signatures, which ideally should go to zero.

In those rare occasions that all the above criteria fall below empirically determined thresholds, PA_X assumes that PA_{NEW} represents the same place and returns a reply token to PA_{NEW}, offering itself as a partner for fusion. PA_{NEW}, which either receives none, one, or multiple replies offering fusion, can select the best reply and evaluate for itself if the reply is of sufficient quality for fusion, similarly as PA_X did. In case both PAs agree to fuse, either of them creates a common child, and merges both their spatial information as reported by the token into a common representation for the child. Afterwards, both original PAs decease, leaving a single child that represents the place, but knows about both PA's neighbors (shown in Figure 49). Note that the remaining blast PA (PA_3, one of PA_1's children) has vanished: as merging two PAs potentially provides new neighbors for the resulting PA, the new PA instructs all remaining children to terminate, and instead inspects its local environment to possibly create new children representing space that yet remains to be explored.

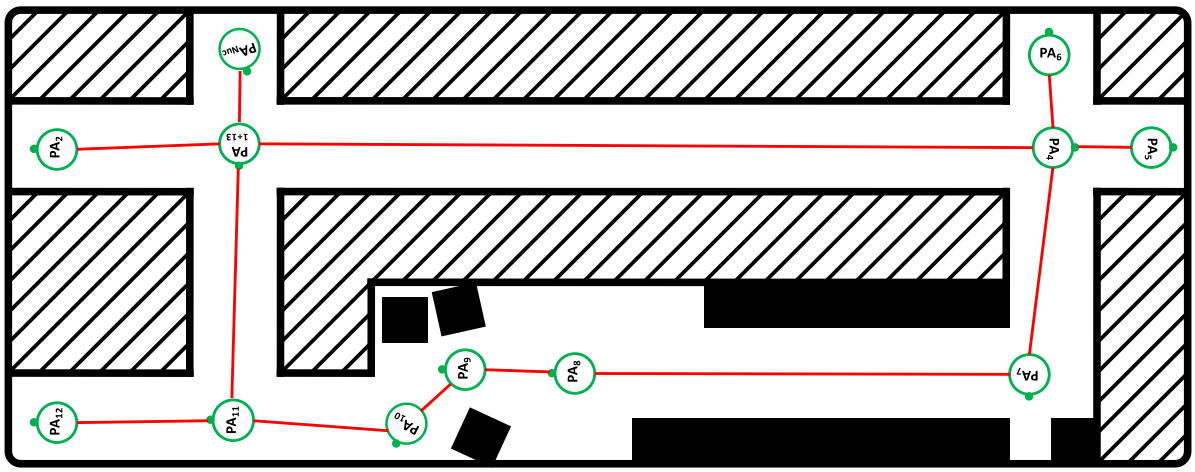


Figure 49: Finished fusion: PA_{13} and PA_1 determined that they represent the same place in the environment. They created a new child, here called "PA1+13", that has all the environmental information that both parents provided, including all their neighbors.

Looking at the figure above we see that the new PA_{1+13} closes a loop in the spatial representation. In reality, the new PA_{1+13} no more or less closes a loop than does any other PA in the loop we see. Remember that the figure is only a display for us; all PAs only know their direct neighbors, but neither of them perceives the loop. Every PA with more than a single neighbor (i.e. every non-terminal PA in the graph) is potentially part of a loop that we as external observer can detect by thoroughly inspecting all distributed knowledge at once.

Note that in our example not only PA_{13} sent out such a token, but all other PAs that have previously transformed from blast to a full PA. So far, in an environment without cycles, all such searches ended unsuccessful, so we have neglected them.

Failing to Merge PAs

Completing the fusion story, we look at a more or less exceptional case in which PA_{NEW} and PA_X decide to be insufficiently similar for fusion, although they do represent the same place in an environment. This might happen due to accumulated error in the token's integrated path, or to an insufficiently rich place signature. In either case, the two PAs do not fuse, and PA_{NEW} represents the same place as PA_X already does, not knowing about each other. PA_{NEW} continues as if it represented a new place - which it anyways assumes it does - and inspects its spatial signature for further directions to explore. Figure 50 shows this scenario, in which PA_{13} has missed to fuse with PA_1, and instead created three children to explore its unknown environment. PA_{13}'s children continue exploration, and typically settle at a place that one of PA_1's children already represent. When these new children inquire in the network for a fusion partner, they are likely to find PA_1's earlier children representing those places, and fuse with those. Similarly for PA_1's remaining child PA_3: it will find PA_{13}'s parent for fusion. This situation is shown in Figure 51.

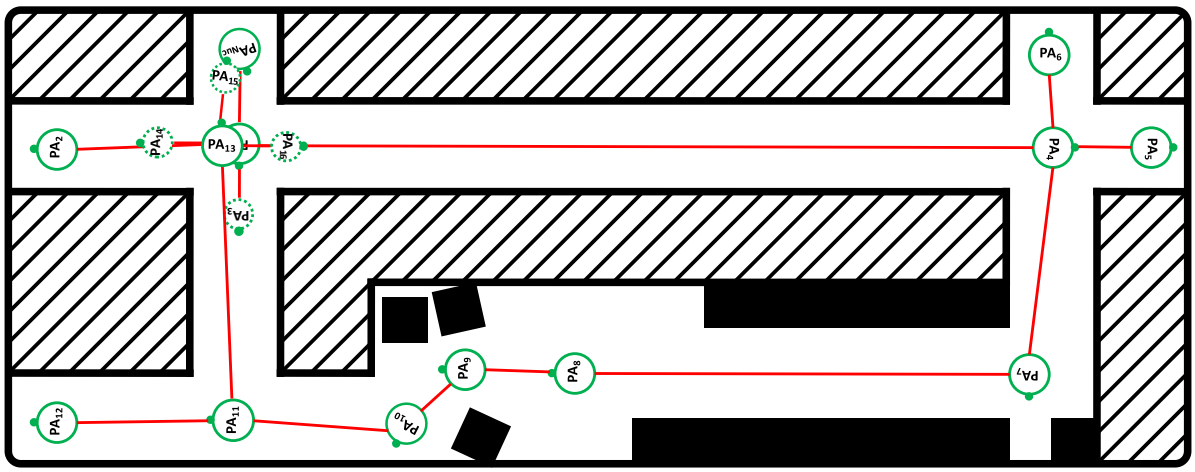


Figure 50: PA_{13} and PA_1 missed fusion because of mismatch in position and/or spatial signature. PA_{13} created children (PA_{14}, PA_{15}, and PA_{16}) to continue exploring its surrounding unknown space. Additionally, PA_1 still has a blast child (PA_3) that will explore unknown space.

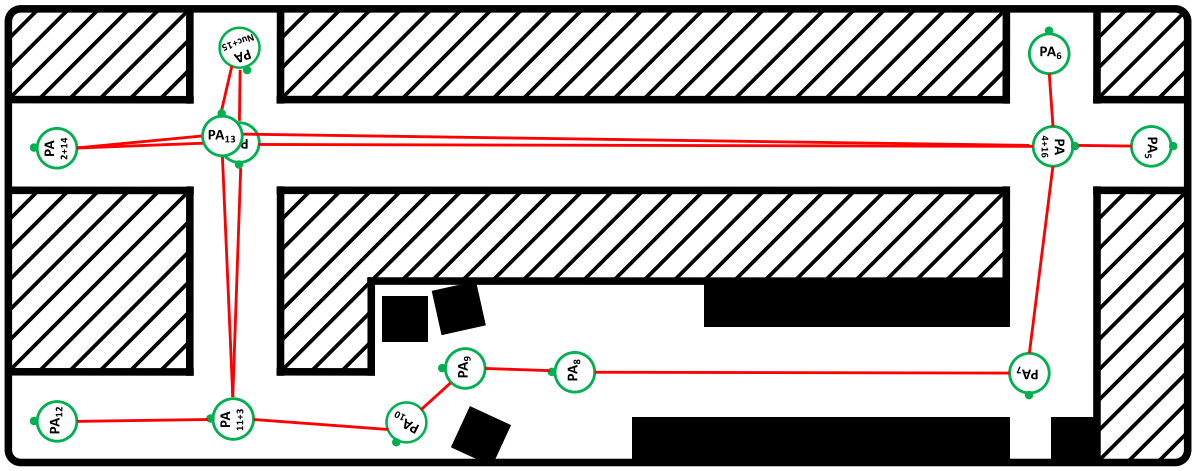


Figure 51: PA_{13} and PA_1 missed fusion; they continued exploration. Their neighboring PAs - after their exploration - might fuse or might not; here for clarity all their neighbors have fused. PA_{13} and PA_1 (which is halfway covered by PA_{13}) might fuse at some time in the future, or simply exist as independent place agents for good without causing harm to the network.

PA_{13} and PA_1 remain in the network, both representing the same place, each knowing about the same four neighbors. Such a situation does not render the system malfunctioning; in fact, it might even improve the network's effectiveness regarding way-finding, as multiple redundant trajectories exist that each show different lengths depending on start and target PA. In a running system, however, the robot continuously re-visits place agents and updates their knowledge about space: A PAs place signature captures more detailed spatial information, e.g. landmarks that happened to get missed initially; and the PA refines its estimates about distances and angles to its neighbors. Sooner or later the spatial knowledge of two PAs representing the same place grows sufficiently similar to allow fusion. As all PAs in the network occasionally re-initiate a search for fusion partners, ultimately these initially resistant PAs decide to fuse at a later point in time.

Summary: Closing Loops

We have presented a method that keeps our distributed network of place agents consistent with the external environment, when the robot surprisingly detects that it returned to a previously visited

place. Other navigation systems typically respond to such an event by invoking a computationally expensive special treatment (Thrun 2002) to keep their global spatial representation consistent. We, instead, ignore the fact that the environment contains a loop, and only update local neighborhood relations of PAs. We do not need to relax error that accumulated in the network while traveling along the loop.

The various empirically determined thresholds introduced in this chapter might concern a critical reader. However, it turned out that all these thresholds yield adequate results in a surprisingly large range, showing the system's robustness. The only threshold that has a significant impact on the final topology of a network is the maximal permitted distance between two PAs to grant fusion. Multiple blast PAs arriving at a single place during exploration typically determine slightly different final places to represent, especially in those environments containing larger open spaces. In such situations, the threshold determines whether to fuse any pair of such PAs or to keep them separate. Hence, in less structured environments with open space, this threshold modulates the density of place agents employed to represent space.

In the running system we set all the above thresholds rather high, as this will only delay or avoid fusion of PAs that are not (yet) sufficiently similar, without limiting the network's functionality. In the alternate case, when accidentally two PAs fuse that in reality represent different places, the situation is much more difficult to correct. We have already introduced mechanisms that ultimately correct such problems, as e.g. removing connections that the system determines un-traversable after several attempts (chapter 5.2.2). In general, we prefer to keep place agents separate as long as they have some remaining uncertainty to whether they represent the same place. This is one of the great advantages of our system: We can delay the fusion decision easily, and even if the fusion does not happen at all, it will still work correctly.

5.3 Exhibiting Globally Consistent Behavior

We have so far presented how a place agent represents a local place, and how a collection of such independent PAs generates and maintains a distributed spatial representation of a global environment. We still need to explain how these PAs enable the robot to perform globally consistent navigation behavior. The following chapters describe how abstract processes - each representing a typical behavior - interact with PAs to find possibly distant targets in the spatial representation and to guide the robot to those. All such behavior processes at any time only interact with a single place agent: the currently active PA_C that is currently controlling the robot. Otherwise, if behaviors were allowed to interact with all PAs simultaneously, such behaviors violated the concept of a truly distributed system. Alternatively, we can think of these behaviors interacting with the robot - which itself is interacting with the currently active PA_C. In our particular implementation we regard the robot as a passive component of the system, and therefore interface behaviors directly to PA_C.

We have implemented a collection of those behaviors as independent computer programs, which each represent one of various typical tasks a robot performs, such as targeting, exploring, foraging, and homing. All these behaviors constantly compete for activity, and only a single winning behavior guides the robot at any time (details in chapter 5.4). However, when explaining the details of designing behavioral processes below we assume that for simplicity only one of them is active at a time and all others are completely decoupled from the system.

All behaviors interact with the currently active PA_C by exchanging messages as introduced in chapter 5.1. This existing framework of tokens and invitations allows behavior processes to collect all required information and to guide the robot. Therefore, the following description of behaviors is astonishingly brief, despite the fact that 'behaving in an environment' is the most important aspect for our system: we do not set out to draw beautiful maps of an environment; instead we aim to provide a spatial representation that is suited for behaviorally relevant actions in the world. This chapter describes how behaviors make use of the developed distributed representation.

Note that these behaviors describe a basic set of functionality required to build up spatial navigation and to show basic interaction with the world. They are by no means a complete set of behavior that a robot can perform on such a spatial representation. Most likely a robot system designed for a particular task adds particular behaviors to the basic set; as an example, a vacuum cleaning household robot might add a vacuum-cleaning behavior, in which it records the amount of dirt found at particular places and revisits places more or less regularly, according to the expected amount of dirt - or only revisits places based on a particular time schedule.

5.3.1 Finding Targets at Distant Places

The target behavior guides the robot to a particular place in the network of place agents, determined by a particular stimulus or a previously recorded marker. Typically neither the robot nor the currently active PAC knows the target in advance. As a simple example, let us assume that during exploration the robot detected coffee machines in the environment as behaviorally relevant stimuli, and left a label "coffee" in the place signatures of those PAs that represent these places. At a later stage, the robot wants to fetch coffee (or we want to robot to bring coffee) - so we have a remote target somewhere in the network described by the label "coffee" - but the current PA does not know where in the network this label exists. The targeting behavior guides the robot to go to such distal targets.

The targeting behavior designs a cost function that evaluates each PA's local knowledge and computes a penalty for each PA in the network with respect to the current search. For such a simple search scenario, a PA's penalty is zero if coffee exists, or is the sum of the minimal penalty of one of its neighbors and the distance to this particular neighbor. Giving and "invitation" with such a cost function to the currently active PA_C establishes a gradient towards sources of coffee within the network as explained in detail in chapter 5.1.2.2. Note that a PA receiving such an invitation does not need to understand the meaning of "coffee"; each PA only evaluates the function designed by the targeting behavior, providing local information from its local place signature.

Here we have introduced a label "coffee" that denotes places which offer coffee. In real world applications the robot unlikely finds labeled places that it understands. Instead, a robot might store video images taken at a particular place in the PA representing this place. The PA memorizes such images as passive data, i.e. without interpreting or reacting on them. Several of such images taken in the kitchen might include a coffee machine. When later searching for coffee, we provide a template picture of a coffee machine, and design a more complex cost function: One that inspects a PA's memorized knowledge, using computer vision techniques to find coffee machines in images. Only those PAs that provide a satisfying solution for the complex cost function report themselves as source for coffee, and thus as a source when establishing the network's gradient.

The above described simple cost function is sufficient to find the closest target represented by a particular stimulus. Typically, multiple sources of different quality exist, as easily demonstrated with the coffee example. In such a scenario, targeting behavior can generate an arbitrarily complex cost function, forming an overall cost that takes various individual evaluations into account, such as quality of coffee, price, distance etc. It is even possible to search for different targets at the same time: either a coffee or a glass of coke might satisfy the robot. All this can be represented in a complex cost function.

Remember that in this framework the individual place agents do not understand - and do not a priory represent - particular targets. They capture aspects of their local environment in multiple forms: one of which is the place signature they operate upon, but also in others such as images of places or particular labels associated with a place (as will be shown again in chapter 5.3.4). Only the cost function, carried by a token or an invitation, is able to interpret the data it needs to evaluate a place's value. Typically, this cost function is designed by the same actor in the system that also stored the abstract information at a place - and thus knows how to interpret this information.

5.3.1.1 Homing

Homing is a sub-function of the generic targeting behavior: upon birth, the mobile robot stores a particular abstract label at the nucleus PA. Whenever it needs to return home, the robot initiates the targeting behavior to search for the home label.

5.3.1.2 Target and Return

In various typical scenarios, a robot shall not only go to a target, but after performing a task at the target (prepare coffee) return to the current place - wherever the current place happens to be. This fits neatly in the existing targeting framework: upon starting such a task, the robot can drop an extra abstract label at the current place agent, possibly describing the task ("bring coffee here"). After finding and preparing coffee, it simply starts a new targeting behavior searching for the previously dropped label. Upon returning to the unique place in the network that contains the label, the robot knows that it returned to the place from which it started. After completing the task the robot removes the label from the current place. Note that the system might initiate different behaviors in-between (such as charging batteries), but still manages to successfully return to the stored label afterwards - possibly with cold coffee.

5.3.2 Run and Hide

The targeting behavior described above uses invitations (chapter 5.1.2.2) to build a gradient to distant goals, and initially it might seem that such invitations are sufficient to fulfill any navigation request in the system. In this chapter we present a behavior "run and hide" that requires a different method to find the best target place: imagine a robot (or an animal) received a shock caused by a strong stimulus, such as a loud noise. Such a robot wants to run away from its current place and hide in a shelter, here defined as any terminal node in the network. A cost function to determine the best such shelter requires combining two different quantities: an optimal distance to run and the fact that a place provides a shelter. The robot should not hide in the nearest possible dead-end, as this might be too close to the source of the stimulus. With invitations, however, the cost function evaluated at a particular PA has no information about the distance between the robot's current PA (which started the invitation) and the particular PA evaluating the cost function.

Using tokens instead of invitations to search for a shelter solves this problem: tokens collect and integrate information along their particular outbound paths, such that whenever a PA evaluates the cost function based on its locally available knowledge, it can include the distance a robot had to travel from its current position to this particular PA's position. The "run and hide" behavior can design a cost function that returns minimal values for a given desired distance - say 10m - and such allows finding the shelter that is closest to a 10m escape run from the robot's current position.

5.3.3 Explore, Re-Explore

Exploration (and re-exploration) of unknown space is performed autonomously by place agents as described in chapters 4.2 and 5.2.3 respectively. The system still requires an "exploration" behavior that competes with all other behaviors for activity to grant those place agents that want to explore proper attention.

The exploration behavior differs from all previous behaviors in such that it needs to inspect the existing network of places prior to competing for activity. All other behaviors presented so far are triggered by an external stimulus, such as desire for coffee or a loud noise. The exploration behavior, in contrast, determines its current relevance based on the number and distance of unexplored places present in the network. Even when inactive, this behavior occasionally emits a token to the currently active PA, which counts the number of detected blast place agents (ref chapter 4.1.2) that still need to explore space, possibly weighted by traveled distance. Emitting such a token has no direct impact on the robot's current behavior; this token operates in the network in parallel to other currently active tokens or invitations (see chapter 5.4). Upon returning to the exploration behavior, such a token carries information about how important exploration is for the overall system - and thus how strongly the behavior shall compete for attendance.

After successfully competing against other behaviors, the exploration behavior generates an "invitation for exploration": a search for the nearest unexplored space (similar to the search for coffee, described in chapter 5.1.2.2), and gives such an invitation to the currently active place agent. The network establishes a gradient towards unexplored spaces, which leads the robot to a blast PA that explores its unknown space as described in chapter 4.2.

Re-exploration shall examine the network and find those behaviorally relevant places that it missed during initial exploration; hence by definition, re-exploration does not a priori know where those places are. The behavior initially selects random links between neighboring PAs and sends the robot to re-traverse such links searching for behaviorally relevant occurrences (see chapter 5.2.3). After unsuccessfully traversing an existing path without spotting a new event, the behavior increments a counter assigned to the link it traversed that stored in the PA that represents the place at the end of the segment. Later, the re-exploration behavior can check those counters remotely stored in the PAs by issuing a token - just as "exploration" does - thereby reducing the likelihood of re-exploring already re-explored segments. Note that such a counter in the PA is direction-sensitive; i.e. the behavior can re-examine paths in either of two possible directions independently, and possibly spot occurrences that it will only detect facing a particular direction.

5.3.4 Network Consolidation

Our system occasionally consolidates its knowledge about the existing spatial representation by revisiting paths and nodes in the network, or at least within locally limited regions of the network.

On such visits, the robot can give updated current place signatures to the involved place agents, allowing them to detect changes or simply improve the certainty about their local spatial environment. The robot's travel for such consolidation can happen as a random walk between place agents, but such a simple strategy does not guarantee that all places - and especially all paths connecting places - get visited at roughly equal occurrences. We would like to implement a behavior "consolidate" that sends the robot to all place agents once, or alternatively sends the robot along each existing path in the network once.

Such a behavior for network consolidation needs to determine where the robot already visited in order to determine where it still needs to go. However, a single actor (here the behavior) that ultimately memorizes all PAs or all paths of the network for a further decision operates on global knowledge - so we have to avoid that the consolidation behavior collects all such information. But we can allow the behavior to label a place that the robot currently visits by leaving a marker in the place's local PA. Having such markers at already visited places allows the behavior to design a cost function that distinguishes between visited and unvisited places or paths. Sending an invitation with such a cost function to the currently active PA sets up a gradient in the network which initially only consists of local minima everywhere, but updates its values once the robot starts exploring and leaves markers.

Note that such a simple principle by no means finds an optimal path to visit all places. Optimizing the path required to visit a number of places is known in computational mathematics as the "traveling salesman" problem(Cook ; Cormen, Leiserson et al. 1990; Applegate, Bixby et al. 2007), which is arguably one of the most difficult problems to solve efficiently. We are satisfied providing a solution that ultimately visits each place in a distributed network once - and without any of the involved actors operating on global knowledge such as a list of all places visited.

5.3.5 Forage for Food

In the context of our robotic system we define foraging as searching for food or other supplies at various locations, memorizing the food contribution of explored places, and ultimately having the option to return to either those places for consumption. We have already introduced individual blocks required to perform such a behavior. During initial exploration and later re-exploration, the system builds and refines a network of relevant places that represent particular features in the robot's environment, which may well include food or items such as a battery charger reported by the robot's sensors. The foraging behavior can place a marker in the current PA's spatial signature, denoting the presence (or absence) of food. An active foraging behavior directs the robot to inspect those places that have no yet a label regarding their food contribution, such that ultimately the network's distributed knowledge about feeding places increases.

For the second aspect - guiding a hungry robot to food - the foraging behavior creates a token or an invitation, similar to a search token. Such a modified search token contains a cost function that reflects the value of particular food at distant places, which the foraging behavior knows how to interpret because it stored the marker at such remote feeding sites, while the robot previously visited those and detected food. In contrast to the simple search mechanism (e.g. for a coffee machine as described in chapter 5.3.1), the foraging behavior needs to update its marker stored in a PA's memory once the robot consumed food available at that place: it completely removes the marker if all food is gone, or simply reduces the representation of the quantity of available food at

the place. While foraging, the active foraging behavior might decide that - due to the reduced amount of food locally available - a distant place is a better food source, and such guide the robot to another place to continue foraging.

5.3.6 Longest of all Shortest Paths - a Two Token Problem

So far we have expressed all behavioral tasks with a single cost function that tokens or invitations optimize with respect to distributed information in the network. Unfortunately, not all such behaviors can be expressed by a single cost function. As an example, consider a behavior that wants to guide the robot to the most distant place in the network that it can reach without a detour from its current position; i.e. find the longest amongst all shortest paths to any place in the network. For such a behavior a token must determine for each PA in the network its respective shortest path to the current robot position, and afterwards select the longest of all these paths. Such a cost function contradicts itself: initially the shorter the path the better; later the longer the path the better. A place agent cannot autonomously decide based on its knowledge at which point in time to switch between these two criteria.

Figure 52 shows an example network in which a token searching for such a path fails. Starting at the currently active PA_C on the left, by inspection the global network we can easily see that the PA in the top right corner is farthest away from the start. Conveniently, this PA happens to be labeled PA_T; but in the task PA_T acts like any other PA, and the token that the behavior generates has to determine that PA_T is the target. Such a token accumulates its traveled outbound distance when exploring the network for possible targets. Due to asynchronous operation of place agents in the network, we cannot predict in which order PA_5 and PA_9 will receive tokens and act upon them. The following scenario is a valid processing order, that ultimately reports the yellow trajectory as longest amongst all shortest - despite the yellow trajectory not being a legal shortest trajectory to PA_T:

- An outbound token along the yellow route arrives at PA_9, reporting the distance along yellow. PA_9 propagates this outbound token to PA_5 and PA_T.
- PA_T - being a terminal node - replies to PA_9 its total distance estimate, based on knowledge accumulated along the yellow path. PA_9, upon receiving PA_T's reply, waits for its other neighbor PA_5 to reply.
- PA_5, however, receives an outbound token along the red trajectory before receiving PA_9's token. It forwards its own outbound token to PA_9 and PA_4, reporting the outbound distance along the red path.
- All outbound tokens from PA_C along the green path need more time to arrive at PA_5 and PA_9; we only see their influence much later.
- PA_9 and PA_5 respectively refuse each other's token, as they each have already seen a better outbound path: PA_9 along the yellow route, PA_5 along the red route.
- PA_9 thus received replies to both its outbound token (sent to PA_T and PA_9), so it computes its own return token sending its best information back along the yellow path, all the way to PA_C.
- Finally, PA_5 receives outbound token along the green path and learns about a shorter outbound path to PA_C (compared to its previous best estimate along the red path). It propagates a new outbound token to PA_9. However, PA_9 has already sent its return token along the yellow path.

- PA_5, PA_9, and PA_T only now discover the true shortest path from PA_C to PA_T and return this path to PA_C along the path.

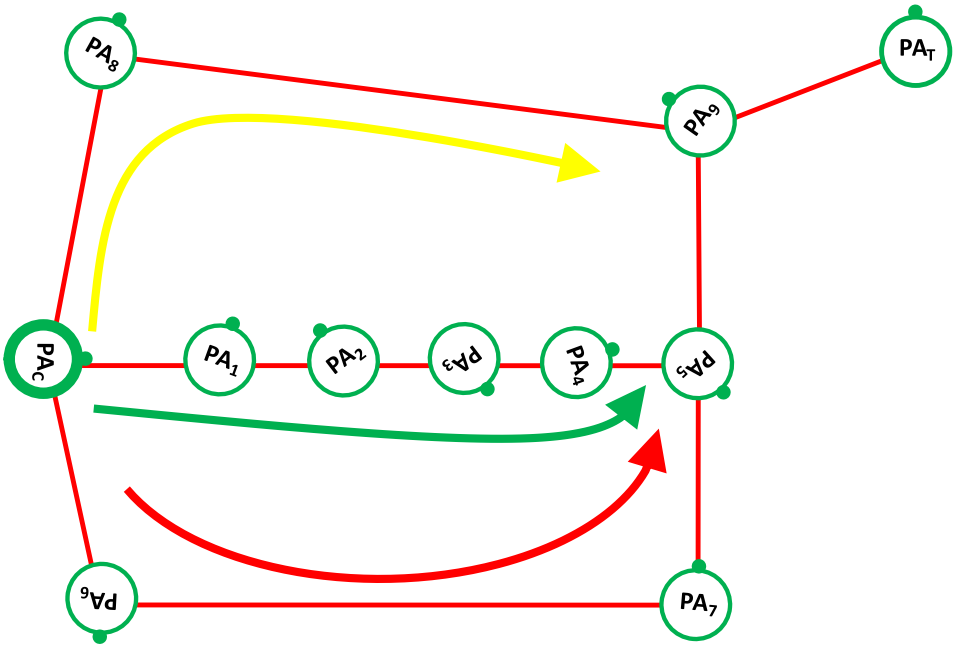


Figure 52: Finding the longest among all shortest paths to the most distant PA. A simple network example, in which a single token might report an incorrect shortest trajectory (yellow) from the robot's current PA_C to the most distant target PA_T. A two token solution correctly reports green to be the shortest trajectory. See text for details.

In this scenario, PA_C waits for all its replies and finally receives two different messages signaling "longest of all shortest path here", which return along the yellow and the green path respectively. If we were searching for the overall shortest path to a distal target (as targeting in chapter 5.3.1 does), PA_C selects the shorter of the two replies and thus finds the true shortest path. However, here we are searching for the longest path amongst all possible replies, so PA_C picks the longer of the two replies. It might receive a reply from a different place much further away that also returns from its yellow neighbor PA_8 - so it has to pick the longer offer. PA_C selects the longest path it receives, although this path is not in the set of shortest paths to PA_T. The problem arises because PA_9 switched from initially searching shortest paths to reporting longest paths before it know that all PAs have found their final shortest distance.

A network as shown in Figure 52 seems bizarre and artificially designed to illustrate problems that do not arise in reality. But in fact due to asynchronous operation of all place agents, we often see situations in which a neighbor of PA_C receives the first outbound token along a loop consisting of various other PAs, instead of directly from PA_C. In such a situation, PA_C might have sent the token to one of its other neighbors, and got interrupted by the computer's scheduling mechanism afterwards. We cannot predict when PA_C gets reactivated and continues propagating tokens to its other neighbors. Therefore, when designing such algorithms, we cannot make any predictions about the order in which tokens travel and report results.

Once we detected and analyzed the problem, a solution was quickly at hand: the troublesome behavior issues multiple consecutive tokens to solve the two contradicting problems independently. In this example, the behavior creates an initial token that determines minimal distances to each

place agent in the network. Such a token has no cost assigned for a particular place, but only accumulates traveled distance along its various outbound paths, which each PA maintains for some time in its memory. Once this token finished (and reported a useless reply to the behavior indicating PA_C's nearest neighbor having the shortest distance to PA_C), the behavior issues another token (or alternatively an invitation), which computes each place agent's cost based on the shortest distance as previously determined by the initial token.

Applying such a two-stage process guarantees that a place's final value is only ever computed based on the true shortest distance that the robot can take to that place; instead of a distance an earlier token reported that might have taken along a detour in the network.

5.4 Behavior Selection

In the previous chapter we have introduced various behaviors related to navigation, but we have not yet discussed how the system picks a winner amongst all behaviors that is allowed to control the robot. In a typical application scenario the winner is selected upon external input, e.g. a human operator pressing a button "bring coffee" or "clean now". Even in a standalone autonomous system - such as a home-cleaning robot - a pre-programmed fixed behavioral pattern typically suffices to generate desired functionality: initially high relevance for exploration, later higher relevance for cleaning with occasional re-exploration.

Here, in contrast, we want to demonstrate a complete test system that autonomously behaves in an initially unknown environment, so we decided to implement a simple behavior-arbitration mechanism that complies with our completely approach to act only based on distributed local knowledge.

Our simple behavior-selection mechanism needs to explore an environment while keeping the robot alive - i.e. arbitrate behaviors such that basic requirements (charging battery) are satisfied, but also apply a well-balanced tradeoff between exploration and consolidation of spatial knowledge (chapter 6.1.3) to create a comprehensive representation of external space. External input at any time can modulate the arbitration mechanism, thus allowing the robot to perform a particular requested task.

All implemented behaviors exist as independent programs in our system, such that we can easily add or remove behaviors. All behavior threads are constrained to interact with the network of PAs only through interfaces provided by a "Behavior Maintenance Thread" (BMT), see Figure 53. Such interfaces guarantee that behaviors do not collect global information, nor operate outside the scope of the currently active place agent. BMT implements two interfaces - a scanner and a motivator - which provide behaviors access to the currently active place agent.

5.4.1 Behaviors Scanning the Network for Distributed Information

For competition amongst behaviors, each behavior computes a measure that reflects its current significance for activity based on information it retrieved from the system. As examples, a battery-charging behavior reports increased significance for low on-board power, whereas the exploration behavior described in chapter 5.3.3 increases its significance proportionally to existing unexplored spaces in the network. Many of such behaviors need information that only exist distributed in PAs

within the network or on the robot to compute their relevance value, but do not have direct access to all PAs.

The BMT provides a "scanner" that accepts tokens (chapter 5.1.2.1) from any behavior, gives each such a token to the PA_C for evaluation, and returns the result reported by PA_C to the behavior that generated the token. We refer to those token as scanning tokens, as they inspect the network for behaviorally relevant information. Besides them originating from a behavior instead of a PA, they are no different to ordinary tokens (chapter 5.1.2.1).

Using this mechanism, all behaviors can inspect distributed knowledge in the network, and use retrieved information to compute their respective relevance values. Note that all behaviors can simultaneously inspect various aspects stored in the network independent of each other at any time and at their discretion. They can inquire information that they themselves created earlier, seen e.g. in the "re-exploration" behavior (chapter 5.3.3) that labeled links to decrease the frequency of revisiting those. But behaviors can also fetch external information, such as currently visible objects, previously recorded objects, or the robot's currently reported battery charge level. Forcing all behaviors to communicate through BMT instead of directly with PAs allows us to guarantee that behaviors only interact with a single member of the network instead of simultaneously addressing multiple PAs. For a different principle how behaviors might interacting with the network refer to chapter 7.3.

5.4.2 A Single Behavior Controlling the Robot

At any time only a single behavior shall control the robot; otherwise competing targets might result in an arbitrarily moving robot. In our example implementation, a behavior only issues commands that guide the robot from its current PA_C to one of its neighbors; typically the one neighbor that gets the robot closer to the behavior's determined target. Actions to be taken at the target PA - such as drinking coffee or charging batteries - get executed by the PA that represents the target place (refer to chapter 7.3).

We require behaviors to report their current requests for actions, which is typically based on information collected from the network of PAs (see previous chapter). BMT compares all such reported significances to select a single winner based on the maximal value reported (winner-take-all decision). Only this winner is allowed to issue a special token - a behavior-control-token - to the currently active PA_C through the BMT's "motivator" interface.

A behavior-control-token extends ordinary tokens with a mandatory function that reports a PA's favorite neighbor when executed at that PA. Such a favorite neighbor is the one that the token determined to get the robot closer towards the target - or null if the robot is already at the token's best target. When creating such tokens with appropriate evaluation functions (chapter 5.3) a behavior can guide the robot from its current place to a distal target, as described in chapter 5.1.2. Note that such a token can include an "invitation" (chapter 5.1.2.2) to set up a gradient in the network - and once established the token can determine the local favorite neighbor by inspecting the gradient.

The BMT - after fetching such a token from the currently winning behavior - issues it to PA_C for execution. When receiving the token's return, BMT inspects the token's reply for the current PA's

best neighbor, and motivates the currently active PA to send the robot to this particular neighbor - hence the name “motivator”. Additionally, BMT shifts its own control from the current PA to the new PA that will be the PA in control of the robot shortly - and gives the same behavior-token to the new PA in control to determine its favorite neighbor - until the behavior-token reports the current PA to be the behaviors final target.

This gives behaviors a very limited set of functionality: inquiring information from the network, reporting their own urgency to be executed, and generating a behavior-token to find a best neighbor. These few functions are sufficient to keep the whole system operating from initially exploring unknown space to ultimately performing various diverse action, as selected by BMT.

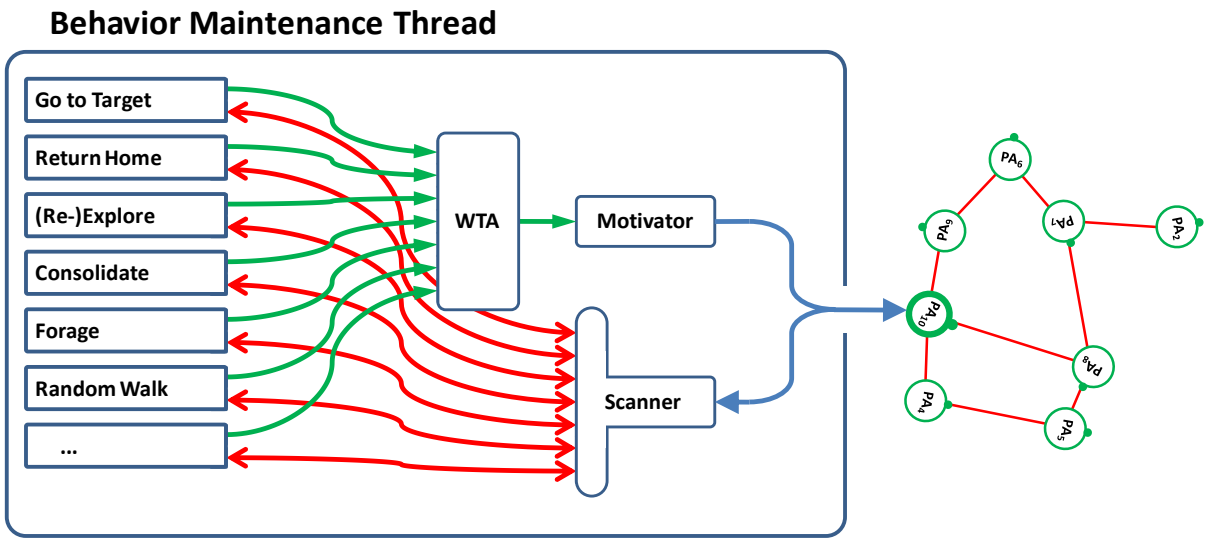


Figure 53: The “Behavior Maintenance Thread” subsumes all individual behavior threads and provides two interfaces from behaviors to the currently active PA in a network of PAs: a scanner allows all behaviors to simultaneously inspect and retrieve distributed knowledge in the network (red), whereas a motivator grants only a single currently winning behavior to issue a behavior-token to PA_C that ultimately controls the robot’s next actions (green).

5.5 Summary and Conclusions

This chapter extended the functionality of place agents. In chapter 1 we introduced basic concepts about handling local places and local actions, but the place agents were not able to take any decision beyond their local place. This is not sufficient for long range spatial navigation.

In this chapter we introduce a messaging mechanism that allows neighboring PAs to communicate with each other, and use this mechanism to propagate a special message in the network: a "token" or alternatively an "invitation". Such a token is a small program - or simply a formula - that allows computing a reward value at a distant PA, based on knowledge maintained by that distant PA. The best reward values travel in the network, possibly overwriting previous best rewards known at intermittent PAs. Upon receiving a return message that reports the best award within the network, the PA that issued the token can take decisions that relate to knowledge maintained somewhere else in the network - without having access to such information or operating on global information about the network's structure itself. "Invitations" are similar to tokens, but set up a gradient in the network towards several sources of requested quantities, e.g. food.

Tokens allow PAs to perform network maintenance tasks, such as re-working connections between neighboring PAs, re-exploring areas or detecting unexpected returns to previously visited places - the so called "closing the loop"-problem, without explicitly representing loops in the system. Each PA only knows a set of direct neighbors, but does not have insight into its neighbors' spatial knowledge. For detecting a loop, an actor in the system needs simultaneous access to at least three place signatures, but such an actor does not exist in our system. Any loop in the environment is only implicitly represented in our system, voiding the computationally expensive task of "relaxing" accumulated errors along such loop.

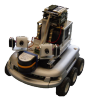
Ultimately, our system is not preliminary aiming to generate a correct graphical representation of a spatial environment; we are not interested in obtaining a pretty picture, but instead we are looking for a behaving system that allows navigation with respect to distant goals. We introduced several typical behaviors that "perform" based on the network of distributed PAs, such as finding a distant target, homing, foraging, etc. All such behaviors only communicate with the single PA that is currently controlling the robot. A behavior issues tokens or invitations to collect information from the existing network of PAs about remote locations that might fulfill the robot's requests. Again, all of such behaviors only operate on local knowledge and only interact with a single PA at any time.

6 Results and Performance

The previous chapters presented the principles upon which our system operates: distributed agents that maintain local representations of behaviorally relevant places in space. We have presented how these agents maintain their local spatial and behavioral knowledge, how they explore space and build up a network that represents an initially unknown environment, and how they autonomously perform network maintenance tasks and ultimately generate globally consistent behavior. All these discussions happened on simple artificial data that illustrates the principles well.

In this chapter we present and discuss several examples of our system operating in larger and more complex environments. We initially show data from multiple explorations of a large office environment (60x23m) and analyze the resulting networks' properties. We use the results of these explorations to discuss differences between spatial knowledge represented in a network of PAs and the best available true geometric spatial knowledge of the environment explored. We conclude with a detailed look at how our system handles - or ignores - on cycles in the environment.

Throughout this chapter we have two sources of spatial information: A real robot (introduced in chapter 3.2) and a simulator of this particular robot (chapter 3.3). We collect and present data from both these sources, but for each result clearly indicate its source with a small picture of a real or a simulated robot respectively:

 denotes data obtained from the real robot

 denotes data obtained from a simulated robot in a virtual environment

Discussing data obtained from a simulated robot in this chapter has two main advantages over data obtained exclusively from a real robot:

- We can track and compare the robot's true (simulated) position against the network's estimate of the robot's position easily, and hence compute absolute errors. The real robot in contrast operates somewhere in the institute, and we will only notice errors when it gets lost - and still cannot quantify such errors. We have no means of obtaining the robot's absolute Cartesian position in our institute, expect for a small test setup in which we can track the real robot using an overhead ceiling camera. This setup only covers 5x4meters floor space, and therefore is only useful for repeating particular behavioral experiments (ref chapter 7.5); we cannot cover the whole institute with a robot tracking system, which would be a huge research project for itself.
- The simulator reports data from a completely known hand-designed underlying structure: we know exactly what environment generated a robot's sensory perceptions; hence we can compare the resulting network against the true spatial layout. In contrast, although we have obtained blueprints of our institute, we do not know the spatial arrangement of objects inside rooms. The blueprints only contain positions of walls and doors, making it difficult to compare a resulting network that the real robot generated against ground truth data. We

have measured and sketched furniture in the four largest rooms within our institute; but people regularly reposition objects, such that the effort did not pay off. You will notice below that in all presented Cartesian maps spatial precision is significantly higher in the four large rooms compared to only a rough sketch in all other areas of our institute. We still use these spatial maps created some time back for the simulator, but also to carefully compare real-world data against.

6.1 Establishing a Distributed Spatial Representation

6.1.1 Temporal Development of Network of PAs

Figure 54 shows time-stamped snapshots of an exploration run in the loopy version of toy world. Each event that contributes to incrementally building the final network is captured in a new frame, such that we can follow in detail how the network is established. The frame at the bottom right shows a top down view of the simulated environment; the robot starts in the top-left corner as indicated.

Going through the frames in order one can see how starting from a nucleus PA (frame 1) the whole network develops. Each new PA inspects its internal representation of space, and upon detecting open space creates children (blast PA, shown as pink circles, refer to chapter 4.1.2) to later explore in those direction. Upon a child receiving robot control it takes the robot for exploration and finally decides for a particular place to represent itself. Once a child's exploration finished, the process repeats: it possibly creates its own children for further exploration (frames 4, 6, 9, ...), or returns the robot to its parent for further exploration elsewhere if itself does not see unexplored space (7, 12, 16).

In frame 12 a child sets off to explore leftwards, taking the upper long aisle that ultimately leads back to a junction already represented by another PA in the system. Frame 13 depicts the brief situation in which the new child has found the place it wants to represent, but has not yet detected that another PA already represents the same place. Note the misalignment of these two PAs: all red vectors are properly relaxed, reflecting the neighborhood knowledge of all involved PAs properly; still the two PAs that represent the same place (large green circles in top left corner; not the smaller nucleus PA!) are misaligned due to sensor errors that accumulated along the way.

Frame 14 shows the network after old and new representation of the same place merged into a single PA representing that place, but also all neighborhood relations as known by two original PAs. Note that after fusion of these PAs the network is significantly skewed: red vectors contained in the cycle do not point towards their respective partner anymore. The collection of PAs represents globally inconsistent knowledge without any of the PAs being aware of that. Only our display program as of now has difficulties showing all spatial relations amongst the PAs correctly. We will discuss the skew in much detail later in chapter 6.2.4.

After closing the loop, the new PA inspects its combined spatial knowledge to search for open space and creates a new blast PA which continues exploration. Finally all blast PAs finished exploring, such that we see a fully developed spatial representation in frame 17 after a total of 12 minutes and 40 seconds.

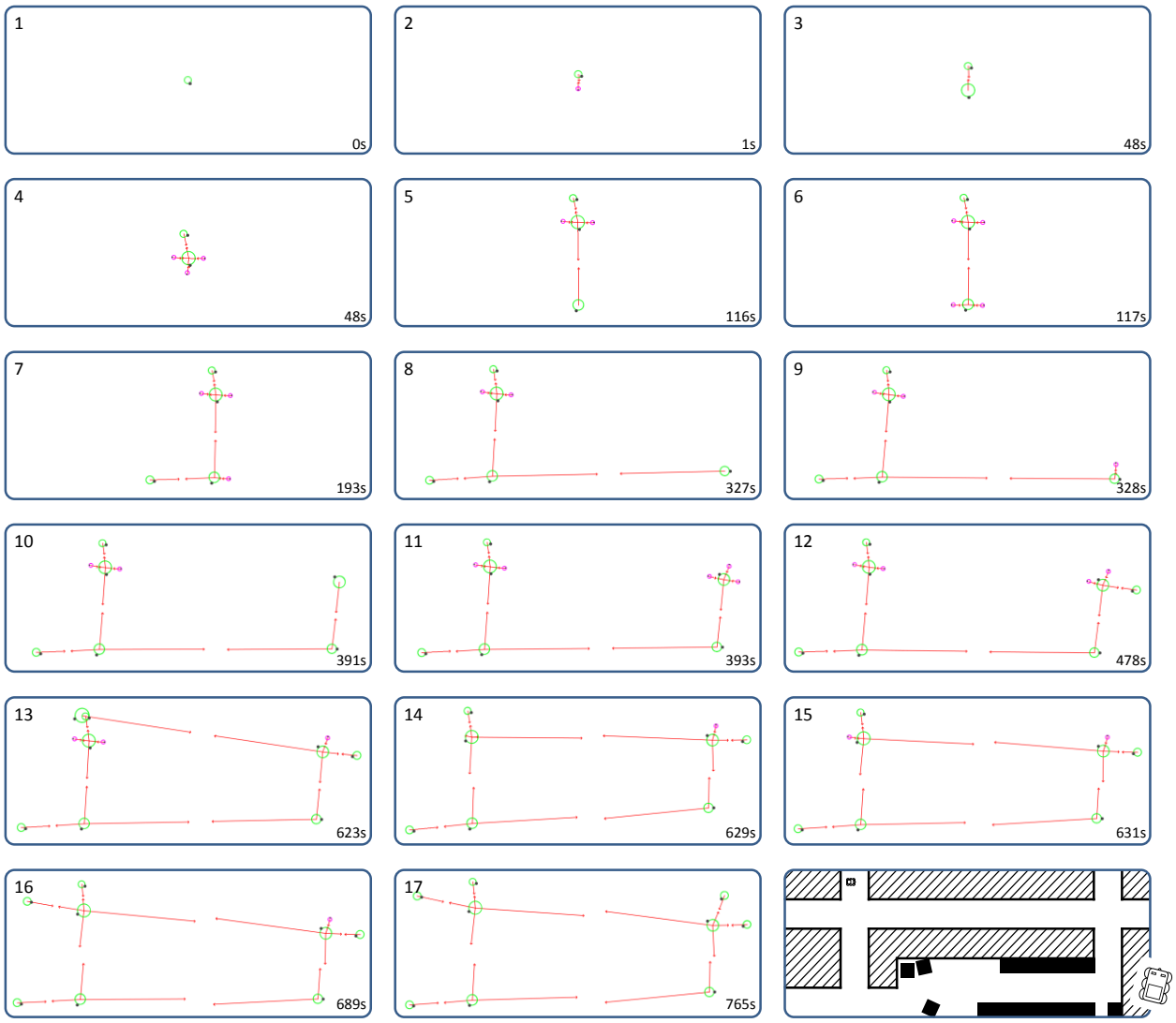


Figure 54: Incremental buildup of a network of places representing a simulated environment with loops of size 25x10m. Note loop-closing in frame 14 which shows skew in the connections between PAs afterwards. Red vectors show 45% length of a PA's recorded distance to its neighbor.

Frame 17 shows the network of PAs after exploration that represents the loopy toy world. We can clearly see that the network of PAs has captured the topology of the true environment; but also that several angles and distances reported by individual PAs are contaminated with error and thus do not show the clear 90° structures of toy-world. An arrangement of PAs in the frame such that all individual vectors point to their respective neighbor is impossible due to these recorded errors. In practice, such local errors are unproblematic as the robot will locally find the correct aisle to turn in at every given place (a detailed discussion follows in chapter 6.2.2). The displayed angles and distances report the PA's knowledge after only one traversal along the path. Figure 55, top left panel, shows the same network after a brief consolidation period in which the robot visited every path in the network for a second time. This network display already shows improved distance and angle representations, clearly visible by the long vectors pointing closer to each other, but also by the terminal nodes arranged in a more right-angled manner. The top right panel in Figure 55 - the same network after a total of 5 consolidation runs - shows hardly any misalignment anymore.

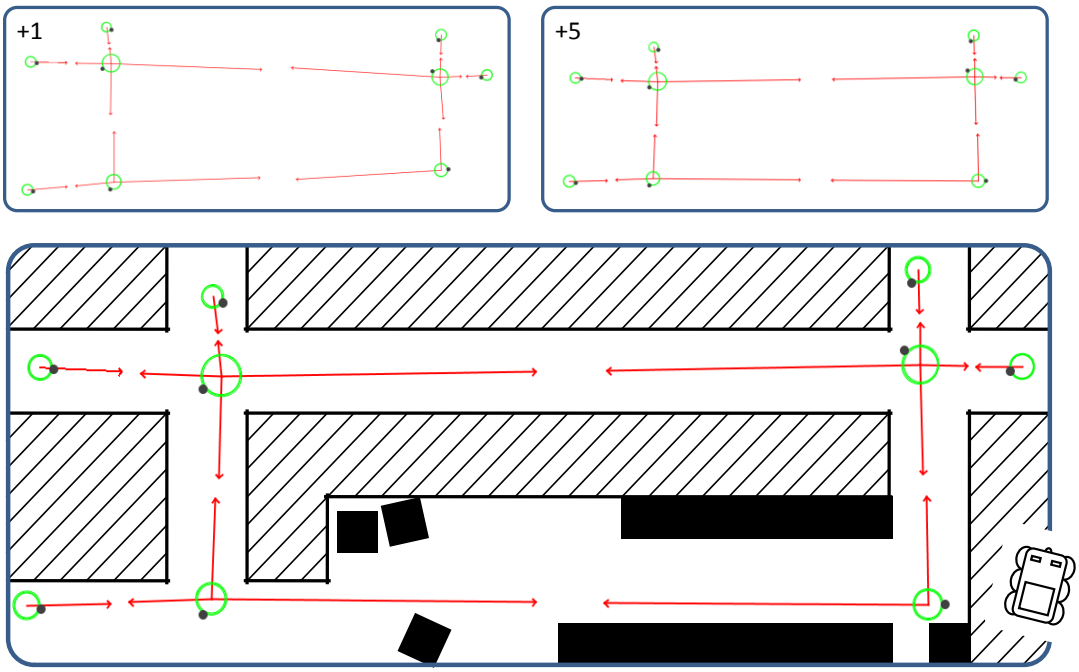


Figure 55: A network of place agents representing toy world. Top-left panel after one consolidation run, top-right panel after 5 consolidation runs in which the robot traversed every path in the network. The bottom panel shows the final network superimposed over the true spatial layout.

The bottom panel in Figure 55 shows the resulting network after consolidation superimposed over the true environment as presented from the simulator. We can clearly see how the network's topology reflects the spatial structure, but also see that the network of PAs seems skewed with respect to the environment: several PAs do not appear at the optimal place, but slightly misaligned. We will discuss such a misalignment in detail in chapter 6.2.4).

Figure 56 shows the same network, but allows us to inspect the spatial knowledge of two participating PAs: The large green circles show all information captured in the PA's log-polar based binned data structures, as stored with respect to the PAs own coordinate frame. The large dot on the outside ring of a PA's image in the network showing the PA's zero-degree orientation; the same zero degree orientation is shown on top of the large green circles.

Various events recorded in the spatial structure are shown in different colors according to the following color-key:

- Black dots denote occupied bin; the darker the more certain a bin is obstructed
- Green dots (with labels) show positions of visual landmarks (only center of gravity shown)
- Red bard show triggers of the robot's infra-red distance sensors
- The blue line originating in the center show the recorded compass direction towards north
- The purple dots indicate the PA's known direct neighbors
- Blue marks close to the center indicate directions in which this PA has created a child (blast PA) to further explore space[4].

[4] The left PA has initially created children in all three open directions (all except the one its parent was in). But the snapshot on display is from after fusion with another PA, such that it only reflects those directions in which it has creates children after fusion.

Please refer to chapter 3.2.7 for more details about the individual sensors.

Note that the place signature does not represent relative sensors, such as a gyroscope or the target sensor, as these have no meaning for a stationary signature: the reported value will be different whenever approaching the place (refer to chapter 2.2.5 for details).

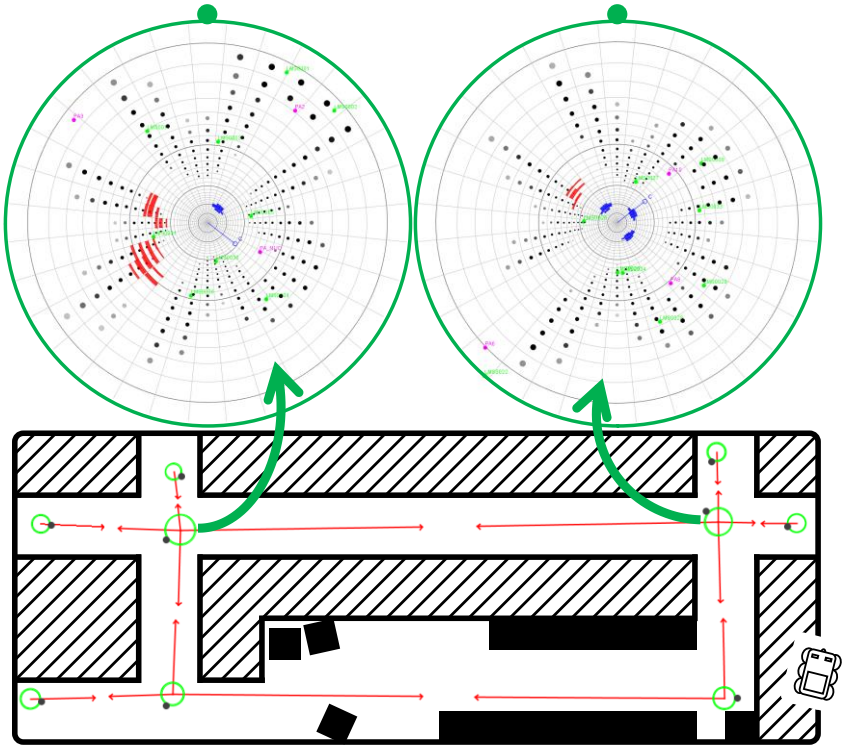


Figure 56: Inspecting the spatial knowledge of two example PAs from the network. Both PAs represent a four-way junction; hence their spatial signature is to some extent similar, but rotated to reflect information with respect to each PA's own coordinate frame.

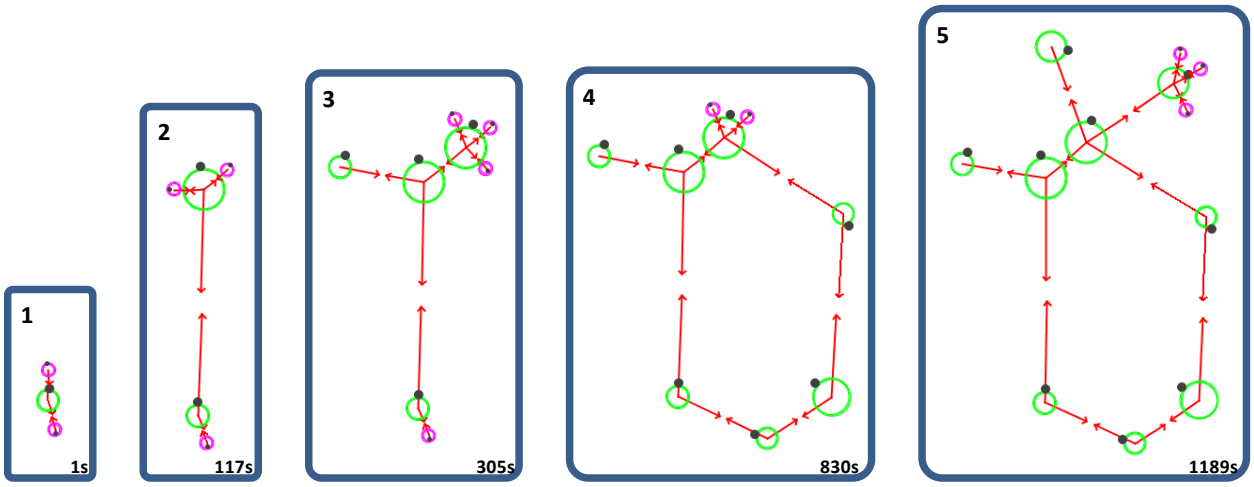
When inspecting such signatures, the best hint we get to orient them properly for a comparison is the compass: in a perfect environment the blue compass direction has to be identical for multiple such place signatures; in our real world scenarios (both robot and simulator) the compass report is sufficiently consistent to get used as a hint. Note that the system doesn't know about this trick; it only helps us observers to match the displays of multiple signatures.

These two PAs are very similar: they both represent a 4-arm crossing with two "closed" and two "open" arms. In fact by the orientation in which they happen to be on display, both signatures show their closed arms towards the right (up and bottom) and their opened arms towards the left (up and bottom). But already the compass data provides a first insight that the signatures need to get rotated with respect to each other - after which the open and close arms do no longer coincide. Additionally, the recorded landmarks show that the two PAs do not represent the same place. When the system compares two such signatures, all sensor data is equally weighted - such that the dominance we see due to the black spots denoting obstructed space does not exist (refer to chapter 2.6).

The toy world is a good example to show principles, but too simple to illustrate real-world performance. Here follows another example where a real robot explores an office space of

8x15meters that has a slightly more complex structure compared to toy world. In the following panel (Figure 57) we only show snapshots of the network's current state whenever structurally relevant events occur during exploration, such as closing loops or starting to explore a new direction. Figure 58 show the resulting network overlaid over a floor-plan view of the office.

Note loop closure of a truly existing loop in panel 4. In contrast, in panels 7-9 the robot explores a corner of the institute that - ideally - should get represented as a t-junction. However, once in the corner (panel 8), this PA sees two open aisles in its environment: one to the upper left and one to the lower left. The upper one is covered by its parent, the lower one remains for exploration. The new PA creates a child and sends this child for exploration, which ends up at a place already represented. The two PAs merge, and hence create a small detour into the corner, that we see as "loop" in the network display (panel 9) - although the loop does not circulate an object. The same situation occurs again in the transition from panel 10 - panel 11: The robot returns to a place already represented along a different path - although not around an obstacle - thereby creating a loop. Those loops that do not enclose an obstacle - easily seen in Figure 58 - might seem problematic when representing space; but in fact they only span open space and provide alternative paths to reach places along the outside of that open space - instead of having a single PA representing the center of such a place.



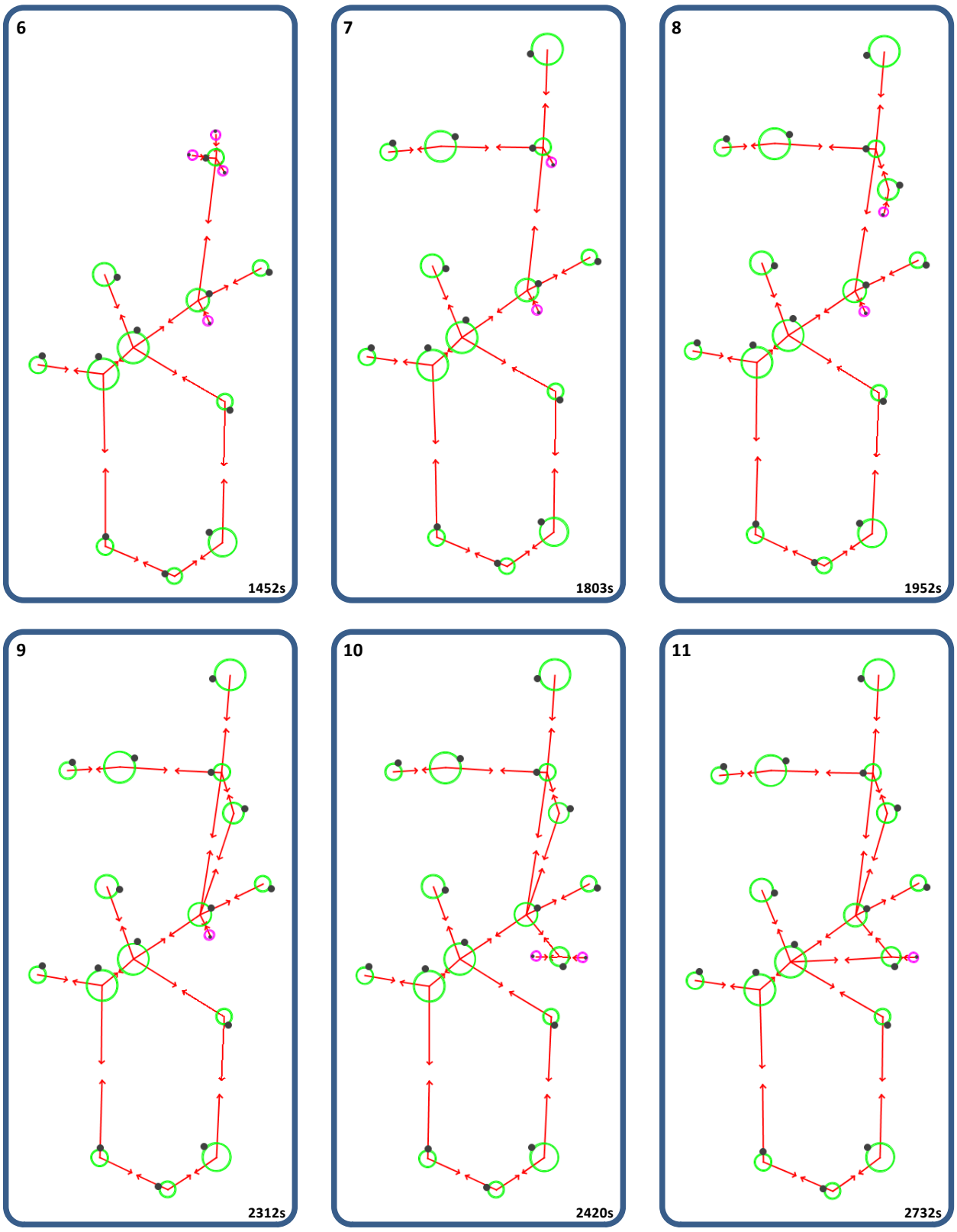


Figure 57: Incremental buildup of a network of places representing a real office environment of size 8x15m.

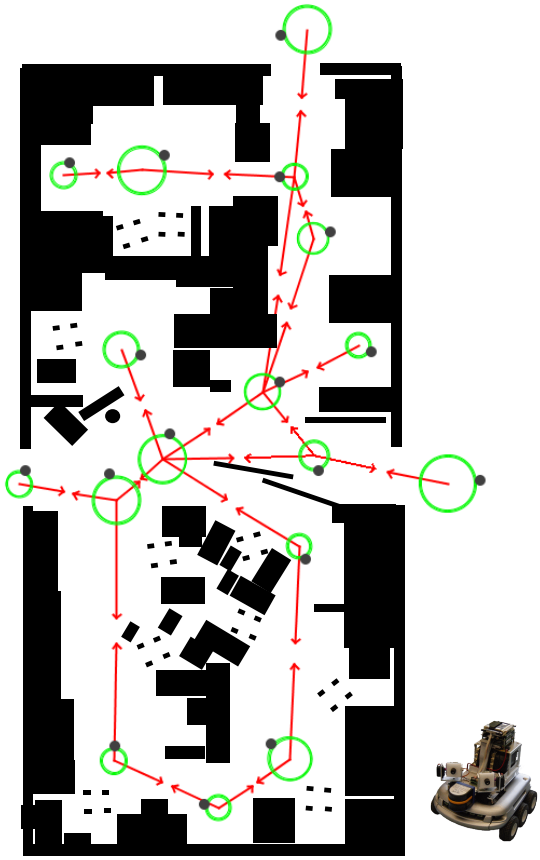


Figure 58: The resulting network superimposed over our best representation of the true office space. Behaviorally relevant places (those where detours are possible or even required) are covered by a place agent.

Figure 58 shows the resulting network superimposed over the best available representation of true space. Note that again - as in the toy world example before - the PA's assumes positions do not coincide with the obvious best positions for PAs when observed in a picture like this. The network seems distorted with respect to the underlying environment. We will address this in detail in chapter 6.2.4.

The following Figure 59 shows an example of a network resulting from the robot exploring our whole institute of 60x23meters. Note that the network shown in Figure 58 is contained in the larger network, as illustrated by the shaded box. The plan view below the network reflects our best knowledge about the institute's true layout. The four large rooms at the bottom center of the map (student labs) show much higher spatial details about the environment because we went through the effort of drawing the true spatial arrangement of these rooms in a CAD program. All other rooms, in contrast, only coarsely show the true spatial arrangement. Those rooms crossed with diagonal lines are inaccessible for the robot, either for security reasons (the current robot hardware cannot detect a stairway - and might simply fall downstairs) or because they are locked for their own protection.

We will use the same Cartesian map for simulated experiments, but extend the accessible space slightly (virtually open all locked rooms and remove staircases, see Figure 68 for a picture). Using a map in the simulator that resembles the real environment as closely as possible allows comparing results from simulation with results from real experiments. For the upcoming analysis we have performed a total of 15 explorations of this large environment: 10 times in the simulator and 5 times

with the real robot. All resulting networks of such explorations are shown in Appendix A, which for now gives a good impression of what those networks of independent place agents look like.

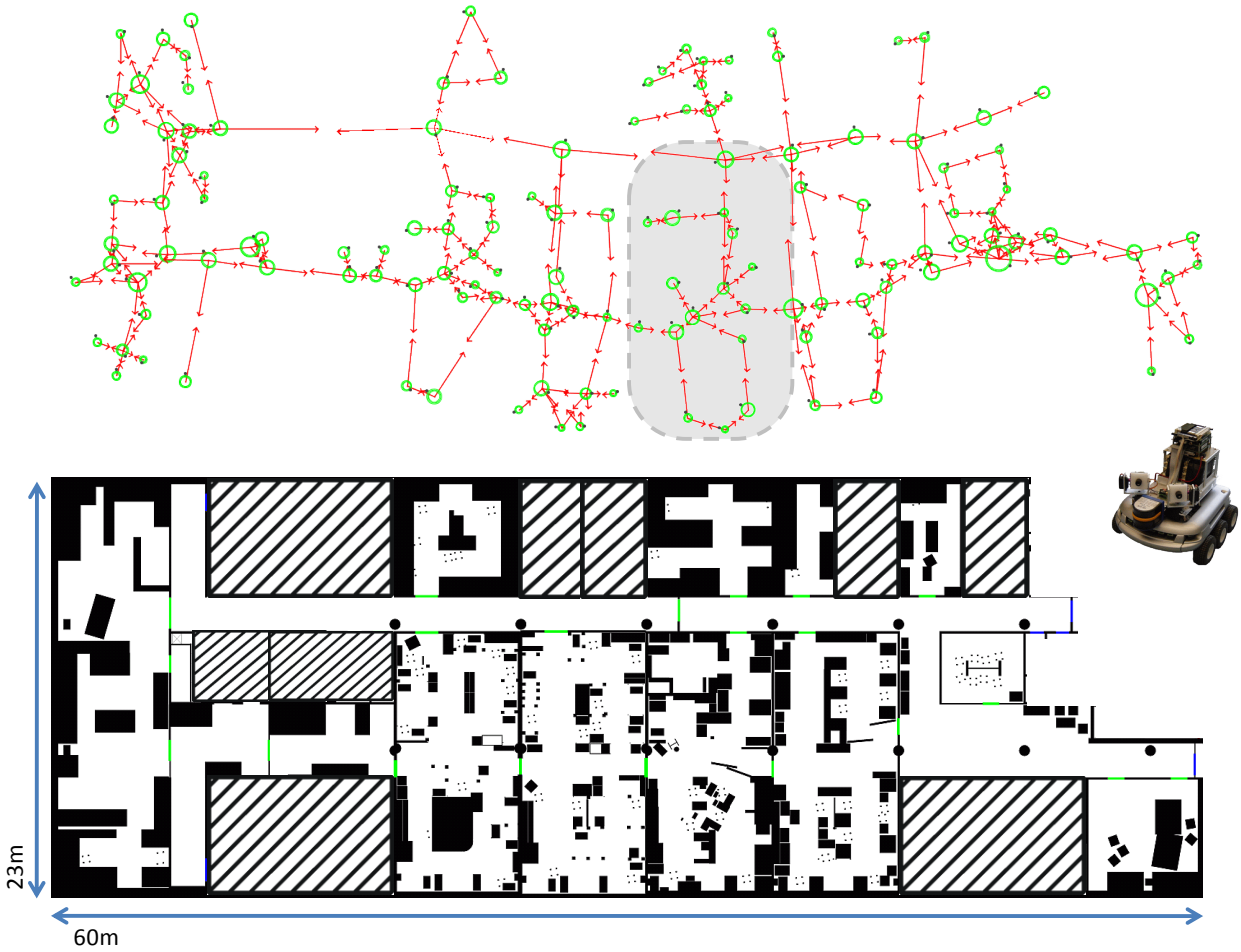


Figure 59: top: A network representation after exploration of the whole institute. The previous network from Figure 58 is contained in this network, as indicated by the gray shadow. Bottom: a floor plan showing our best knowledge about the true spatial structure in our institute.

6.1.2 Statistical Analysis of Networks

In this chapter we want to inspect the results of explorations - both in simulation and on the real robot - to get a better understanding of the structure of networks of PAs that represent the underlying space. All of the following graphs are generated from the same 15 explorations of our institute, of which 5 were performed by a real robot and another 10 by a simulated robot. Each figure in this chapter shows data derived from all 15 explorations together in a large display, but has two small additions in which the data from real world experiments and simulations are separated. We cannot detect a significant difference between real-world data and simulation, except in Figure 62 where we will address this difference explicitly.

All 15 explorations together create a total of 2172 place agents:

Real Robot:	**Σ = 669** (144, 132, 129, 132, 132), **avg. = 133.8**
Simulated Robot:	**Σ = 1503** (173, 170, 163, 138, 126, 130, 132, 114, 191, 166), **avg. = 150.3**

The on-average higher numbers for the simulated robot can be explained by looking at the respective environments that the two robots explored: the simulated robot was allowed to enter a few extra rooms, and thus often required more PAs to represent this space. The low numbers of PAs for the simulator in runs 4-8 have different reasons, which become obvious when inspecting the resulting networks (available in Appendix A): those networks either found a very compact representation for open space (only a single PA represents open space instead of multiple PAs) - and/or simply ignored parts of the environment to explore, i.e. the simulated exploration missed an opening into a room and thus has not mapped that room. Both these reasons are particularly true for the 8th simulated exploration run, which ended with only 114 PAs. This statistical analysis only inspects the networks as soon as the network finished exploration; it does not include a later consolidation phase that will be explained in chapter 6.1.4, when a robot might discover areas that it initially missed.

The following Figure 60 shows a histogram of the number of neighbors a single PA knows. We can see that typically PAs have a small number of neighbors, with exceptions ranging up to 9 neighbors for a single PA. Those PAs with only one neighbor represent dead-ends in space; those with exactly 2 neighbors a place where the robot needs to change direction of motion. Those PAs with three or more neighbors denote places where aisles intersect, such that a traveling robot can chose between one of many possible paths to continue. In those rare occasions where a PA has 7 or more neighbors, that PA represents the center of a place with large open space, where the outside boarder of that open space is also covered by several PAs. All trajectories from the center to the outside PAs are possible; hence the center PA might end up having a large number of neighbors. Examples are easily visible in Appendix A towards the center left and center right of many displayed networks.

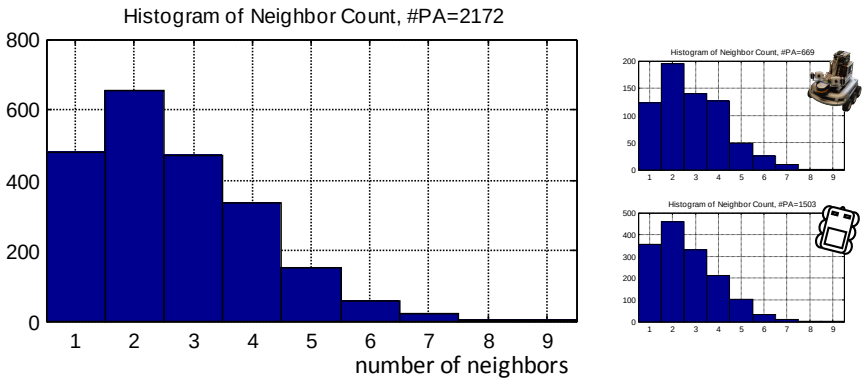


Figure 60: Histogram showing the distribution of the PA's neighbor count.

Figure 61 shows distance and angle distributions between PAs in the 15 networks. The left diagrams show histograms of inter-PA distances: we find distances ranging from about 0.5 meters up to well beyond 8 meters, with a clear preference for distances in the range between one and three meters. This reflects our design principle - only representing relevant space - well: the system does not create new places in a regular grid, but each new PA continues exploration until it detects a new relevant place in the given environment; often very quickly within a few meters, but occasionally only after a long exploration. Again, this distribution is easily visible in the networks shown in Appendix A, where we typically see long distances between those PAs that represent places in the

hallways (horizontal the top in the images), but mainly short distances where PAs reflect highly structured environments as typically found in tight office spaces.

On the right side of Figure 61 a set of histograms show the distribution of all PAs' inter-neighbor angles. Note that only those PAs contribute that have two or more direct neighbors; otherwise a PA with only a single neighbor would contribute only to 360°. Again, we can see very clear peaks in the distribution of angles, reflecting human designed environments with the most dominant peak at 90° and smaller peaks at 180° and 270°. The relatively large number of angles close to zero degree denotes PAs that have multiple neighbors in almost the same direction. Such a situation often happens when three PAs represent a narrow passage that is yet sufficiently wide open such that the span a triangle instead of a direct aligned connection. A typical situation is shown in Figure 59 involving the PAs at the top-right end of the gray shaded inset.

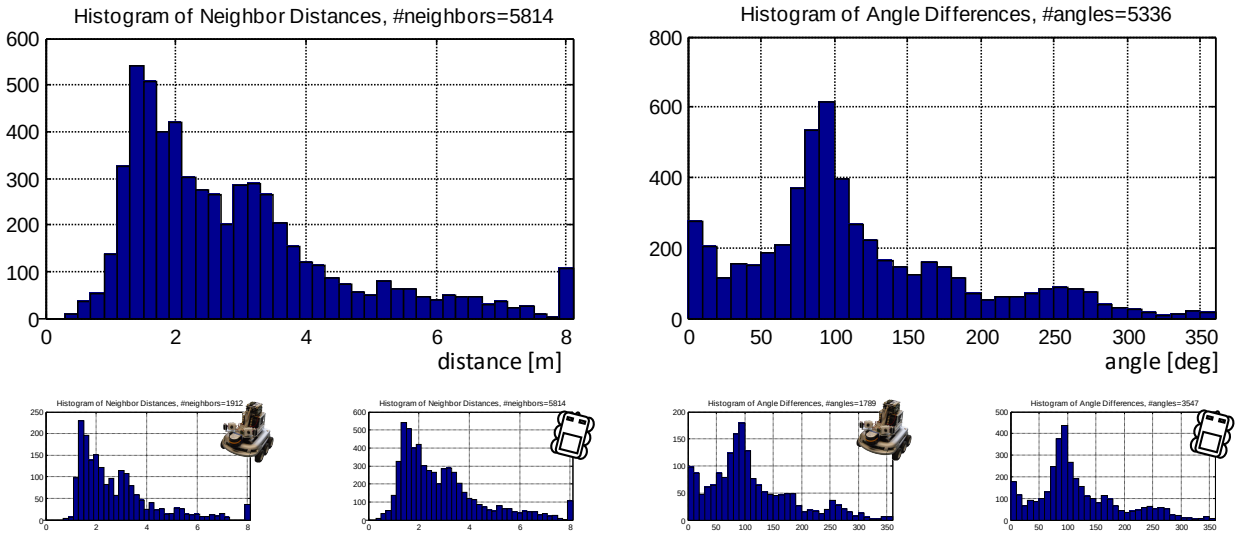


Figure 61: Left: Histogram showing distances between neighboring PAs as recorded in the PAs local spatial environment. Right: Histogram showing inter-neighbor-angles from all PAs with more than one neighbor.

Finally, Figure 62, left, shows the total area explored for the number of PAs involved in the network during exploration. We compute the explored area as the convex hull that contains all existing PAs at the position the network-GUI determined them to be, applying a position relaxation algorithm (refer to chapter 4.4.1). We can clearly see that the more PAs contribute their local spatial knowledge the larger the area, but clearly saturating well below 1400m^2, limited because of the size of our institute of 60x23m = 1380m^2. The area covered initially grows more quickly, but later reduces growth rate, because initial PAs are likely to increase the area of the convex hull, whereas PAs added later to the network are more likely to be already contained inside the convex hull. The variance (blue shadow denotes 1 standard deviation) decreases towards the end, because several networks only employ about 130 PAs, such that they do not contribute to the error margin later anymore.

The right panels in Figure 62 show the average area represented by a single PA, i.e. the total area represented by the network divided by the number of PAs in the network. We can see that ultimately each PA - on average - accounts for roughly 10m^2 of environment. However, this number needs cautious interpretation: large areas inside the convex hull are not represented at all by PAs, such as space in-between two distant neighboring PAs, but also space corresponding to rooms that

remained closed during exploration. Such areas contribute to the convex hull, but not to the number of PAs; hence the true average space a PA represents is below $10m^2$.

Looking at the graph we see that initially the area covered by a PA grows larger than $10m^2$, because during initial exploration the robot builds up a coarse representation that spans a relatively large area with few PAs only; but as soon as it returns to consolidate information or to continue exploration in a different direction, the area per PA decreases, finally approaching a value slightly below $10m^2$. Note that the standard variance (blue shaded region) is large, showing that the (random) order in which the robot explores an environment has significant impact on the area currently represented (see following chapter).

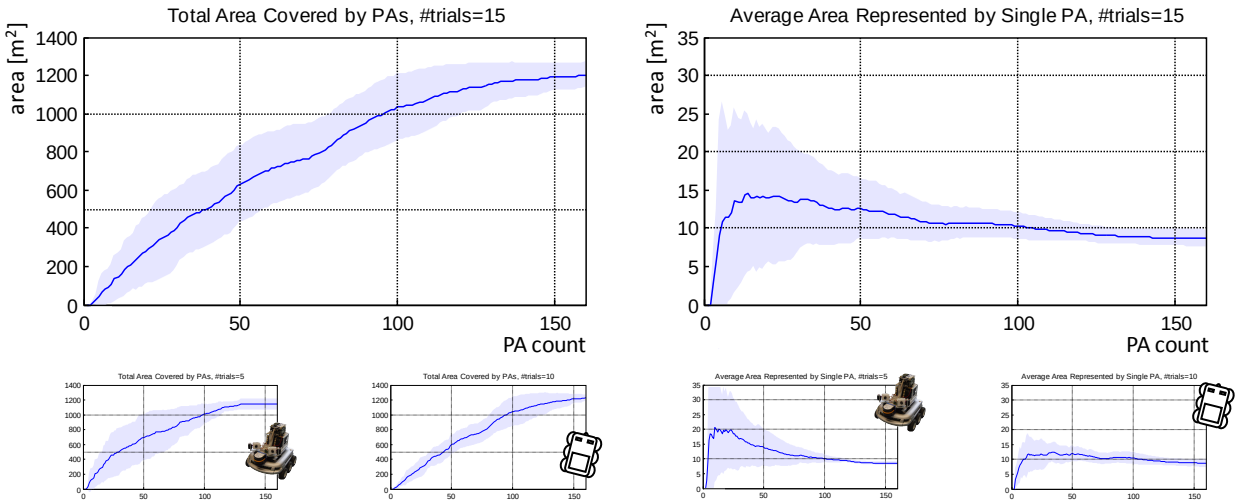


Figure 62: left: The size of a networks' convex hull (representing the area covered by network of PAs) grows with the number of PAs contained in the network. Right: Average area covered by a single PA.

Figure 62 shows the only aspect in the statistical analysis in which we noticed that the real robot's behavior diverges from that of the simulated robot: looking at the four small panels reveals that the real robot covers larger areas initially with a significantly increased standard deviation. Ultimately, the graphs of real vs. simulated robot approach the same limiting values. The only explanation we have for that is a different starting position for simulated and real robot: the real robot always started in one of the student rooms, whereas the simulated robot always started in the middle of the long aisle at the top (see Figure 63). Starting in the aisle might have caused the simulated robot to acquire an initial set of PAs that span a small convex hull (mainly one-dimensional area); whereas starting in a complex structured environment might have increased the area from the beginning. Note that both, simulation and real robot, quickly agree on a common average total area and area per PA.

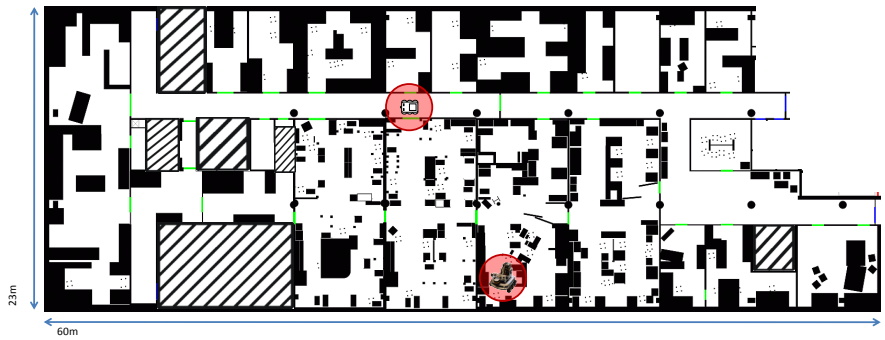


Figure 63: Different starting positions for the simulated robot (top red circle) and real robot (bottom red circle) are most likely the reason for initial differences regarding the explored area as shown in the small panels in Figure 62.

6.1.3 Exploration Strategies

We have initially implemented a strategy in which the robot continuously explores its environment and creates maps as large as possible in as short time. However, we realized that ultimately such maps suffer from large accumulated errors in angles and distances, making it difficult for PAs in these maps to detecting cycles and do not close loops properly. Unclosed loops, however, lead to re-exploration of large areas in an environment (refer to chapter 6.3) before the loop is ultimately detected, such that the overall time required for exploration again increases. If the robot were to explore a maze without loops, really such error accumulation does not matter for performance (to be discussed in chapter 6.2), but in general we cannot make this assumption.

We empirically found that the most successful map building happens when the behaviors "explore" and "consolidate" (which revisits existing places to verify distances and angles, chapter 5.3.4) run at roughly equal time fractions. Alternating exploration with any other behavior such as consolidate breaks the direction continuity of the current exploration: when exploring e.g. a long aisle the robot typically picks the open space to explore next that continues in the direction of the aisle, because it already faces this direction; whereas after returning to a previous place further exploration most likely happens in a different direction. Changing the direction of exploration regularly increases chances of traveling along small loops, which are easier to close compared to large loops (chapter 6.3). We also tried forcing the robot to chose the left-most (alternatively right-most) open space with respects to a PA's parent during exploration, which leads to closing the smallest loops initially. However, this resulted in a large number of turns that again accumulate error and take time.

Alternatively, we tried an exploration strategy in which the robot always returned to a blast PA's parent after establishing a new place, thereby consolidating the last detected link. This generally produced good results but slowed down exploration significantly, as often the robot went from parent to blast, back to parent, and then back to blast and thus traversed a single path three times instead of once. Occasionally, even worse, the robot started exploring an aisle in two opposite directions and extended knowledge in those two directions in an alternating fashion, traveling all the way back and forth.

6.1.4 Completeness of Networks

Our system incrementally gains knowledge about an initially unknown environment by exploring all areas of unknown space that it detects within its internal representation of the currently explored environment. For any size-limited environment, at some point in time the system will find no more

unexplored spaces that are internally marked for exploration, causing the exploration behavior to stop: the network has no more interest in exploring, and even when triggered externally the behavior has no work and returns immediately. Does that mean that the whole environment is represented in the network?

Figure 64 shows a snapshot of a network taken at the time the exploration behavior first declared it finished exploration. We can see that the robot "missed" to capture several places and connections between places (marked by red circles). Typically, the open space leading into such a place is too small such that the PAs on both ends of the particular open space did not expect the robot to fit in. In this example the robot missed three individual paths, but also a full place (left-most red circle); occasionally the robot misses the entrance into complete rooms - and hence does not represent the room in the network (for an example see Figure 68: top-center and top-right rooms are missing in the exploration).

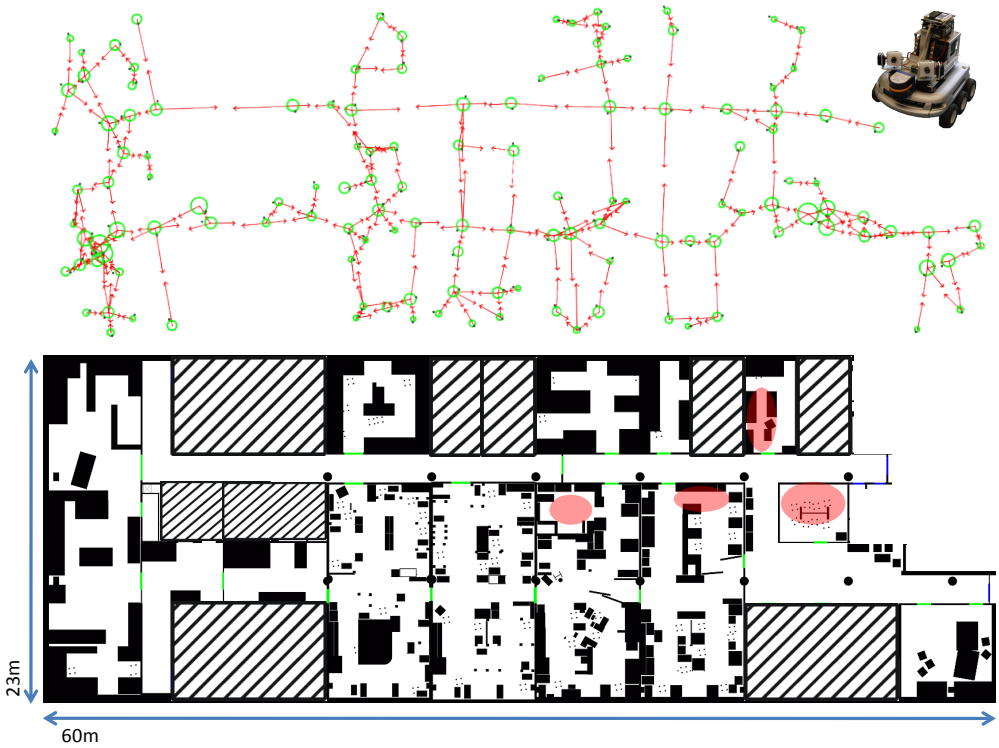


Figure 64: A network of PAs missed several aspects of the environment during initial exploration as marked by red circles. In later re-exploration and consolidation phases the network continuously compares its own knowledge with current perceived sensor readings from the environment, which ultimately allows the network to detect those spaces and to represent them.

Such missed spaces do not concern us, as the system does not have a clear boundary between exploration and application phases for the network. The robot continuously provides current sensor perceptions, such that all PAs upon being revisited revise their spatial knowledge and possibly later detect open space of sufficient width for the robot to pass through. In such a case a PA will later create a new child for exploration, and the robot will ultimately explore those spaces it missed initially.

If the robot spent more time driving slowly during initial exploration and on capturing a more precise initial place signature for new PAs, the resulting network will be more complete from the beginning.

The system faces a similar tradeoff between time and accuracy required to represent a possibly large environment. The situation closely resembles how humans report exploring space: we might often walk along a corridor ignoring possible places to the left and right - until we happen to find an open door and start exploring...

6.2 Spatial Correctness of Networks

6.2.1 Inaccuracies in a PA's Local Spatial Information

Every PA maintains a place signature that in addition to the PA's local spatial environment contains information about its direct neighbors and their positions in the PA's reference frame. A single PA has no access to information that allows it to verify positions of its neighbors in a global context; no PA is aware of cycles in the environment nor has any PA access to absolute positions. Therefore, all PAs must assume that their knowledge about their neighbors reflects the true environment.

Having the option to inspect all PAs at once we as external observers can investigate the correctness of a single PA's knowledge about its neighbors. The network PA-GUI (refer to chapter 4.4) displays all PAs in a globally consistent arrangement, which reflects all PAs' distributed information about their respective neighbors as accurately as possible. Comparing distances and angles on this display against distances and angles reported by each individual PA provides an error measure between recording and the best global arrangement. We have evaluated all PAs from the data set introduced in chapter 6.1.2 that consists of 5 explorations by the real robot and another 10 simulated explorations. The histograms in Figure 65 show that both, distances and angles, have an error distribution that peaks sharply around zero, indicating that all PAs record distances and angles to their respective neighbors well. The few outliers in distance errors at up to ±1m can get traced back to those PAs that report neighbors in large distances - such that a distance error of about 1m corresponds to less than 15% of the neighbor's distance and does not cause problems when sending the robot to the respective neighboring PA.

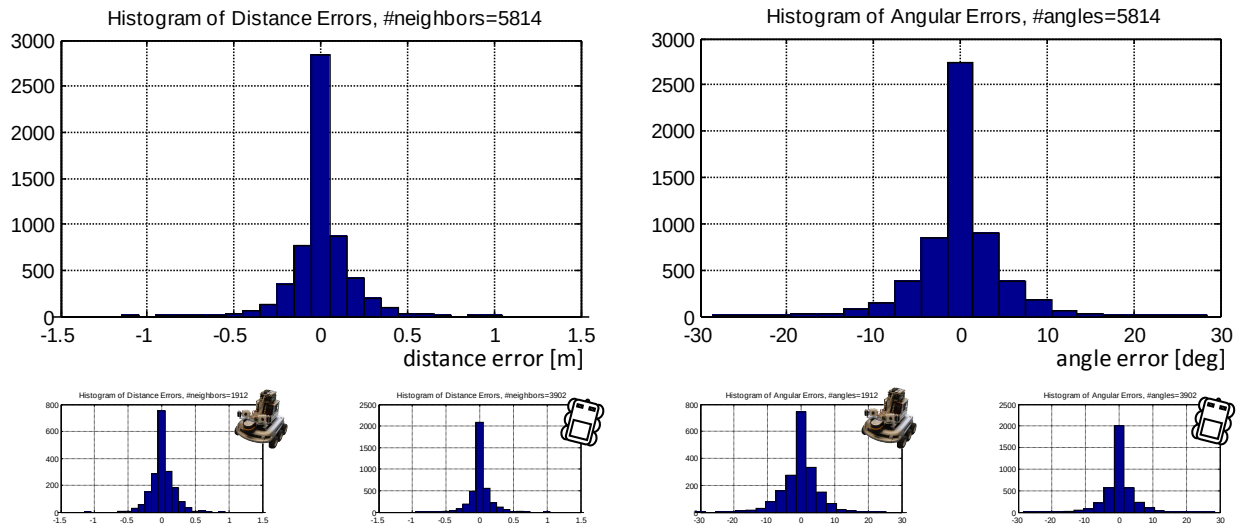


Figure 65: Histogram of differences in distances and angles reported by PAs compared to the best network arrangement the GUI found to satisfy all PAs (refer to chapter 4.4.1). Both distributions show sharp peaks at small errors, indicating that the distributed PAs maintain a globally consistent representation of space. Note that the real robot shows a slightly wider spread of errors in distances and angles compared to the simulated robot, but the difference is not significant.

Figure 65 only compares single PA's estimates against the best global arrangement of all combined knowledge - which does not necessarily reflect the geometry of the true underlying environment. For the 10 simulated explorations the simulator in addition provides true Cartesian coordinates of the robot when recording place agents, such that we can compare the PA's acquired neighborhood information with true distances and angles for those simulated runs (Figure 66).

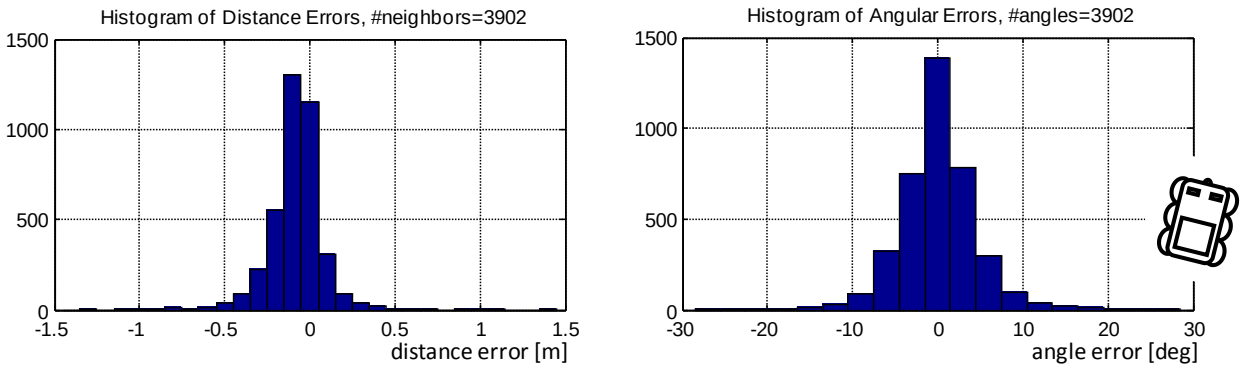


Figure 66: Histogram of differences in distances and angles reported by PAs compared to all PAs' true positions as reported by the simulator during exploration. Note that angles are well balanced around zero, whereas the peak in the distance graph is shifted towards shorter distances.

Note the wider spread of both these histograms compared to their respective small insets in Figure 65: distances and angles seem less precise. The histogram contains the identical angle and distance reports from the same set of place agents - only the baseline is different. In Figure 65 we compared each single PA's knowledge against the best global arrangement of all the combined knowledge - here we compare against ground truth. The global arrangement does not reflect the underlying space exactly but already is a compromise based on all provided data; hence the error is smaller in Figure 65. For the real robot experiments we do not have correct positions in space, so we need to check against the best arrangement. Both figures show that the difference between best arrangement and true data - however, is not drastic.

More dominantly, the peak of the distance distribution is clearly shifted towards negative values, showing that - on average - a PA's recorded distance to any of its neighbors is smaller than the true spatial distance reported by the simulator. The reason for this is because all PAs continually update their estimates about their neighbors' distances after establishing the initial network. Each such an update happens when the robot travels between two neighboring PAs. On such a travel, the robot likely stops a little early, as soon as its currently reported place signature and the signature from the target PA show sufficient similarity. We have arbitrarily decided on a distance threshold of ½ times the robot's body length, here 16cm. Alls PAs therefore receive updated distance information that reports slightly shorter distances compared to true distances on every visit of the robot after the initial setup. In practice, the reported distances are shorter by about ⅛ times the robot's body length; which is small distance compared to the average distance between places, so that we decided to ignore the constant offset.

6.2.2 The Path from a PA to one of its Neighbors

A PA sending the robot to one of its neighbors initially orients the robot towards the receiving neighbor, by comparing the robot's current view against its own place signature (refer to chapter

4.3.2). While directing the robot to rotate, the PA has to rely on its internal knowledge about the position of its neighbors, which might be slightly inaccurate as shown in the previous chapter. Furthermore, the distance measure required to aligning the robot's current view and the PA's spatial signature (refer to chapter 2.6) typically shows a variance of about 10 degrees. With these two possible sources errors, the robot might well face a substantial direction offset when starting to approach the neighboring PA.

Looking at Figure 67 which shows recorded trajectories in an earlier version of a simulated student room, we see such offsets when the robot travels in space guided by the network of PAs. We can clearly detect "nodes" along the recorded green trajectory, i.e. places on which all the trajectories converge. These places are the ones represented by an individual place agent, which "attracts" the robots towards the place's center. When starting outbound from such a node, however, the green trajectory at various places shows a substantial direction error towards the target PA. Examples of such are clearly visible for the two long vertical trajectories towards the bottom end of the figure.

The sending PA releases control of the robot after it oriented the robot, whereas the receiving PA takes over control (refer to chapter 4.3.2). For large distance between sending and receiving PA the robot's field of view at the start and the PA's field of view at the target do not overlap - or have very little overlap only. So the receiving PA cannot correct the robot's motion with respect to the external environment; the robot solely relies on dead reckoning to reach the target's vicinity, until both environmental signatures (the robot's current view and the receiving PA's) have substantial common information. At this point that target PA computes motion corrections that allow the robot to center itself at the target (chapter 4.3.2). We can clearly see such trajectories in Figure 67, where various green trajectories abruptly change direction and converge towards the center of a place. Note that this only happens for trajectory segments that cover a long distance; for short segments the receiving PA's place signature and the robot's view at its start position already share sufficient spatial knowledge that allows the receiving PA to attract the robot based on sensor data immediately.

In the current implementation of our system we decided to grant only a single PA control over the robot at any time. One can imagine a system where both PAs, the sending and the receiving PA, together control the robot, with each contributing depending on its relative spatial knowledge at the robot's current position. The starting PA guides the robot while it is close to the start; the receiving PA guides the robot closer to the end of the trajectory. We will discuss such a system in chapter 7.1.1.

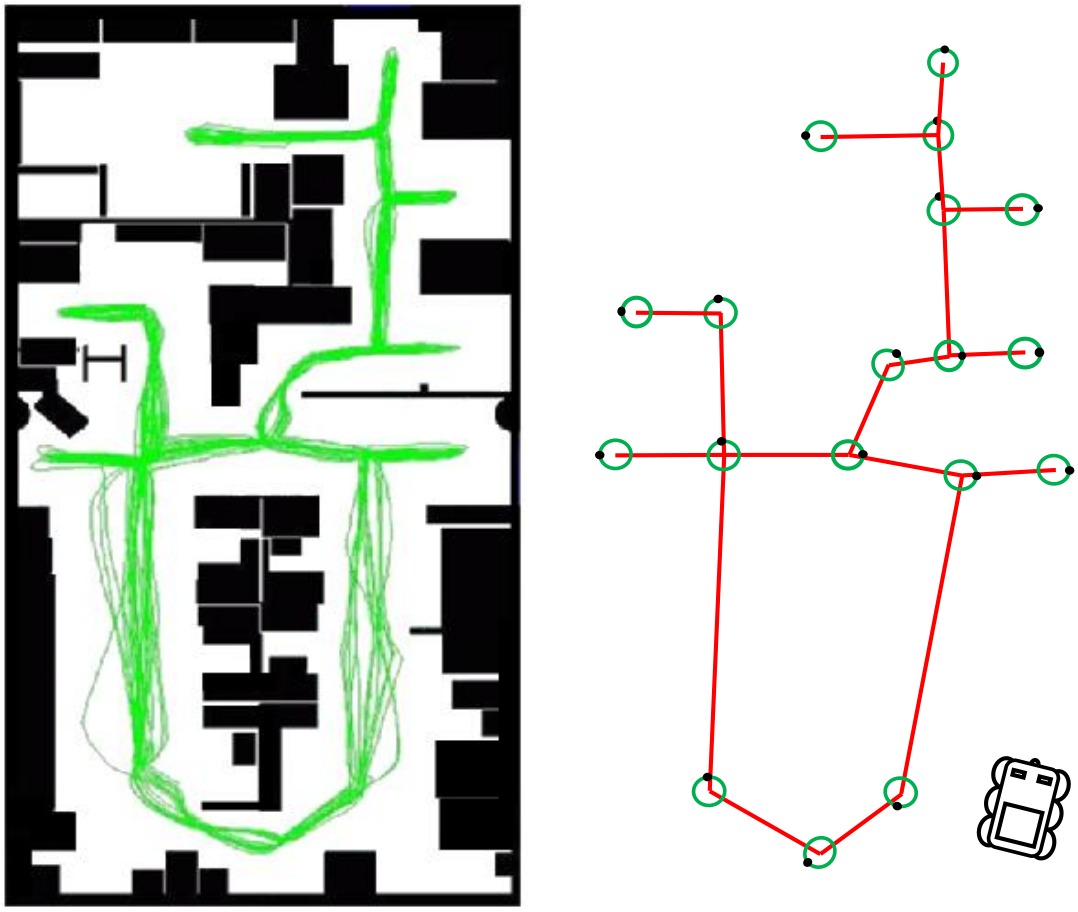


Figure 67: A recorded trajectory generated by the network of PAs to the right. Note the strong deviations from direct paths especially for long inter-PA distances (vertical segments at the bottom of the figure).

The upcoming Figure 68 shows a trajectory of the robot traveling in a larger area. Larger open spaces show even more diverging paths that the robot took. In addition, several places (the most obvious marked in red) show how the robot autonomously performs local obstacle avoidance - completely autonomous of the receiving place agent's driving commands (chapter 3.1.3). Note that the behavior activated to generate that plot performed a random walk without further exploration, so despite spending significant time traversing the environment the robot never found rooms or trajectories it missed during initial exploration.

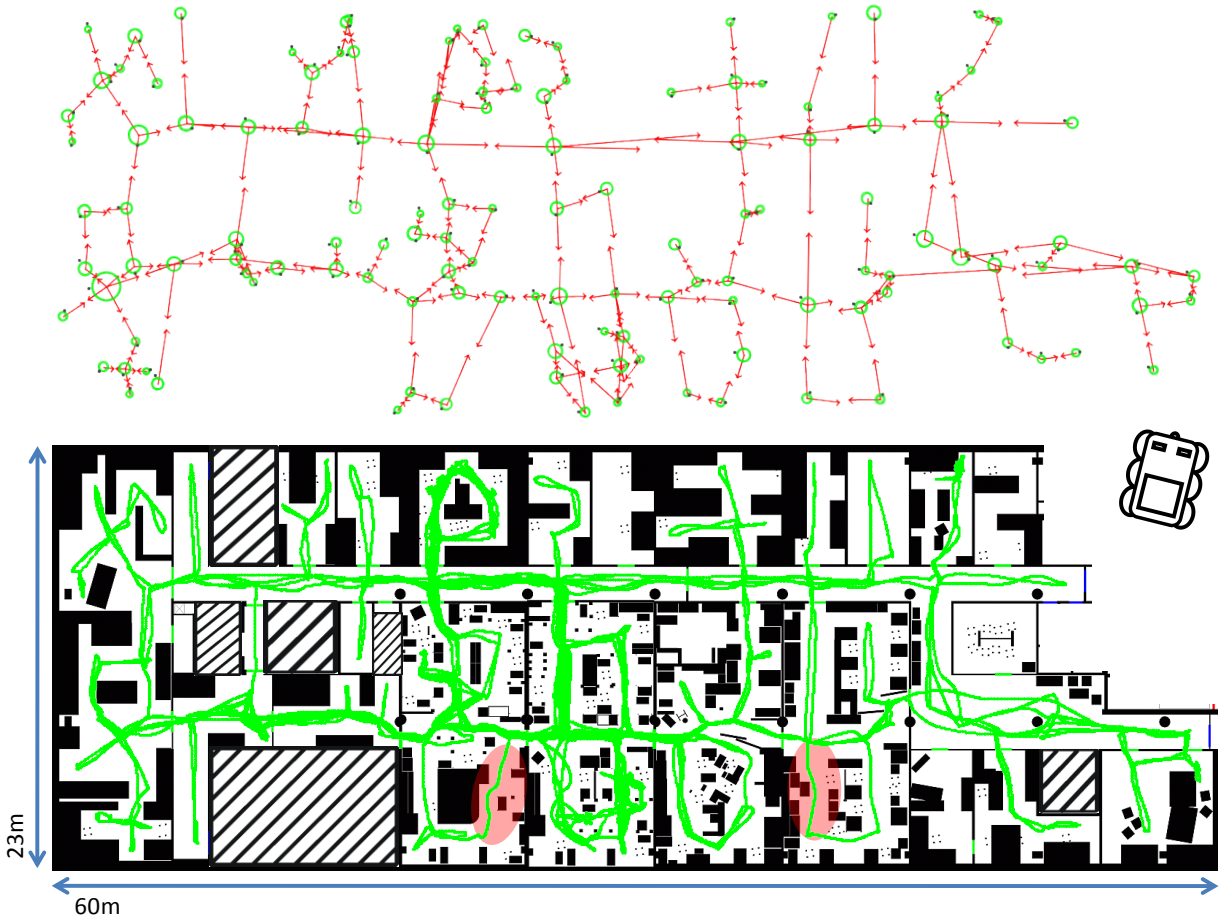


Figure 68: A recorded trajectory of a simulated random walk showing the robot's autonomous obstacle avoidance which is especially well visible at the places marked in red.

6.2.3 Position Accuracy when Revisiting Places

We have recorded the robot's simulated final position whenever it approached a place agent to inspect how close the robot returns to the PA's true position. Figure 69 shows a plot in which red dots denote a PAs true Cartesian position, whereas blue dots show the simulated robot's Cartesian position, at the time the corresponding PA finished attracting the robot to its center. We can see that most blue dots are very close to the red true positions; Figure 70 provides a detailed histogram of distances between blue dots and its corresponding PA's position. Note that PAs might adjust their position when the robot revisits, and - more dominant - when two PAs fuse into a single representation. Hence multiple clusters of blue dots slightly next to a red dot are likely to represent two initial PAs that later merged into a single PA shown by the red dot.

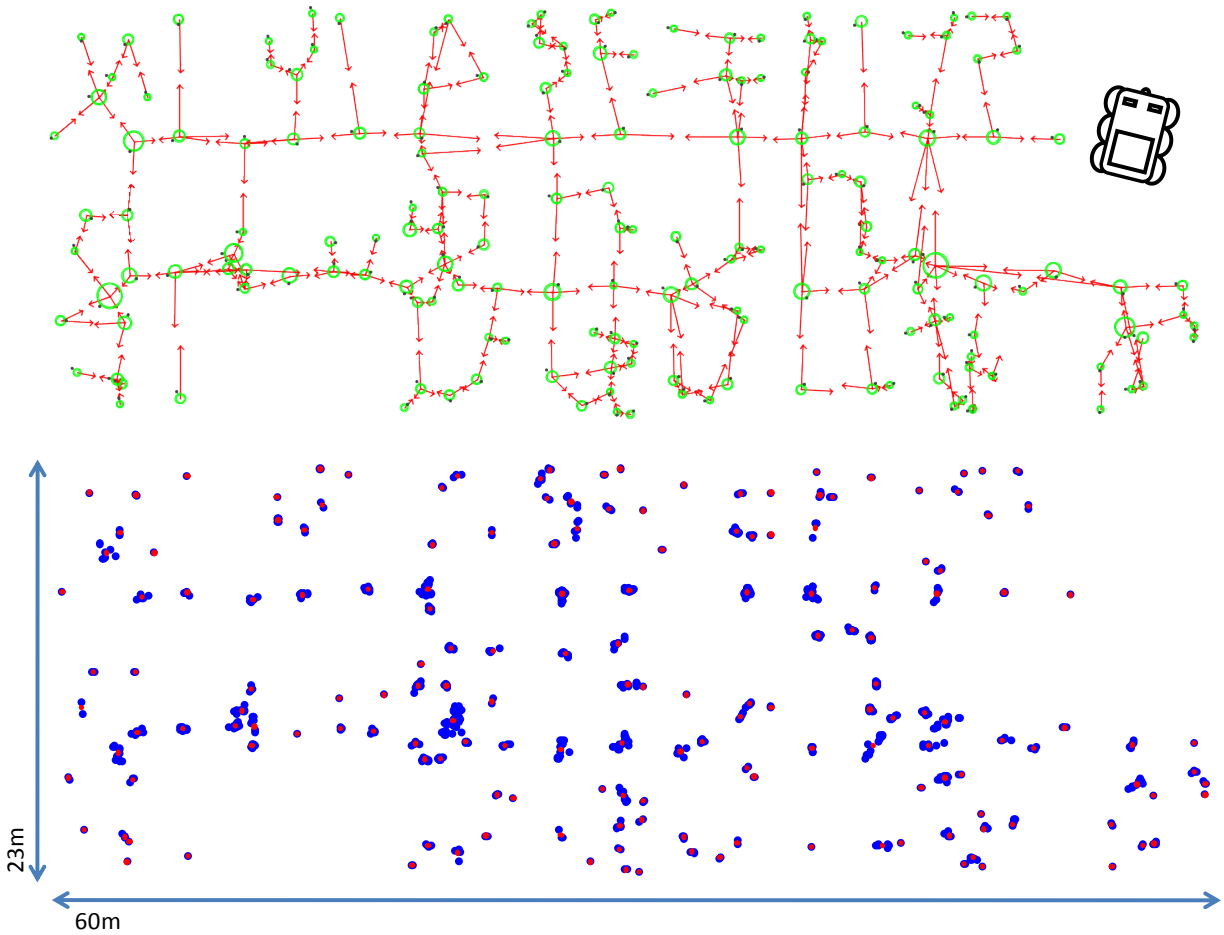


Figure 69: Simulated robot positions (blue) when revisiting place agents (red)

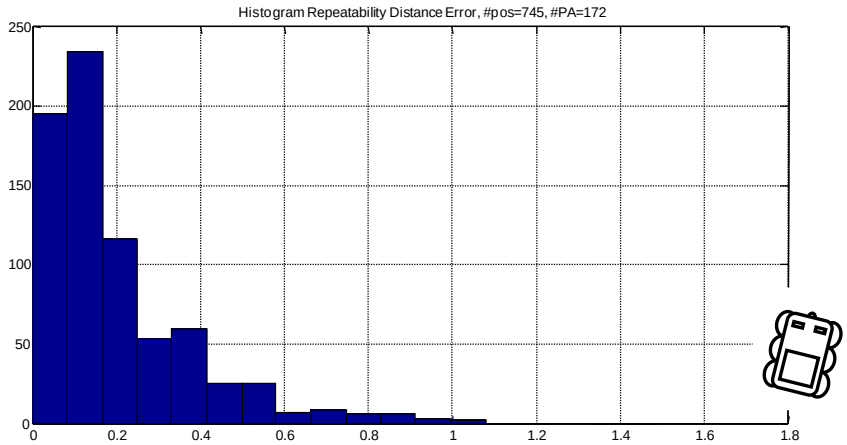


Figure 70: Histogram of distance errors between current simulated robot position when arriving at a PA and the PA's true position.

6.2.4 Distortions in the Network of Place Agents

The network shown in Figure 58 that is superimposed over a Cartesian map showing true space is a particularly good example, where the displayed positions of place agents coincide with underlying open space in the environment. Usually, a network of place agents arranged by the PAN-GUI superimposed over a picture of the true underlying environment looks more similar to the one

shown in Figure 71, where we can see a clear mismatch between the position of place agents and open positions in space. The network seems to be highly distorted with respect to real space, especially towards the upper end of the figure. Proper navigation based on such a network seems impossible.

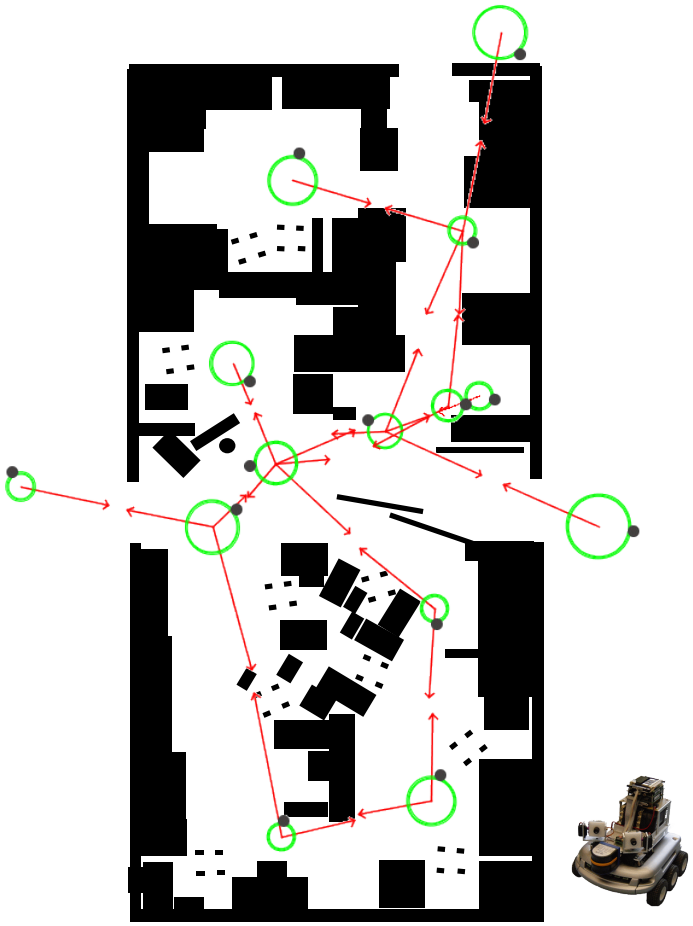


Figure 71: A representative layout of a network of nodes arranged by the NPA-GUI superimposed over true space. The network's topology represents the structure of the environment, but the exact places of PAs are severely misaligned with respect to free space.

The distortion we see in Figure 71 does as such not exist in the system! In fact the arrangement of nodes does not exist in the system; we only have individual place agents, each not knowing the position it represents in space. All such network displays are only arranged for humans such that we are able to discover the underlying structure; in the computer only single PAs exist that all know their local neighbors, but have no information about the rest of the network. Therefore, the complete system (the collection of all place agents) is unaware of the skew we perceive in the figure.

Looking at any single place agent we discover that its local knowledge regarding directions and distances to its neighbors reflects its particular local environment well; only the global arrangement of all PAs reveals local inconsistencies. Such small local errors at each of the place agents (that we already analyzed in chapter 6.2.1) are not problematic for navigation, as the robot will correct its trajectory when approaching the upcoming target PA (as shown in chapter 6.2.2). A link in the network between two PAs indicates that the robot can travel between these, as it did during

exploration. Therefore, as long as the robot finds the path between any two neighboring PAs (which it can do despite local errors), it will be able to follow a longer piece-wise trajectory in the network.

Figure 72 shows a larger globally arranged network that contains significant distortions when compared to the underlying environment; still the network is able to guide the robot in the environment without losing track of its position. Generally, the larger the network and the more loops captured in the network, the stronger the resulting distortions. In contrast, the more time a robot spend in the network, the better the individual estimates of position and angles for neighbors, and hence the better an overall arrangement of PAs. It again is a tradeoff between time and accuracy during exploration.

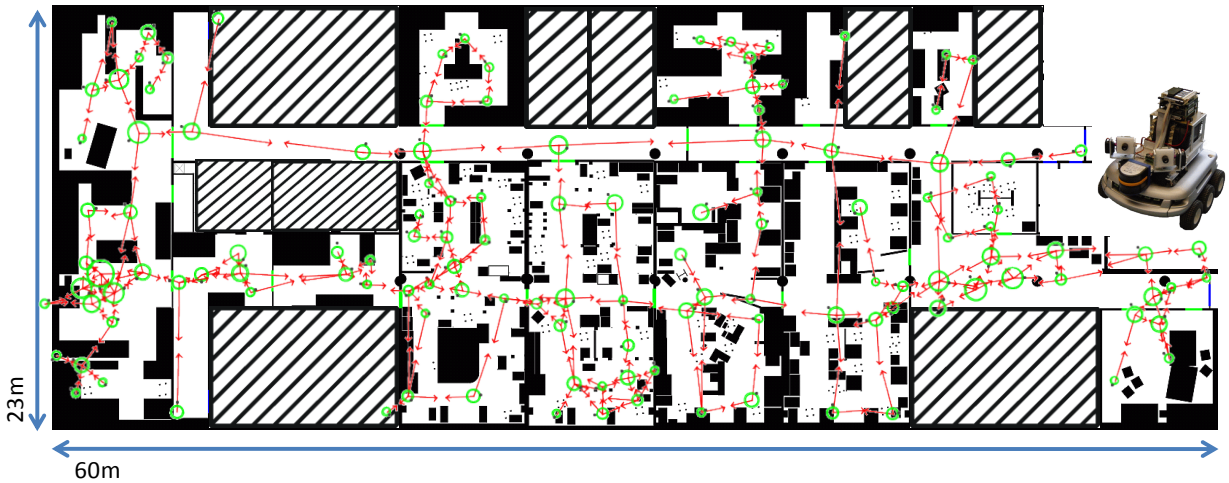


Figure 72: This network shows severe distortions with respect to the underlying environment at several places, most dominantly visible along the horizontal aisle on the top end. Still, the network can successfully guide the robot within the environment, as all individual PAs only suffer from small local errors.

6.3 Closing Loops by Merging Place Agents

When exploring an unknown environment amongst the most significant occurrences that happen to a robot is that it surprisingly returns to a place it visited before; i.e. traversing a cycle in the environment without a-priory realizing that it going to return to such a place. In traditional robotic systems for navigation such an event - when detected based on sensor data - causes a chain of actions to happen, mainly related to relaxing the error contaminated recent recordings. If there were no errors in the recorded data, the robot would have expected to return to that place, and can easily fuse all previously acquired spatial information into a globally consistent map. Typically, however, accumulated error causes enormous computational problems when detected to be corrected. Still worse, when such an event stays undetected for some time: the navigation system continues recording spatial knowledge, but records all further information with incorrect spatial references in the global map. When ultimately detected, the system has to correct significantly more recorded data, requiring more efforts to keep the spatial representation consistent.

Our distributed system deals with loops in the environment very differently: as no actor in the system is aware of cycles, we can "close" those representations at any time - or leave them undetected in the system as long as required. Upon the robot returning to a previously visited place, the new PA representing that place might discover that another PA in the network already

represents the same place. In such a situation those two PAs together create a common offspring PA, which combines all environmental information of the initial two agents, including their neighborhood relations. This does not explicitly represent a loop in the environment, as no single PA is aware of a loop but only of its direct neighbors. However, as of such an event, an implicit loop exists within the network of PAs.

Alternatively, if the new PA misses to notice that an alternate representation of its place already exists in the network, it will simply add itself as a new 'duplicate' representation of that single place. As no part of our system interprets individual nodes with respect to their true spatial position, such duplicates do not cause conflicts in the existing system. The second PA will again start explorations from its place, ultimately exploring the same space multiple times without noticing.

Two different reasons for missing a fusion between PAs that represent the same place exist: either the spatial knowledge acquired by these two PAs is insufficiently similar for fusion, or the integrated error in position between the two - as laid out by other PAs along a path connecting the two - is too large to allow fusion. In the first case both PAs coexist in harmony, until future re-visits of the robot at either of the two places enriches their place signatures. Receiving updates of the true local environment constantly increases the similarity of the two PAs' spatial signatures, until they eventually decide to merge once past the similarity threshold - possibly only after several updates by a revisiting robot. The similarity between the two representations might never reach the threshold; such places will stay represented by two independent PAs for good. An example of such can be found in appendix A, in the simulated network #5, top center; where the cycle in a small room is represented twice by several PAs, which each did not realized that another PA is representing the identical place.

The other reason for missed fusion is that the assumed spatial distance between both places is too large. This typically happens due to accumulated orientation error, such that parts of the network are rotated with respect to other parts. We will discuss an example of such an event in detail in the upcoming chapter 6.3.2. In general, the more small loops exist in a network the less likely such problems happen.

6.3.1 Deciding about a Fusion of two PAs

When a new PA decides to represent a place it emits a message (token) into the existing network of PAs, searching for other PAs with a sufficiently similar place signature; upon passing a threshold the PA considers merging with another such PA to form a common representation of that place, given that the accumulated distance error recorded along the trajectory connecting the two places falls below a relative or absolute threshold. Determining such thresholds might seem difficult, as various sensors contribute to the overall place signature, and several representations of places rely strongly on one sensor whereas other places rely strongly on another. Finally, for a robot that shall operate in vastly different environments such a threshold might best be learned and adapted based on feedback from the environment: a wrong fusion leads to non-traversable paths that the robot can correct later, see chapter 6.3.3. In this chapter, however, we will investigate similarity between various captured place signatures, clearly indicating that a broad range of threshold parameters work well for fusion decisions in our typical office environment - including a large variety of available sensors.

Such a comparison between place signatures is based on data reported by various on board sensors. In our system we have the following sensors, which contribute differently to the fusion decision:

- Relative sensors, such as a gyroscope or the dead-reckoning virtual target sensor. Those sensors do not contribute at all; they are not recorded in the respective PAs spatial signatures as a relative sensor's current reading depends on the previous reset of that sensor. A target sensor e.g. will always point to the center of the PA, thus not providing information
- On-board IR-distance sensor have only limited distance outreach of about 30cm maximum; when they trigger they show a large blob of obstructed space which is good for obstacle avoidance but very coarse and unreliable when comparing signatures. Although including this sensor in the comparison does not substantially blur the overall result, it also does not contribute significantly. We decided to ignore the IR sensors for such comparisons and only use them for on-board local obstacle avoidance.
- Ultrasonic transducers show similar characteristics to IR distance sensors; although they cover a much larger distance range (up to 4m) they only report data at very coarse angular resolution (about ±30 degrees), such that we only see large blurred blobs in the spatial representation - again, for computational efficiency we decided to neglect those values for spatial comparisons. We more sophisticated preprocessing techniques one might well be able to increase the sensor's angular resolution.
- The remaining three types of sensors, the laser-range-finder, the visual landmark detection system and the on-board compass, each provide their respective data at high spatial resolution, presenting a well structured 1d or 2d signature respectively. All these sensors contribute substantially when comparing spatial signatures.
- Abstract recorded information - such as the spatial arrangement and/or the identity of neighboring PAs - increases the richness of available information when comparing signatures. Two PAs that each have a common neighbor are more likely to represent the same place - especially if such a neighbor itself is the result of a previous fusion between a neighbor of each of the currently doubtful PAs. On the other hand, PAs that do not yet have common neighbors shall not get penalized; it might just be that they are the first instance that closes the cycle. In contrast two PAs that each know a large number of landmarks but have no common landmarks will receive a very low similarity report.
- Lastly, the distance that a token traveled in the network from one PA to a potential fusion partner contributes to the fusion decision: the larger the accumulated vector offset between the two PAs (refer to chapter 5.2.4), the less likely a fusion happens. Such a criterion is very helpful for short distances between potential PAs, as the accumulated error is small which allows a clear statement regarding the PA's vicinity. In contrast, as soon as the distance grows - especially also the hop-count along PAs grows - the larger the uncertainty in distance error.

To investigate in the similarity measure for fusion between PAs we create an artificial scenario based on real-world data: assume a robot explored the same room twice, resulting in two independent - yet identical - networks of places in this room. The resulting situation is identical as if the robot returned to that room along a different trajectory and again started to explore space in this room, without ever fusing PAs: it decides about the positions where to capture PAs autonomously. We

have cut sections out of two explored networks to demonstrate such a scenario as shown in Figure 73, where the green and the red PAs each represent the resulting network of an independent exploration of the room shown to the left. We afterwards put human-readable labels ranging from 1-13 to these agents such that we can refer to particular PAs when comparing all to all. Note that the topology of the two networks differs at two places: the red network created two place agents (7a and 7b) where the green network only used one; similarly the red network created and extra PA (13) between PAs 10 and 12.

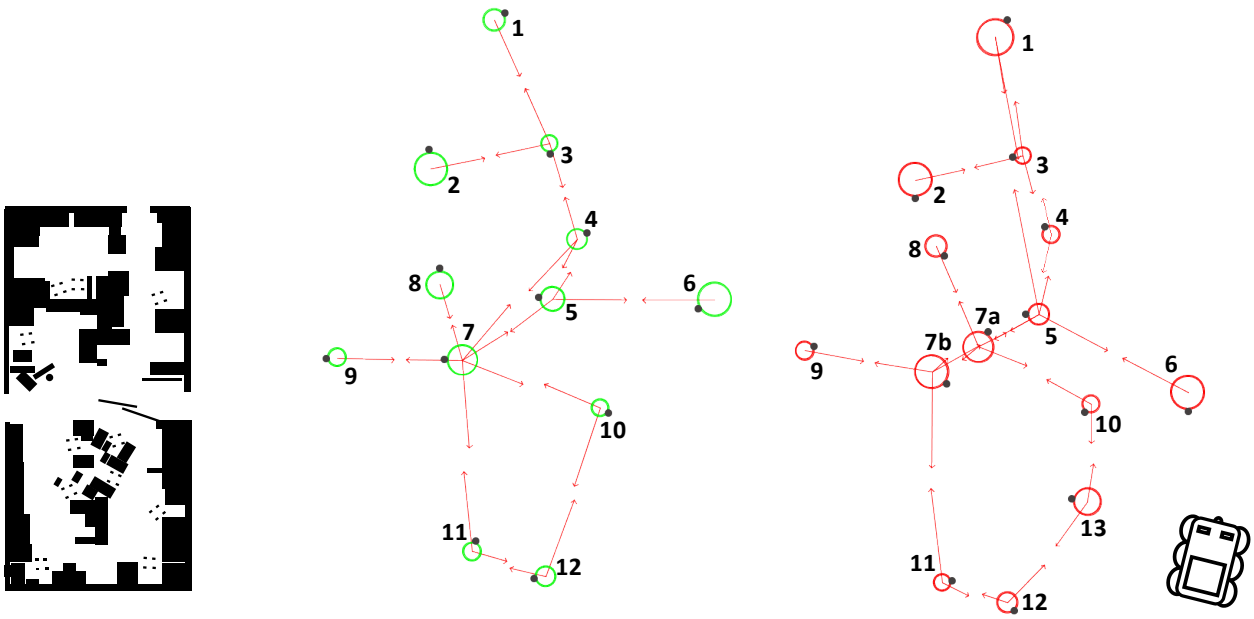


Figure 73: Two separate networks representing the same room, captured on two independent exploration runs

The following Figure 74 shows a confusion matrix, in which we compare every single PA of the total pool of 26 PAs against every other in the pool. The diagonal line in the matrix shows results of comparing a PA to itself, resulting in maximal possible similarity of values close to 1. The PAs are conveniently ordered by number, such that along the horizontal and the vertical axis of the matrix each group of two PAs denotes those PAs that represent the same place - with an exception for PA7 where we have 3 PAs and PA13 that exists only once in the pool of PAs. We can clearly see high values when comparing those PAs that are members of a single 2x2 block, meaning that the spatial signatures recorded at those nearby places reflect the similar environment of the PAs. In fact the similarity values are so significantly different, that we are free to choose a threshold value for fusion anywhere above 0.7 without having false matches. The higher we select the threshold the fewer true fusions we get, but the less likely we might encounter a wrong fusion later in the system. Both these rare but occurring scenarios are discussed in the following two chapters 6.3.2 and 6.3.3.

Note that the confusion matrix shown below only relies on data from the laser-range-finder, compass, and landmark sensors; all other "abstract" sources of information, such as neighbors or traveled token distances, are not yet taken into account and will further increase the certainty when deciding on a fusion between two PAs.

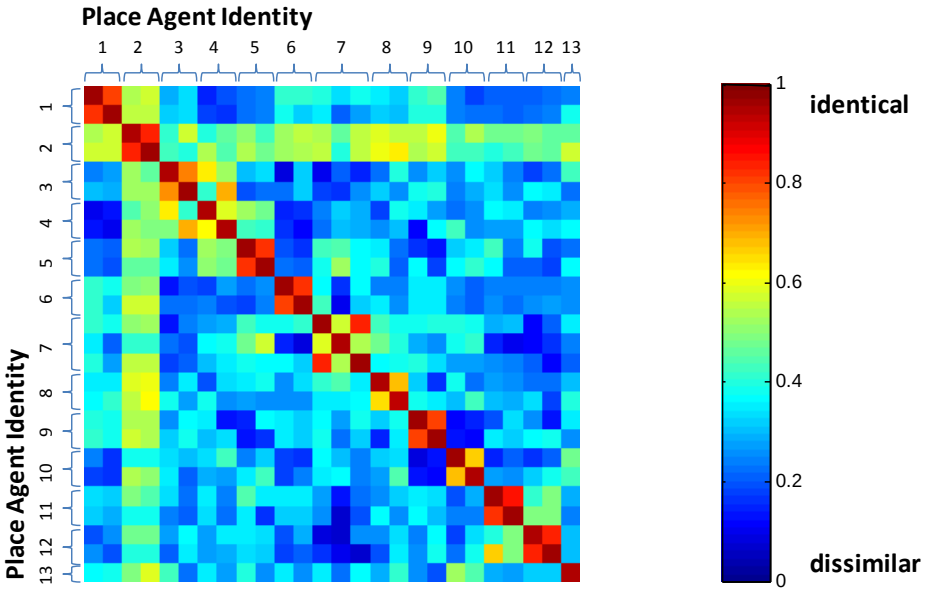


Figure 74: Confusion Matrix comparing 26 place agents; dark red denotes identity. Pairs of corresponding PAs typically show red and orange colored squares of size 2x2, also the triplet for place 7. Note that comparing PA_X with PA_Y does not necessarily yield the exact same result as comparing PA_Y with PA_X, due to the binned nature of the data representation (refer to chapter 2.2).

The figure leads to the assumption that place 2 resembles a "generic" place: it shows slightly higher than standard similarity when compared against any other PA in the network, as indicated by the light green/yellow horizontal and vertical bars in the confusion matrix. Note that this similarity is still well below the proposed fusion threshold of 0.7 - 0.9. This astonishing effect has a simple explanation: PA2 represent the "dark room" at the Institute of Neuroinformatics, which is used for optics experiments and thus is on a different light system. During explorations the lights were permanently off, and thus this room provided no visible landmarks. In such a signature the landmark sensor does not contribute to the similarity measure, but more important it also does not reveal de-similarity between places, which is what landmarks mainly do: places that each have a large number of landmarks but no or hardly any overlap - or only such overlap of individual landmarks that does not consistently translate from one signature to the other - cannot represent identical places. We can verify this statement by the following Figure 75, which shows resulting confusion matrices for three separate experiments in which one of the three contributing sensors was disabled in turns. We can clearly see that in the left-most map the dominance of place 2 vanished, after all landmarks were removed from all spatial signatures. Still, the confusion matrix without contributions from landmarks allows separating those PAs that belong together from those that do not easily.

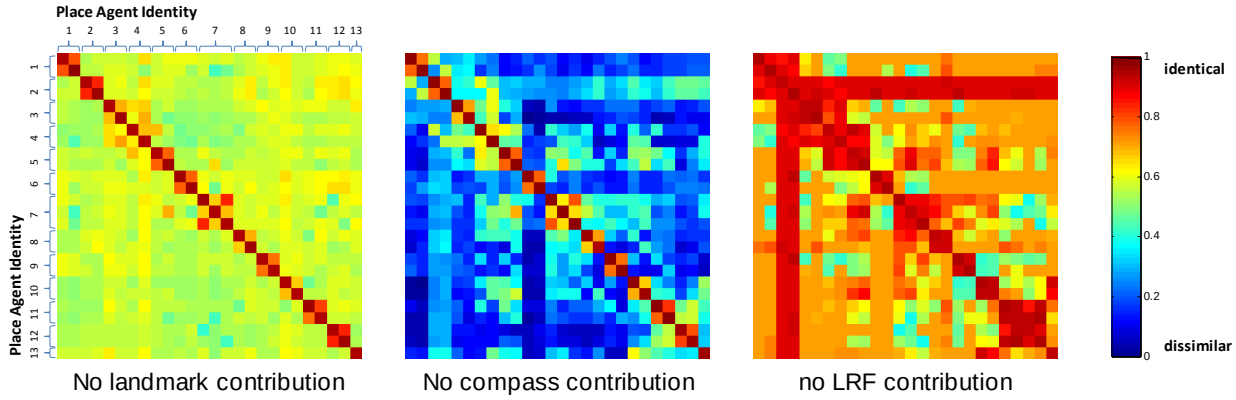


Figure 75: Confusion Matrices for the same set of 26 PAs, each deprived by a particular sensor modality. Clearly the system can operate without landmarks and without compass information. The laser range finder (LRF) in contrast seems crucial, as this sensor currently is the only source for local spatial structure.

The panel in the middle shows a confusion matrix deprived of compass data: still here, valid pairs of PAs show significantly higher similarity compared to non-matching PAs. But finally, removing all contributions from the laser range finder as shown in the right panel seems to break the system. The laser range finder currently is the only sensor on board of the robot that reports spatial structure; a comparison between PAs without such structural information has to rely on compass and landmarks only. Again, note that place 2, the one without landmarks, looks highly similar to all other places: the only information left is a compass reading towards north; and all other places have a compass value somewhere.

In this subchapter we showed several examples of similarity measures for spatial signature that represent places in space. It is obvious that for a rich spatial signature (Figure 74) we can estimate a threshold that will hardly lead to accidental merging of PAs that truly represent different places, but still allow fusion of such signatures that correspond to identical places, given that the distance-error estimated by tokens is small - which we have neglected throughout this comparison. We will have a closer look at the impact of token-distances in the following chapter. Still, the best possible threshold cannot prevent occasional misclassifications (wrong fusion or missed fusion), such that the network of PAs has to handle such situations. The following 2 chapters discuss such situations.

6.3.2 Missed Fusion, Delayed Fusion

The previous chapter discusses the similarity measure consulted before fusing place agents. As shown the measure is highly reliable, but not absolutely correct; i.e. it might accidentally miss two PAs that in reality shall get fused, or it might decide to merge two PAs that should not get merged. The latter case will be discussed in the upcoming chapter 6.3.3; here we present what happens when PAs in the network miss an option to fuse correctly.

PAs that have a sufficiently rich signature typically only refuse fusion when sensor noise has introduced severe errors in the network of place agents. Figure 76 depicts such a situation, where anyone (or possibly multiple) of the PAs marked in the red circle recorded a poor angle estimate towards one of its neighbors. The two PAs involved did not notice the error and continue sending the robot for further exploration. Ultimately, the robot starts to re-explore parts of space that it has explored before and that is already represented in the network. Figure 76 shows such an event, where those PAs representing space initially explored are marked in yellow, and those representing

the second exploration in blue. The identical topology of the two colored subnets reveals that the robot actually explored identical places.

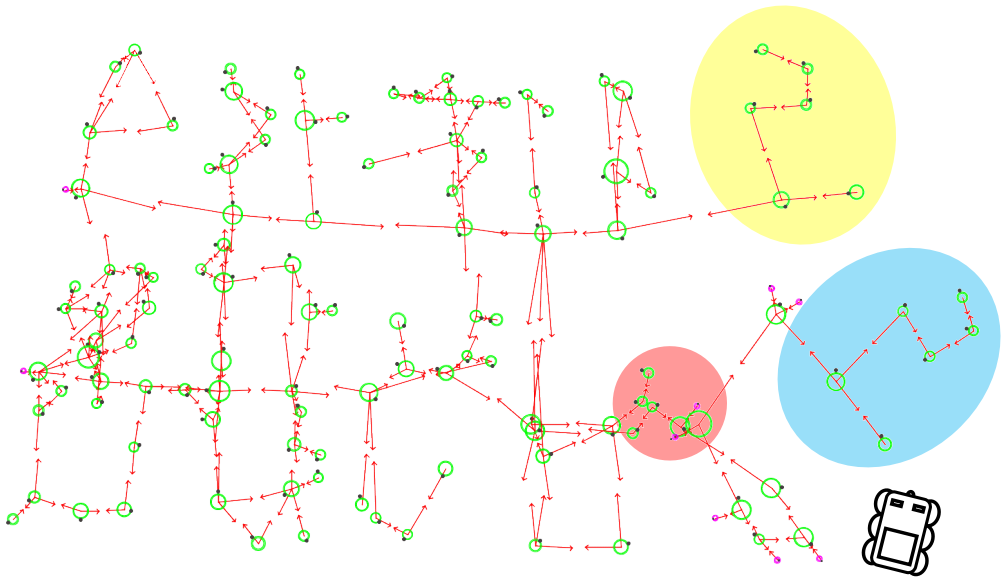


Figure 76: Local angular error between neighboring PAs (occurring between PAs in the red circle) causes severe global misalignments in the display, but also leads the system to re-explore areas (blue) that have been explored previously and are represented by other PAs in the system (yellow). The "blue" and "yellow" PAs refuse to merge into a single representation because of the angular error somewhere along the path of PAs connection the two areas.

Remember that such a spatial arrangement of PAs as shown in Figure 76 only happens to help understanding the distributed network. This particular arrangement does not exist in the system. However, such an arrangement closely resembles the distances that token traveling in the network will report. When the topmost PA in the blue area issues a token to search for fusion partners, the topmost PA in the yellow area will detect high similarity; but that token upon arriving in the yellow area reports a large distance offset, such that the PAs will not fuse.

In the exploration experiment that we happened to record, a single large angular error during the exploration process resulted in a large "double exploration" (Figure 77). Note that the networks marked in blue and in yellow look suspiciously similar - and in fact they represent the same space. The system got confused somewhere along the way, and started to "re-explore" areas that it had previously explored and represented; but the new PAs (blue) representing this new space do not realize that they represent identical places as other PAs do, because tokens searching for fusion partners report too large distances offsets between matching old and new PAs. An angular error in the system can have a severe impact on global spatial consistency. Note that no single PA in the system is aware of the problem; only the display GUI that arranges the PAs for us can realize such problems.

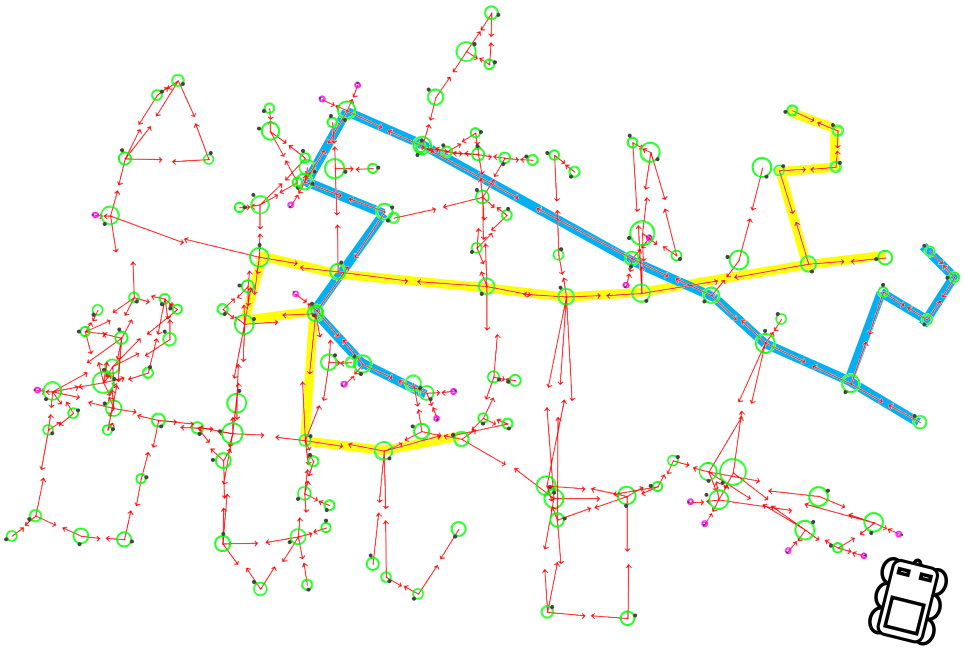


Figure 77: Place Agents in the network have sent the robot to re-explore a large area (blue) that is already present in the network (yellow), without noticing the double representation.

What will ultimately happen in such a situation? If the error is too large, the two representations will never realize that they represent the same place and the robot will create a second full map of the spatial environment. Typically, however, at some time identical places of the new and the old map show such a significant similarity (a small identical set of landmarks or a unique environmental structure), such that the similarity of PAs outweighs the distance error reported by tokens. As the token need to travel longer and longer distances to read the particular corresponding place in the old representation, the allowed relative error in the token's estimate also increases. Ultimately, a new PA will recognize its match in the old representation and fuse into a single PA representing that place, as shown in Figure 78.

After an initial set of two such PAs merged their knowledge, all other PAs in the two independent chains can detect their respective fusion partners: a token they issue can take either one of the two paths, with or without the angular error, such that all places ultimately will detect the corresponding PA as shown in Figure 79.

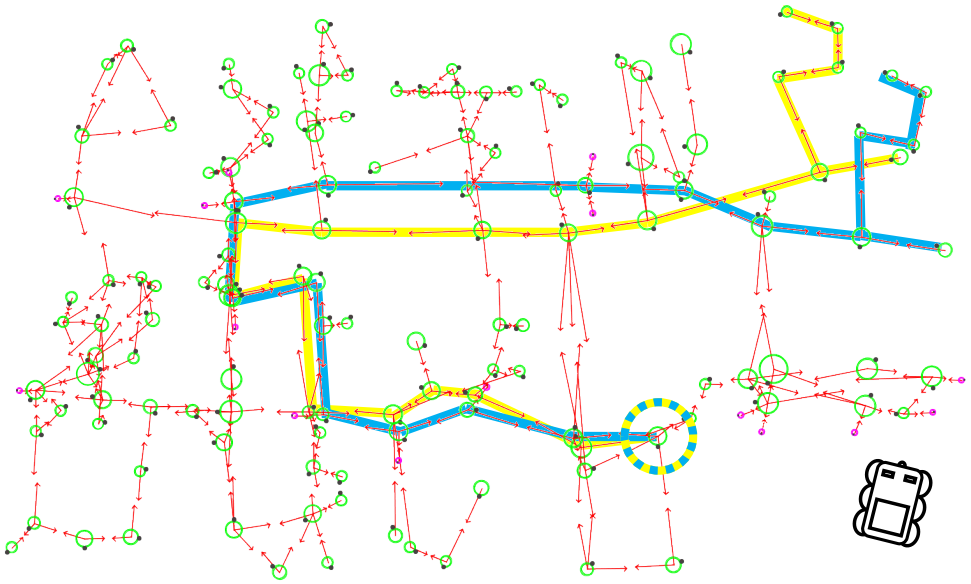


Figure 78: Here a PA from the blue set realized that it represents a place already represented by a yellow PA. Rich place signatures with significantly above average similarity and a longer path that lowers the required similarity threshold for fusion ultimately allows such an event to occur. Upon detection, the two PAs fuse into a single representation, which "pulls" the trail of PAs over.

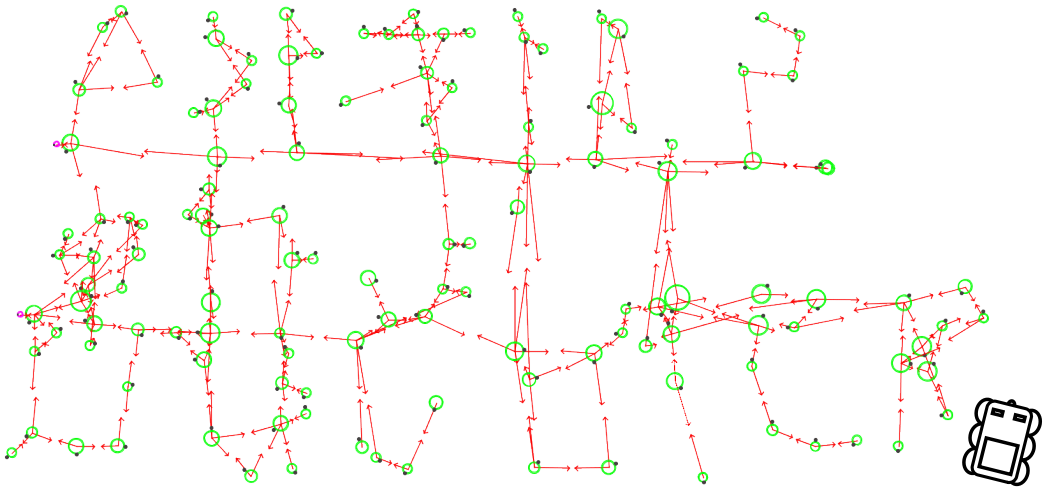


Figure 79: All PAs along the initially separate paths found their respective partners and created single representations of places.

In summary, when two (or multiple) PAs miss the fusion decision they can stay separate as long as required. Separate PAs representing identical space does not cause problems for the system; it will only take time to re-explore the system partially, until some PAs along the independent trajectories find appropriate fusion partners. In rare cases, several places stay represented twice in the system for good.

Note that after fusion the PA that has initially recorded a poor error estimate still shows the poor angular estimate: it has not learned anything about its angles to its neighbors. Only the PAN-GUI, which displays a proper spatial arrangement of PAs for our convenience, now finds a better solution to match the PAs' true Cartesian positions. The PA that maintains a poor angular estimate to its neighbor still has the same error. If the error is sufficiently small such that the robot can still find the path to its neighbor, the PA only slowly updates its estimate with every new piece of information it

receives from the robot traveling to its neighbor. In contrast, when the angular error is so large that the PA does not successfully manage to send the robot to its neighbor, it will ultimately break the link and continue to explore in the direction of its neighbor once again - thus ultimately correcting the angular error by closing another loop in another direction.

6.3.3 Incorrect Fusions

Instead of missing fusion, place agents might also accidentally fuse, although in reality they represent two different places in space. This is a situation much more difficult to handle, so we need to set the threshold for fusion rather too high than too low, thereby avoiding accidental fusions of PAs as often as possible: although the system can maintain various simultaneous representation of a single place without them interfering with each other, once two places are fused the previously unique information (about network topology) is lost, and only a wrongly united representation exists in the network.

Although we can select a high threshold for fusion, we cannot set it sufficiently high to completely avoid such a problem, so the system needs to have a way of handling such a situation. Figure 80, a, shows such a situation, in which the robot has explored a large fraction to the right of our institute (panel c shows the relevant area of the floor plan), but two unusual events happened: 1) the robot explored the long aisle on top all the way to the right, but then got recalled for network consolidation into another part of the institute, without exploration neighboring space along the aisle on top. This recall resulted in having a single PA at the top right, which correctly represents the place close to the door (shown by top red circle in panel c). 2) the robot later - after the consolidation phase - continued to explore towards the right bottom side of the institute, where exceptionally both doors leading into the seminar room at the right bottom where closed. That exploration created another place agent, which represents the place indicated by the lower red circle on panel c. Also this place is close to the door leading outwards, at the end of an aisle. This aisle is significantly shorter compared to the aisle on top, but as both places only record local knowledge, neither of them realizes the difference. Furthermore, both place signatures representing the two places are initially rather poor regarding distinct features: only one of them captured a single landmark. All other information typically used for the similarity measure - laser range finder and compass - show the same signature: an end of an aisle, with the compass oriented in the same direction (towards the open end of the aisle).

Having two places that look really similar and that are only connected through a long chain of PAs that might be error-contaminated, the two PAs assume to represent the same place - and thus fuses into a common representation, as shown in panel a, top right end. For us external observers this situation seems highly distorted, but PAs in the network cannot distinguish this situation from a situation where two PAs indeed represent the same place and have to fuse.

The real problem giving raise to such a situation is the missing spatial information about the areas marked in blue. If the robot had explored those regions before creating the second PA, the token-distance between the two PAs that accidentally fused were significantly shorted, and thus the legal error margin for fusion smaller. The similarity measure would have denied the fusion.

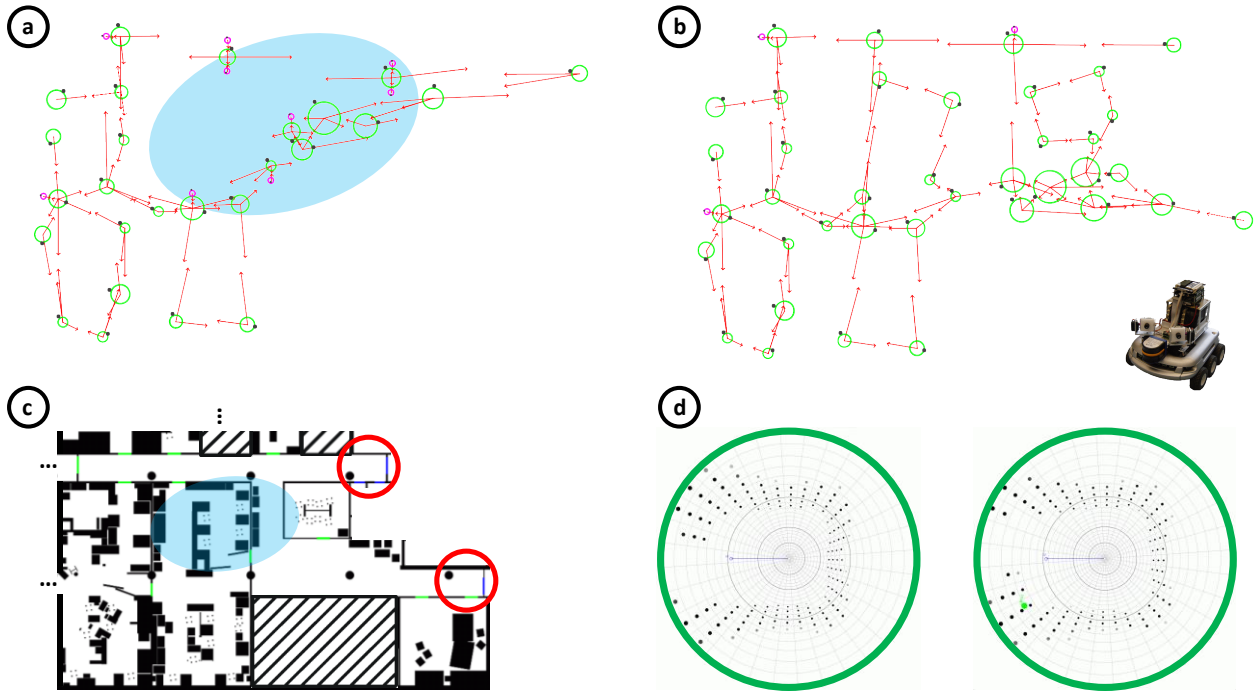


Figure 80: a) a poor choice in the order of free space to explore (blue not yet explored) leads to the fusion of two PA that present similar but not identical places (c + d), because the only connection for token between those involves a long detour. b) after correcting the error by removing one of the links, the system continues to explore open space and finally builds a complete representation.

What to do to correct the situation? With the previously introduced mechanisms the system already handles such a situation properly: it does not need to detect the conflict as such, but will recognize a local problem in the network. After this initial exploration, the robot might be at the place shown by the lower red circle - and decide to travel to the “upper” neighboring PA - that is the PA it cannot reach in true space starting from its current location; the one that only exists in the current PAs signature because of erroneously fusing with another PA. The robot will try to reach this place but ultimately give up (refer to chapter 5.2), return to the fused place and report the failure. The PA might decide to try again, but must fail again because of the true spatial layout. Ultimately, the PA will give up and render the link unusable, until eventually removing the link in the network (refer to chapter 5.2.2). Upon removing this link, the former “upper neighbor” will recheck its environment and - again - detect free space to explore, such that it will create a new child to “re-explore” this area, ultimately creating a new representation for the end of the top aisle (Figure 80, b). This new representation is less likely to fuse, as the robot will have explored space in-between the two places in the meanwhile. Note that it does not matter if the robot fails to travel to one of the two neighbors starting from the top or bottom true space position; it will remove either of the two neighbor-links and recreate the appropriate PA again. Ultimately, the network represents external space in a globally consistent fashion.

6.4 Summary

This chapter presented and discussed results obtained from multiple explorations performed by a real and a simulated robot within our institute. We initially examined how the system, starting from a nucleus place agent without any externally provided spatial knowledge, incrementally builds up a network of such place agents that represents a large environment as decomposed patches of local

information only. We analyze the individual PA's contributions to the overall spatial knowledge, which reveals that the system captures behaviorally important aspects of space, rather than representing the environment in a regular grid-like fashion. We compare benefits of different exploration strategies, arguing that a balanced mixture of exploration and consolidation (of previously recorded spatial structures) leads to best results when exploring large spaces. However, time required for exploration can get traded against structural completeness in intermittent representations of explored environments: e.g. a rescue mission might prefer a "quick in - quick out" approach in unknown environments, without learning a detailed topology of the network. An alternative exploration strategy suits such a scenario better, in which our system uses whatever structural information available.

The following subsections discussed the "correctness" of an acquired spatial representation, inspected from an external observer's point of view, which is able to see all distributed spatial information maintained in the network. For simulated environments we can compare such data against ground truth, as the simulator knows the true Cartesian position of its simulated robot at any time with respect to the map it applies for the simulation. For real robot experiments, in contrast, we neither know the robot's true Cartesian position within the institute, nor the exact external spatial arrangement[5]; hence, we reduce the analysis of such networks to their internal consistency. All these analyses show how closely the networks of distributed knowledge represent the true external environment: several places show high accuracy, whereas several other places show lower accuracy, yet sufficient to guide the robot during local navigation. Note that all such evaluations in this subchapter about consistency with true space provide interesting insight into the network's internal representation, but ultimately it really does not matter how "correct" the environment is represented in the graph. Ultimately, we want to demonstrate that a distributed representation of local patches can actively exhibit globally consistent behavior, which we can do as soon as the individual local representations are sufficiently correct to guide the robot at their respective local place and to anyone of the neighboring local places. Sub-chapters 6.2.2 - 6.2.4 investigate this local correctness, demonstrating that the locally maintained knowledge in the network is sufficient to guide the robot between neighboring places.

Extending the network's behavior beyond local navigation, chapter 6.3 discusses results of PAs reworking the network's structure by adding and pruning local connections to reflect the true space's topology within the network. Conflicting situations that initially seem problematic - such as a missed fusion of PAs that represent a single place, or accidentally fusing two PAs that in reality represent different places - are discussed, and we present simple strategies to overcome such problems. The chapter thus presents how our system - with little computational effort - solves the difficult problem of detecting and closing loops in external environments.

With this chapter we finish the presentation of the principles and the performance of our system. The following chapter 1 discuses several optional extensions to our system, and places our approach in context to traditional methods of navigation found in current engineering and scientific models.

[5] See Chapter 7.5 for an exception, where we use an overhead camera to track the real robot within a small region of space, and use well characterized objects to construct an exact external environment.

7 Discussion and Conclusions

This chapter presents several modifications and additions to our currently implemented system, that increase flexibility or improve performance - and outline ideas beyond the scope of basic navigation in an unknown environment. We demonstrate the flexibility of our spatial representation a) by discussing an extension into 3-dimensional spaces, which our system handles with no more effort compared to its 2-dimensional representations; and b) by allowing it to represent "non-stationary places", such as escalators, elevators or even airplanes - which are difficult to model in traditional systems.

A brief discussion about the most dominant differences between our system and existing artificial navigation systems follows, highlighting advantages and drawbacks of our distributed representation when compared against such globally managed systems. Following we address some navigation studies performed on humans and animals, most of which show result that we can relate to concepts in our model. We do not, however, have results of an experiment which clearly indicates that brains use a distributed method like ours to represent space. Finally, we discuss some ideas how such a distributed information processing system can be interesting for modeling problems outside the spatial domain, and present an overall conclusion.

7.1 Motion guided by a Network of Distributed Actors

7.1.1 Multiple Contributions to Current Robot Control

In chapter 4.3.2 we presented how the robot travels an elementary path - a single link - from a starting PA_S to any one of its neighboring place agents, a target PA_T. Once on this journey, the target PA_T alone guides the robot by comparing the robot's current view with PA_T's stored signature; i.e. with the signature the robot will see once at the target. Especially for longer distances between the direct neighbors PA_S and PA_T, the receiving PA_T encounters initial difficulties guiding the robot, because its own spatial knowledge does not have any overlap with what the robot currently perceives of the world: the guidance initially has to be based solely on dead-reckoning.

If - instead - both PAs involved in this transition issue control commands to the robot, they together control the robot well all along the track: the sending PA_S possesses detailed spatial information about the area around the robot's starting position; the receiving PAT instead possesses detailed spatial information about the receiving area. Weighing each PA's contribution according to the robot's distance to the respective PA yields a single but more accurate motion control command all along the way, compared to a command computed only upon the receiving PA_T's spatial knowledge.

7.1.2 Multiple Concurrent Hypotheses: a Lost or Kidnapped Robot

Extending the idea from above, instead of a single PA or two neighboring PAs controlling the robot at any time, we could imagine that all PAs in the network contribute their individual "best driving commands" to the robot all the time. Each PA estimates an associated certainty value, indicating how sure this PA currently is to do the robot any good; e.g. a PA that notices the robot currently perceives a completely different environment compared to its own spatial knowledge understands that it can contribute little to controlling the robot, and hence should not be taken too serious in its

driving estimate. As multiple PAs in a network represent places that are perceived highly similar - and thus might try to control the robot while it is elsewhere in the world - we imagine shifting a "bump of activity" from one PA to its neighbor, when this PA sends the robot to its neighbor. Such induced activity resembles the network's certainty that the robot currently is at a particular place, independent of its current spatial perception.

Such an approach neatly addresses the so called "lost" or "kidnapped" robot problem: imagine some unfriendly experimenter put the robot at a random place into an existing network, and asks the system to localize the robot within the network. Usually, a single snapshot of the robot's current perception is insufficient to localize it with high certainty amongst the various existing place agents. But all those PAs that perceive the current sensory report to be similar to their own signature can agree to send the robot to their respective neighbor that is closest to the direction the robot currently faces. Each such a PA sends a small bump of activity to its respective neighbor; so e.g. three PAs in the network that each think they have the right to control the robot, each try to send the robot to their neighbor and each signal their respective neighbor to receive the robot. Of those three neighbors possible one or multiple will fail to attract the robot to their respective place, as it truly is at a distant location in space. Those PAs failing will decay or even remove the bump of activity - whereas finally the one that succeeds in attracting the robot maintains and possibly increases activity. A single transition to a neighbor might not be sufficient to determine a unique position, so the procedure might repeat among multiple steps, eliminating possible positions of the robot whenever new inconsistencies arise - and ultimately finding a correct position.

Controlling the robot simultaneously by a large number of PAs initially seems an appealing approach, but on closer inspection one realizes that such a situation violates our principle of "local information processing only". As a large number of PAs - up to ultimately all of them - relate their knowledge to the robot's actions simultaneously, the robot acts as a global element in the system. However, it is easy to imagine how the above described scenario of blobs of activity in the network can be implemented by tokens to solve the kidnapped robot problem.

7.1.3 Bi-directional Transitions between Neighbors

A legal path that leads from a PA_1 to a neighboring PA_2 by definition must be traversable, as the system established the link between these PAs only after a robot traveled along such a path. However, the reverse path must not necessarily be accessible: imagine a small downwards step in the floor when going from PA_1 to PA_2 that represents an obstacle the robot can only ever overcome in one direction - or simply a narrow passage that is easy to enter from one side, but difficult at the other end, like a funnel.

In our current system all links are treated as bi-directionally identical links, such that every set of two PAs assumes the robot can travel equally well in either direction. However, it is straight-forward to see that each of the PAs can remember an independent cost for the link, up to marking the link illegal from one side. The network's communication methods (tokens and invitations, refer to chapter 5.1.2) need to get updated to reflect independent costs for inbound and outbound travel, but besides the rest of our distributed system can operate as currently on such different cost measures.

7.1.4 Motion Primitives

Throughout this thesis we have presented the connection between two neighboring PAs as a directly traversable path; and by design when the true path taken by the robot diverts substantially from a direct path, the system introduces a new place agent. Choosing direct paths was an arbitrary design decision that maximized our comfort in the implementation. We could also imagine a "pool of primitive trajectories", of which each takes the robot along a different path towards a neighbor. The sending and the receiving PA only need to remember which one of the set of primitive trajectories to apply, and the robot can repeatedly travel between the two PAs. We do not need direct paths.

Taking this idea further, we do not need a pool of fixed trajectories; but instead we can memorize a behavior at a PA, that leads the robot to transition from that PA to one of its neighbors. So really at each PA in the system, the mobile agent can perform one of a set of local known behaviors: charging battery, preparing coffee, etc - or trigger such a behavior that gets it to one of the PA's neighbors.

The connection between two neighboring PAs thus actually is represented by a behavior. Our simple set of behaviors to choose from has only a single entry: "drive forward and perform obstacle avoidance"; but we can quickly implement more complex behaviors such as driving along an arc, line-following, aisle-following, etc. Whatever behavior succeeds in taking the robot from one place to a neighboring place suffices as link between these two places. Such a behavior does not have to relate directly to a fixed distance in a metric system, it might well be a behavior described by "step on the escalator and wait until the end", or "enter the one room with the red light at the door", as we often do in waiting rooms. Note that the sending and the receiving PAs are the active elements in our system, which know what behavioral repertoire they can offer and how to execute such behaviors - it is not the robot itself. All these behaviors only have a meaning while at a particular PA; they are not relevant elsewhere in the system.

7.2 Representing Spatial Structure

7.2.1 Efficiency of Representation

Our main objective in this approach to represent spatial information is that - after exploration - the system enables a mobile agent to exhibit globally consistent behavior based on its acquired locally distributed knowledge. We are no "map makers", who typically request a map to be as precise as possible based on all recorded data. Therefore, in contrast to standard metric mapping approaches, our system only represents behaviorally relevant places from an environment. The presented place agents (chapter 1) themselves determine which aspects of an environment are relevant and what needs to get represented. Currently, that decision is based solely on local spatial structures, but we could quickly extend the criteria to cover particular objects (e.g. a coffee machine), particular behaviors, or those places to which the robot needs to deliver goods, etc.

Representing only places that are relevant for behavior reduces the computation and memory requirements, as typically several parts of the environment do not get memorized at all. The analysis of explored networks in chapter 6.1.2 reveals that - on average - the system created one PA for every $10m^2$ of total space explored. The local radial outreach of a single PA's knowledge in our system is 5m; so the local area represented in a single PA's place signature is $A = \pi \cdot (5m)^2$, which is

about 78.5m^2. The system uses data for 78.5m^2 to represent 10m^2 true space? That does not seem very efficient!

We need to consider that an individual representation maintained by a PA - a place signature - uses a log-polar binned structure to represent data, meaning that those places towards the outside of a place signature are mapped into very coarse bins, while information from close to the center falls in very small and thus precise bins. We employ a total of 32 bins in radial and 32 bins in distal directions, hence a total of 1024 bins per place signature. This allows representing data in the vicinity of the center in tiny bins of sizes below 10cm^2, whereas the same number of total bins arranged in a grid results in bins of size 766cm^2 and 97.7cm^2 for a total area of 78.5m^2 or 10m^2 respectively.

The binned log-polar representation thus allows keeping spatial knowledge close to the center of the signature in very high detail, but information from further distances with only low spatial resolution - at an overall identical cost compared to a regular grid of boxes that span 10 times the size of the smallest log-polar bins. As our complete navigation process operates on local information only, no actor in the system requires precise spatial information along the outside of such a place signature; but many do require high precision close to the center, in order to perform behaviors accurately.

Those navigation systems that maintain large areas in a global data structure cannot use log-polar binned memory, as they ultimately have to represent various important places away from the center of such a structure. Here, we simple create a new PA for such an important place with a new log-polar based place signature, which again represents the relevant space in great detail in its center.

For simplicity we have chosen identical parameters to compute all log-polar bin sizes in our system, resulting in an identical pattern of bin sizes for all place signatures contained in the network of PAs. We can, however, easily imagine that each PA independently adapts the sizes and the number of its bins to suit that particular place's structure - and thus get more efficient in representing space.

Finally, regarding the efficiency of required computing power, again we gain huge benefits from distributing the spatial representation amongst individual PAs that operates on their individual local knowledge only: a) we only need to maintain local consistency within a PA, which is significantly easier to achieve compared to global consistency in a large map (see chapter7.4); but also b) we can immediately distribute all such local processing units onto multiple parallel computing devices, e.g. a small network of standard computers, if the computing power available in one machine becomes insufficient. Taking the approach to an extreme, we might consider a network of tiny computing machines - such as microcontrollers - that each processes a single PA. Note that communication between PAs in such a distributed system only happens occasionally - for maintaining the network's structure or searching for a distal target - and is not time critical. In fact, only one PA at a time needs to act in real time: the one that is currently controlling the robot. We can imagine an implementation in "cheap" hardware, in which a single microcontroller processes the one currently active PA, and another microcontroller processes all other PAs in turns. We expect that we do not need a standard computer to operate such a distributed navigation system.

7.2.2 Concurrent Representations of Places and Closing Loops

Each place agent in our system represents a local place of the external environment, but none of such PAs has any understanding of the true Cartesian position or orientation of that place with respect to other places in the environment. In fact, the PAs do not even try to estimate such a position; they do not care about their own position in a global coordinate frame. The only - facultative - instance in the system that tries to estimate such a position for each PA is the display GUI (refer to chapter 4.4), that arranges PAs nicely for a human-readable display. The navigation system does not use this display to exhibit behavior.

Unaware of their spatial positions and separated as independent computer programs, multiple such PAs do not interfere with another; even in such cases where multiple PAs represent the same place of an environment: imagine a robot returned to a previously visited place along a new path reports a place signature that is similar to the previously captured signature, but not sufficient for the two representations to fuse in to a single PA. In such a situation two independent PAs exist that both represent the same place. The display GUI realizes that both these PAs are close to each other - or even on top of each other - according to the integrated position information along the trajectory that connects them; but the PAs themselves do not. An example of such is visible in Appendix A, simulated exploration number 9, in the top left corner, where the robot explored an area twice, representing each significant place by two independent PAs.

In a traditional 'flat' map-like representation such a situation causes problems, as the existence of two nearby PAs needs to get relaxed into a globally consistent representation. Here, in contrast, such PAs simply coexist unaware of the respective other; each increasing the richness of its local spatial knowledge whenever the robot revisits, and possibly at some point in time fusing with its partner PA. Until such a fusion happens they each represent a possible place for the robot to go - although the same place in reality - and tokens are free to pick either of the two when competing for a target. We have seen situations in which complete regions in the environment were intermittently represented by two disjoint sets of PAs, and tokens picked a path represented in either set of PAs to cross that region, depending on the currently active behavior. Ultimately, all members of such representations tend to fuse as soon as a first fusion combines two members of the groups, resolving such a suboptimal situation.

Allowing such delayed fusions opens a completely new approach to handling cycles in an environment: whereas traditional globally consistent spatial representations have to decide immediately upon returning to a previously visited spot whether and how to re-arrange erroneous spatial knowledge acquired along the past trajectory, our network of PAs can keep two independent representations of that place as long as required. This releases our system from the expensive process of maintaining all intermittent sensor information until a final decision about the global consistent spatial representation is taken. We in contrast take independent local snapshots that do not require to be globally aligned, and such we can allow a delayed fusion of doubtful places to close cycles only once the places involved are sufficiently certain about their fusion decision; or we postpone such a fusion infinitely, still having a properly operating navigation system that maintains a slightly suboptimal spatial representation.

7.2.3 Representing 3D Space

This thesis presented all environments for exploration as flat 2-dimensional structures, whereas the real environment the robot operates in is a 3-dimensional world. Investigating how our system can represent 3-dimensional spaces, we distinguish between two different levels of 3d: a world that is composed of several discrete 2d layers, such as a building consisting of multiple floors, is a different level of complexity compared to a true 3d-structure, where an agent - possibly a flying robot - is allowed any position in 3d space.

Our system can directly represent environments that consist of multiple layers of 2-d space without any modifications: a slope that connects two places on different levels for our system still is a trajectory that connects these two places. The fact that two places are on different altitudes in the real world is not captured in the system, but also is not required for navigation. Each of the places at the end of the slope leads into a network of places representing the respective floor; if the mobile agent wants to change from one network to the other it needs to go along the slope as the only connection between the two sub-networks. The fact that it has to follow a climb or descent is not relevant when changing between the two networks. So ultimately, in a tall building, each floor is represented as an independent sub-network of places that is only connected to the respective floor below or above by a set of place agents that represent places in the stairway.

Much more interesting is the system's representation of a building that has no staircase, but only an elevator to move the robot between floors. In such a building the order of floors from ground to top has no meaning for navigation: the mobile robot only enters the elevator, selects a target floor by pressing a button, and leaves the elevator into that particular floor. Our system represents "the elevator" as a single place by a single PA, which has a large number of neighbors: one for each floor. This elevator PA can guide the robot to any floor, selected only by the particular button pressed. Each floor is represented as an independent network of places that does not know about the other floors; except for having a trajectory to each other through the elevator PA. The true elevation of all these floors - with respect to ground, but also with respect to other floors - does not matter for navigation. As the robot cannot move from floor X to X+1 except through the elevator, the places of floor X and X+1 do not need to know that they are closer together in real space compared to the places in X and X+2. In behavioral space their distance is the same, represented by the behavior "go through the elevator PA". Therefore, our system has no reason to maintain a correct order of such layers - in fact the system is unaware of layers; only our human-readable display suggests that we have layers, each spanning of as a sub-network starting at the elevator PA.

The second level of representing 3-dimensional spaces, in which a possibly flying agent needs to reach arbitrary places in 3-d space, requires a modification of our system; however, the modification is conceptually simple and straight forward. Each place signature, instead of representing a local 2-d patch in a log-polar binned structure, needs to represent a local 3-d spatial patch, possibly in a log-polar-polar arrangement that we can visualize as a sphere instead of the circles presented in the thesis. In such a structure, not only distances and angles towards events (including neighbors) within a plane can get represented, but the additional angle allows representing events with respect to the height in with they occurred. All other concepts, from maintaining spatial information (chapter 2.5) to message exchange in the network (chapter 5.1), remain conceptually intact, but get extended into 3-d. All principles allowing to exhibit behavior in a distributed spatial representation hold in 3-d space as they do in 2-d spaces, e.g. finding a path to a distant target.

7.2.4 Hierarchical Representations

In the current system one PA is as important as any other. Whether a PA codes for a desk, a place in a hallway, a tram stop, or any other real world position - all PAs represent elementary places in an environment the robot operates in. This has the disadvantage that searches for a target we know is far away need to address all tiny incremental steps along possible routes. It would be beneficial to group parts of an existing network under a common representation, e.g. having an element representing “the institute”, “the university”, or “Zurich”. When we want to travel from ETH to Caltech in Pasadena, California, we might find a path stating “ETH - main station - airport Zurich - airport Los Angeles - Pasadena - Caltech”, instead of searching all tiny streets in downtown Zurich for a possible path to Pasadena. We can think of such place representation as place agents on a different level in a hierarchical representation.

The main advantage of such PAs is that they contribute connections between themselves and other PAs representing distal places, without initially providing all the details about the final transitions. It is sufficient for such a PA to know that there is a way from “ETH” to “main station”; it does not need to know the exact path which is represented by PAs on a lower hierarchical level. A search process for a distant target can operate much more efficiently on such broader knowledge.

However, such an approach seems involve global actions on the network of places: how does a node up in the hierarchy determine which lower nodes to represent? It needs to inspect large regions of a network at once to reason about the true space represented in the part of the network; e.g. by detecting loops or connected regions with a only a single or at most a few paths leading out - such as an institute. We expect that such an approach needs to operate globally on the distributed network, so it is beyond the scope of this thesis and we leave this question for further investigation.

7.3 Open Directions for Future Investigations

This brief chapter lists several aspects that we determined relevant for a navigating system, but those have fallen behind the outreach of this thesis and are left for future exploration:

Non-Stationary Environments or Additional Actors

We focus on representing stationary environments, with an exception briefly discussed in chapter 5.2.1 for quasi-stationary objects, such as doors that modify the network’s topology in a binary fashion: either a path is traversable or it is not. A real environment that a robot shares with humans in contrast undergoes permanent changes when chairs or desks get moved or - more difficult - when human or other robots simultaneously operate in the same environment. We have presented ideas to capture structural changes in environments, but we have not yet investigated in representing other agents that interact within the same environment simultaneously.

Learning - instead of Tuning - Parameters

The current implementation performs well in our structured office environment, but might perform worse in larger open areas, or generally in structures that are different compared to typical offices. Various parameters in the system regarding the representation of local spaces, such as minimal distances between PA, thresholds for similarity, etc are tuned for this environment. We would like to investigate a learning scenario, in which the system adjusts such parameters according to success and failures when returning to previously visited places. While exploring such parameters the system might need to discard previously learned spatial knowledge and have to restart exploring spaces, but

ultimately such an approach is more likely to find a suitable representation for any environment compared to the current hand-tuned solution.

Behavior Interaction with the Spatial Representation
In the current stage of the project we focused on acquiring and maintaining spatial information in a representation that is well suited for a mobile agent to later behave in a globally consistent fashion within that explored environment. We implemented several exemplary behaviors and a simple selection mechanism that alternates activity between such behaviors based on a winner-take-all principle.

We expect that a more advanced system shows best performance when behaviors get closer interlinked with the existing spatial representation, e.g. by interacting directly with a subset of PAs in the network. A “find-coffee” behavior might directly tap into those PAs that provide coffee, and trigger activity directly at those places, instead of sending tokens or invitations forwards through the network. Ultimately, we aim to merge the distributed spatial representation with other distributed models that all together generate consistent behavior in an agent.

Technological Extension: Implementation in Tiny and Cheap Distributed Hardware
Although our system runs as a collection of completely de-coupled processes, we still run the whole network of place agents on a standard computer. Ultimately, we would like to present a system that is implemented on-board of a small mobile robot, enabling that robot to explore an arbitrary environment completely autonomously; e.g. in a search-and-rescue mission or on a foreign planet. We expect that we can map our system 1:1 onto such distributed hardware.

Technological Extension: Supporting Multiple Robots for Concurrent Map Building
Although not relevant for understanding principles of representing space, a current research interest is how to combine knowledge acquired by multiple robots that together create a representation of a commonly explored space. As a first working hypothesis, such robots can join their respective individual maps when sufficient overlap exists, although such a method seems to require a global supervisor.

7.4 Conclusions with Respect to other Robotic Navigation Solutions

Robotics, in the past decade, has presented impressive solutions for solving the SLAM (Simultaneous Localization and Mapping) problem; for reviews see (Adams 2007; Thrun 2008). These techniques concentrate on creating a precise representation of a robot's environment, which allows estimating the robot's position with respect to features in the environment. Most of these approaches use a global coordinate frame, while several amongst those represent current landmark positions with respect to the current robot position in local coordinates. The core SLAM problem - mapping and localization - has been so much in the focus of mobile robotics research, that subsequent problems (such as representing a target, actions and behaviors, or computing a trajectory that involves shortcuts or detours) have been treated in isolation, rather than looking at "mobile" in "mobile robotics" as an overarching problem. Consequently, while there are individual techniques for solving each aspect of the problem, there are no simple systems that successfully solve the entire problem on moving mobile robots in real world environments.

We agree that the mathematical framework of SLAM is appealing; however, the required processing power and the global nature of processing and data representation does not make it well suited for small robot implementations - not to mention an implementation in neural-like distributed hardware. Most robotic SLAM solutions use the fact that the world is 'globally consistent' to increase the overall correctness of their spatial representation by adjusting angle and distance measures based on a geometrically correct world. This improves their navigation ability in the sense of localizing the robot exactly with respect to a global origin, and allows aligning perceived features on a global map properly; but in contrast causes huge problems regarding computational requirements for map alignment and loop-closing.

A few implementations of navigation systems use hybrid topologic and metric maps for navigation, which reduces the problem of computational complexity significantly (Bosse, Newman et al. 2003; Kuipers, Modayil et al. 2004; Tapus and Siegwart 2006). All these systems still place mapping and localization in the focus of their research, rather than behaving in space. They use local maps on different scales, ranging from overlapping large areas to discrete places. The boundaries between such places are determined either by human reasoning ("a room") or by direct sensory perceptions ("change in landmark count") - whereas we argue that only the behavioral relevance of a place should lead to a representation of that place in the system. Such behavioral relevance might be determined with respect to pure navigation (intersections of aisles) or with respect to actions in an environment (charging batteries) - but clearly several places exist that have no relevance for the robot and do not need to be represented accurately to reduce computational complexity.

The most significant difference between our system and all above presented systems for robot navigation is the fact that our system does not exist as a single system. All other approaches require a single operating agent: a "supervisor" computer program that manages all data - whether in a global or distributed fashion. Still this one agent has access to all information, arranges all such knowledge in a global context, and can interpret all information with respect to all other information. Here, in contrast, we present a system of distributed agents, each only having access to local information and integrated bits of information originating from other PAs in the network as seen in its own perspective. No agent in our system can take its decisions based on global knowledge; still this distributed network of active local PAs achieves globally consistent behavior.

In a recent issue of Science with a special focus on robotics, one of the contributions (Bellingham and Rajan 2007) outlined full robotic autonomy - i.e. operation without direct human supervision in remote and hostile environments - is becoming a necessity to solve pressing research problems regarding climate in oceans or space; yet we are far from success. Even at much smaller scale, looking at vacuum cleaning robots for households which currently follow simple patterns to collect as much dirt as possible, purposeful autonomous operation drastically increases efficiency. Instead of covering all areas equally, a robot that autonomously creates a representation of its particular home realizes that several places - such as a kitchen - typically are dirtier than others - hence it can spend its time cleaning more useful. Today's technological solutions are by far too complex and thus too expensive - both in required sensor precision and computing power - to become applicable in such scenarios. Our model might open a path towards cheap and simple distributed autonomous navigation systems.

Throughout this thesis we do not want to argue that we can build better maps than SLAM approaches do; especially not in terms of precision and completeness about representing external space. But our system is sufficiently good for navigation between behaviorally relevant places and - due to distributed local nature - computationally much more efficient. We are convinced that we do not need what SLAM is so proud of: a global correct map.

7.5 Conclusions with Respect to Animal Navigation

In the last few decades much attention has been devoted to recordings from so called Place Cells in the Hippocampus of rodents (O'Keefe and Nadel 1978; Muller 1996 ; Moser, Kropff et al. 2008). Such cells seem to code for an animal's spatial position with respect to its current environment. Remapping of place cells' firing pattern (Cressant, Muller et al. 2002; Fyhn, Hafting et al. 2007) seems to occur whenever an animal enters a distinct new environment. We speculate that activity of such place cells correlates to activity of place agents in our model, which each code for a particular place but transfer activity to another agent when the robot changes to a new relevant location.

A recent study by (Foster and Wilson 2006) presented cells in rat's hippocampus that perform a sequential replay of spatial episodes in reverse temporal order. The authors speculate that such a mechanism is involved in forming a spatial representation and in representing places of high reward value, as it is stronger present in new environments compared to familiar environments. Our system - in contrast - uses an analogy to such reverse reply for finding routes to targets (chapter 5.3.1).

However, all such experiments have been constrained to relatively small areas, significantly below the size of a wild animal's foraging area; so much about how brains represent large discrete environments stays speculative.

Looking at the behavioral level, we find various experiments that describe rats solving spatial tasks. Among the most recognized are those by (Tolman 1948) and (Morris 1981), in which they train rats to return to behaviorally relevant places. We will look at an experiment from (Tolman and Honzik 1930) in more detail and compare how our distributed system acts in such an experiment.

Tolman trained rats to explore a three arm maze as shown in Figure 81. Rats learned three different paths to a food source: a direct path (a), a path with a small detour (b) and third path with a substantial detour (c). They were free to explore the rest of the maze during each of the trainings session, but the entrances of the two alternative paths were blocked during the respective session for a path. During training periods all rats were well fed, such that they mainly ignored the existing feeding side and did not spend an exhaustive amount of time there.

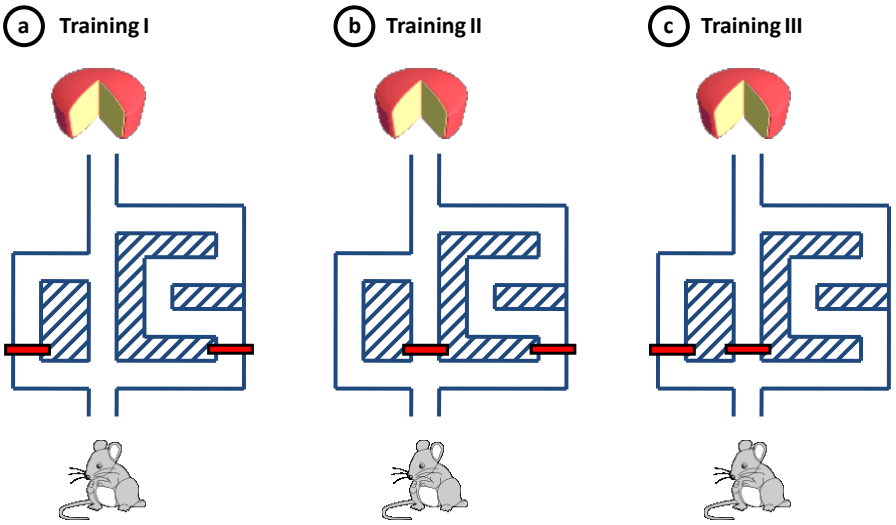


Figure 81: A three arm maze as used by Tolman and Honzik in 1930. Red blocks denote obstructed space. While rats were allowed to explore a maze freely during each training session (a-c), only a single path lead to a feeding place. During testing (d), an obstruction was placed outside the rat's initial field-of-view, closing the two more favorable paths to the feeding side.

After training, Tolman tested the trained - but food-deprived - rats in the same maze with only a single obstruction as shown in Figure 82, d. The obstruction is positioned at a new place at which the rat has never encountered it before, but placed such that it is initially not visible for the rat. All rats headed straight off along the direct path towards food, only noticing that the path was obstructed when arriving at the block. They needed to re-plan their route to food, starting from their current position as shown in Figure 82 e):

- Following the red arrow enters a path in a direction that - during exploration - has never taken led to food. If rats prefer this option, we can conclude that they have no understanding of the global spatial arrangement in the maze.
- Following a path along the yellow arrow - returning to the start and testing the second most favorable path to food - allows us to conclude that the rat has not realized that the new block also renders the second most preferable path useless.
- Upon following the green arrow immediately - which is usually the least preferable choice towards food - the rat demonstrates its global understanding of the maze: only the longest path successfully leads to food.

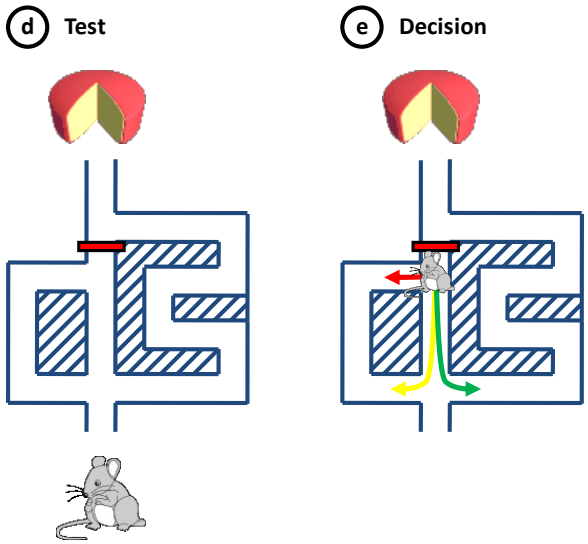


Figure 82: d) Test scenario for the rat. The red block is invisible from the rat's starting place. e) Arriving at the obstruction, the rat has to decide which new direction to take to get to food. Rats in Tolman's experiments always (14/15) decided to follow the green arrow directly.

The rats in Tolman's test almost all directly followed the green path, instead of trying the yellow path first - thus indicating that they have an understanding of the global spatial arrangement.

We duplicated this experiment in a robotic version, in which our robot explored an arena of size 5x4m, obstructed with cardboard boxes that represent Tolman's maze. Figure 83 a) shows the robot at its starting place in a side-view photograph, whereas panel b) presents the view from an overhead camera used to track the robot. The trajectory taken during exploration is superimposed in blue. Note that we do not implement three gates during training, but instead allow the robot to explore the whole maze at once. Panel c) in Figure 83 shows the resulting distributed network of place agents that a nucleus PA built without having a prior description of the underlying space. Note that we did not use landmarks in the maze to resemble a uniform maze as used in Tolman's experiments.

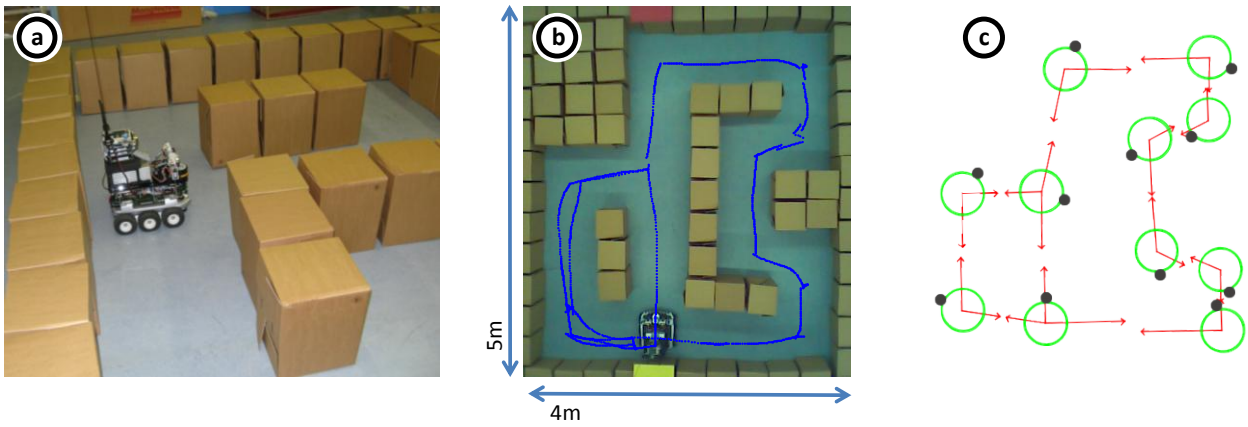


Figure 83: A robotic experiment to duplicate Tolman's findings in rats a) side-view of robot at starting position b) start (yellow), target (red), trajectory during exploration as recorded by an overhead ceiling camera c) Resulting network

When testing our robot in the same scenario as Tolman's rats (Figure 84, a), we find that the network of place agents also initially guides the robot along the direct path towards the target, as the place agents still are unaware of the introduced obstruction close to the target. However, when

the robot fails to reach the target along the direct path on the last segment, the two PAs involved in this last segment temporarily disable their connection, as shown in Figure 84, b) by a black X. The system needs to re-establish a gradient towards the target (chapter 5.3.1). Starting new tokens or invitations (chapter 5.1.2) from the current PA sets up a gradient in the system that leads the robot immediately along the "green" path shown in Figure 82 e - the path that initially is least favorable for the system.

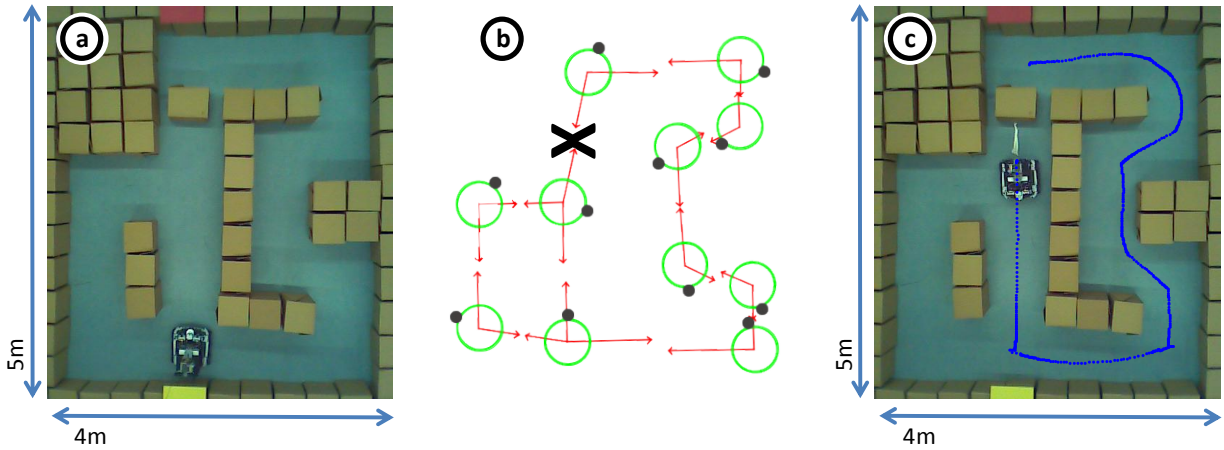


Figure 84: Testing the network of distributed PAs in a maze task identical to Tolman's test for rats. a) shows the top view of the maze with an additional cardboard box as obstacle close to target (red at top of image). b) upon the robot arriving at the obstacles, the two PAs involved temporarily disable their connection as they do not succeed in guiding the robot. c) the PAs send the robot along the correct path after re-establishing a gradient in the network that resembles the updated knowledge.

Performing this experiment we can demonstrate that our system of distributed place agents - which only act on their local knowledge - performs a task that previously was thought of to require a global view on a spatial representation. The experiment as performed by Tolman is the classic example to argue that an abstract globally arranged representation of the environment must exist in the animal's brain; otherwise it would not realize that only the longest path leads to the feeding place. We demonstrate that we can solve such a task using only distributed local knowledge, without a global supervisor inspecting all spatial knowledge.

7.6 Final Conclusions

The system outlined in this thesis implements a new approach to perceiving the world: instead of interpreting spatial information as a large globally consistent map, we perceive the world as a patchwork of independent behaviorally significant spots that - one at a time - contribute to global behavior.

Starting with zero knowledge, a nucleus node builds up a distributed network of behaviorally relevant nodes while exploring the world. These nodes are active processes, producing offspring for exploration and taking independent decisions based on limited local information. Using such an approach keeps our system computationally tractable in real-time on current hardware.

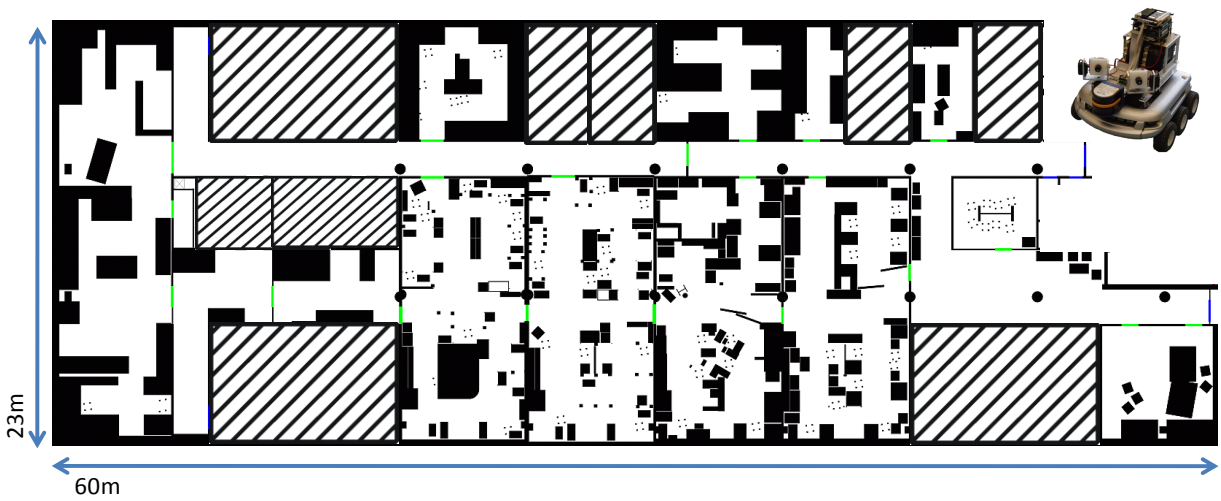
In contrast to traditional methods for representing spatial information, a unique advantage of our system is that the network of patches does not need to be globally consistent. Because all decisions

are taken on local information, local consistency, e.g. in angles and distances, is sufficient. When the mobile robot returns to a previously visited place and closes a loop, traditional systems have to spend much computational effort in correcting acquired information for global consistency. In our system, in contrast, the loop is represented only implicitly, as all nodes only know about their direct neighbors.

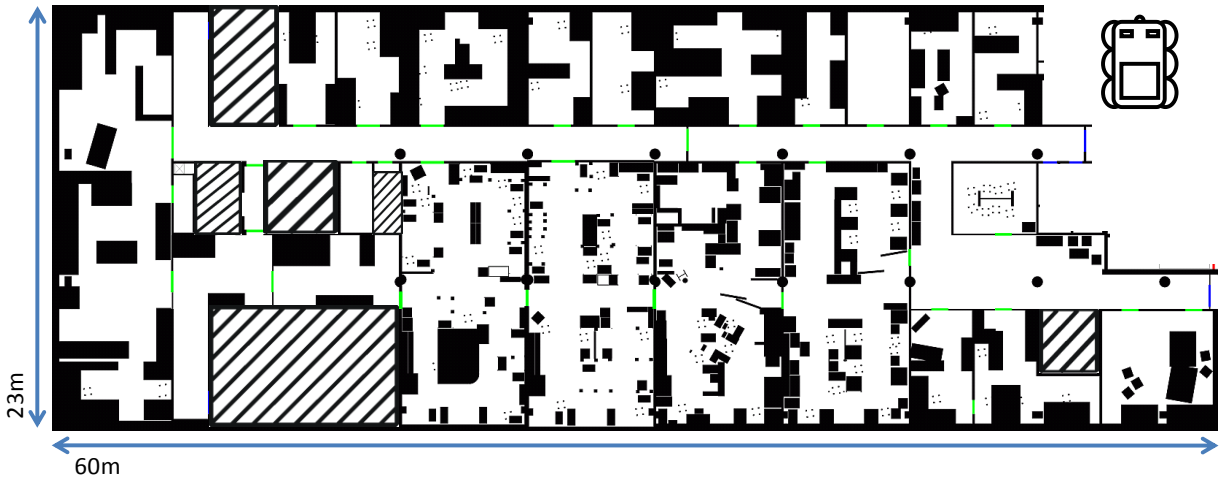
We have presented a compulsive framework that links a mobile agent's spatial actions, cognition and behavior in a distributed system. The system does not require a global supervisor to exhibit globally consistent behavior.

Appendix A: Results

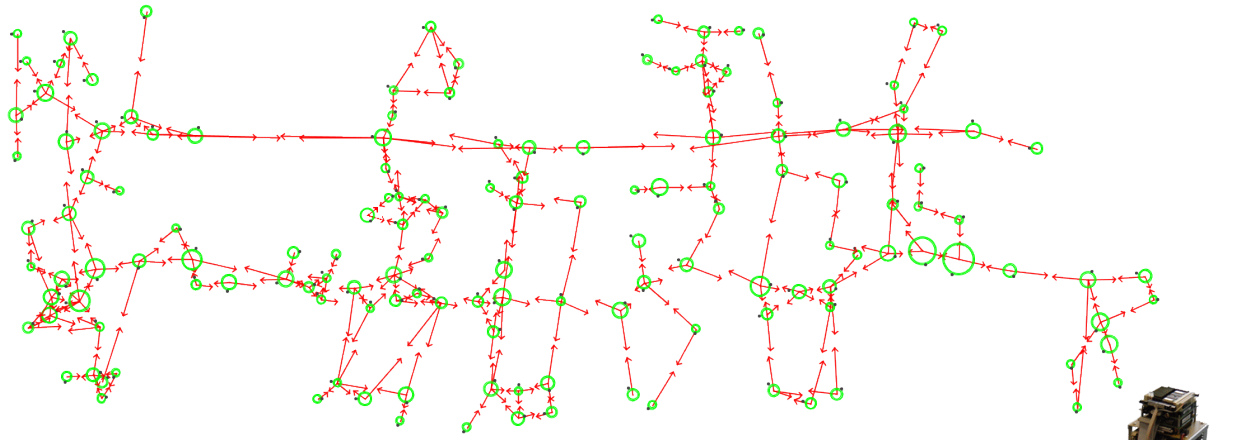
The following pages show exploration results of 15 independent experiments, in each of which our system guided a mobile robot in reality (5x) or in simulation (10x) to explore an initially unknown environment: one whole floor of the Institute of Neuroinformatics. The two sketches below show the underlying floor plans. The top figure depicts our best knowledge about the true spatial arrangement in the institute. Note that we have carefully characterized the middle four rooms, but the interior of all other rooms only coarsely reflect true space.

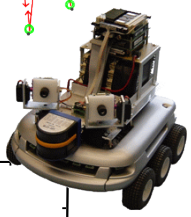


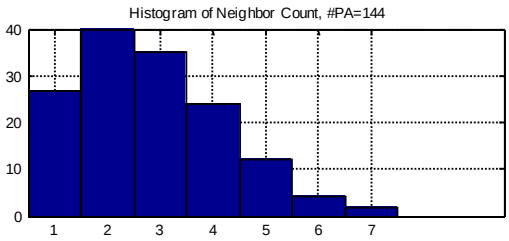
The following figure shows the simulated environment. To allow comparisons we chose the same layout, but "virtually opened" several of the blocked rooms. The resulting networks from simulation thus on average show a higher number of PAs representing space.

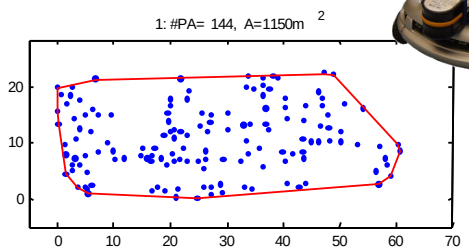


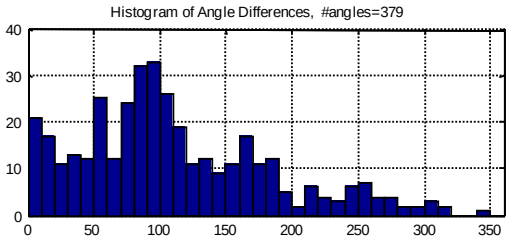
Exploration 1 (real robot)

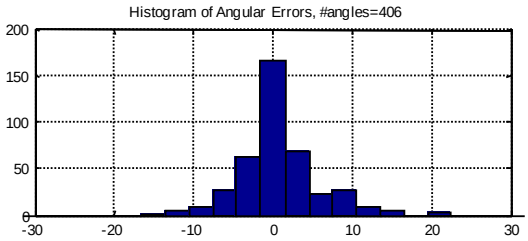


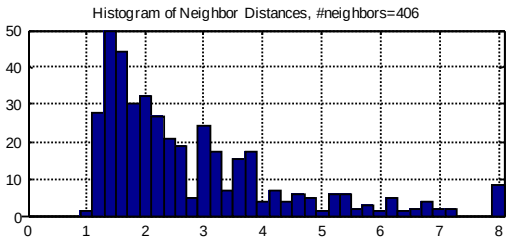


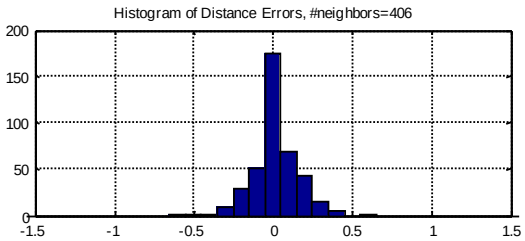


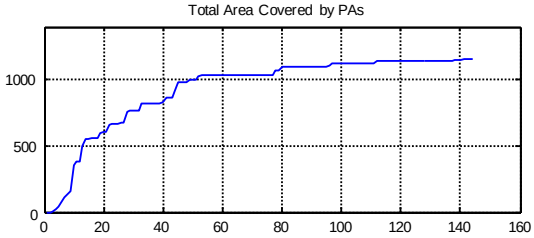


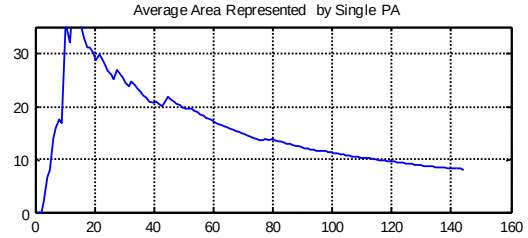




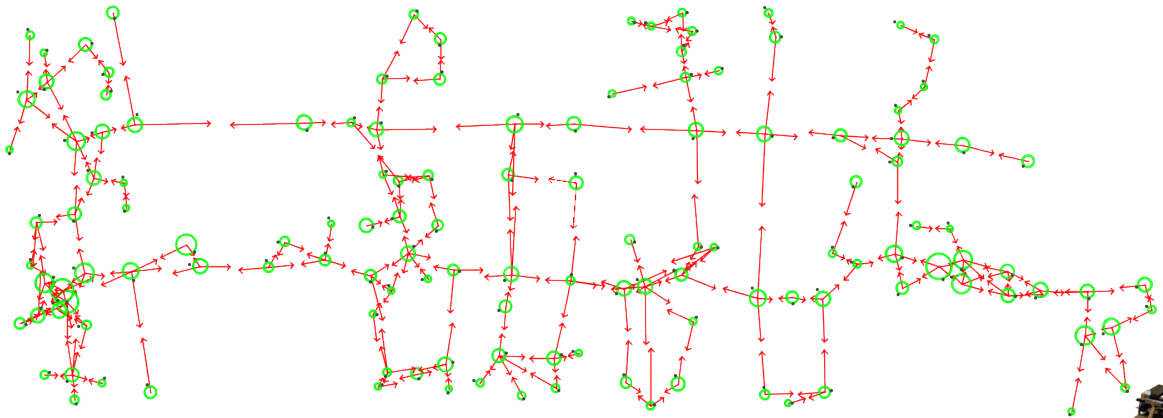


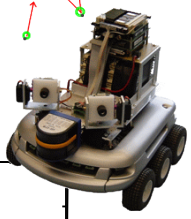


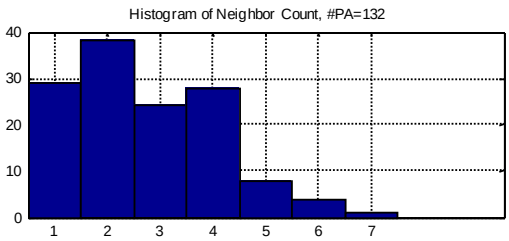


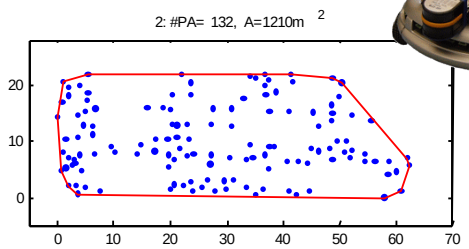


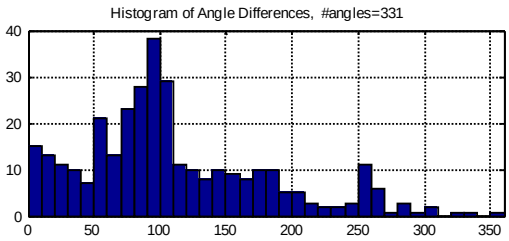
Exploration 2 (real robot)

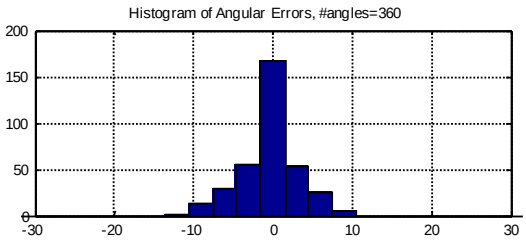


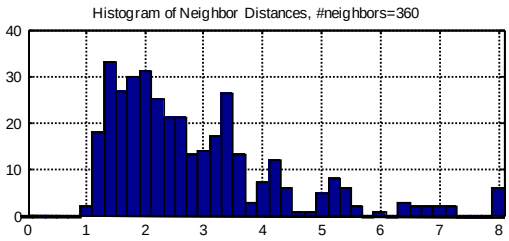


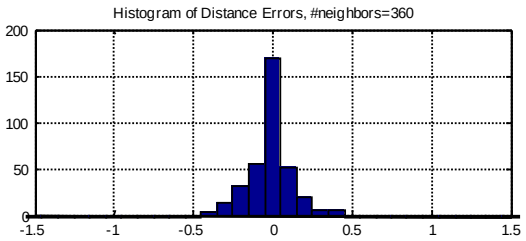


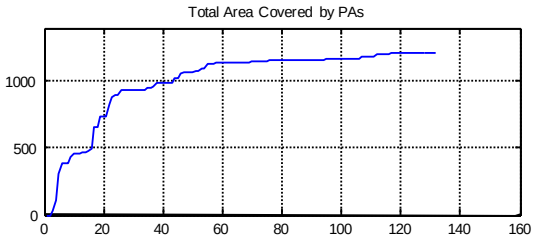


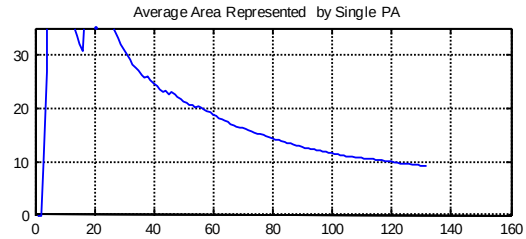




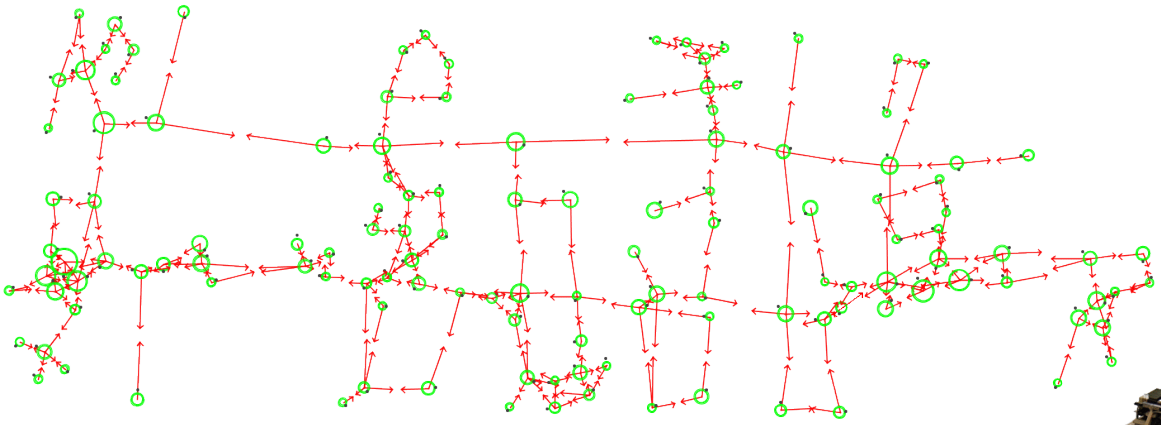


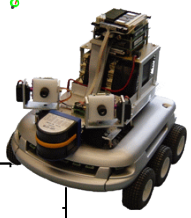


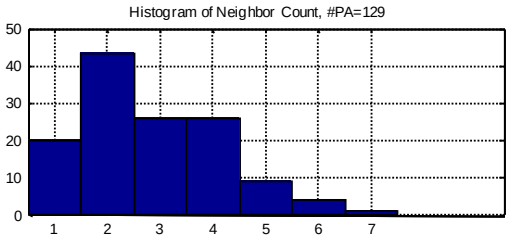


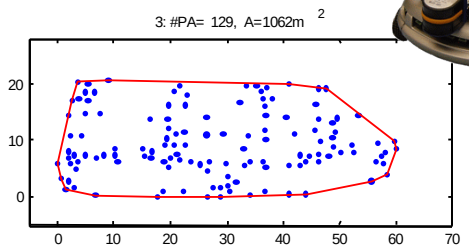


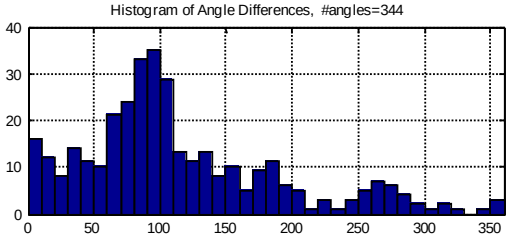
Exploration 3 (real robot)

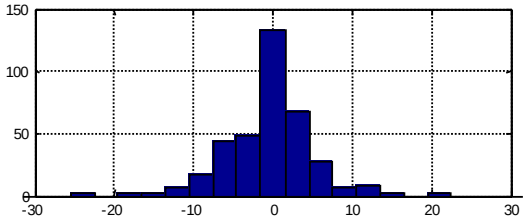


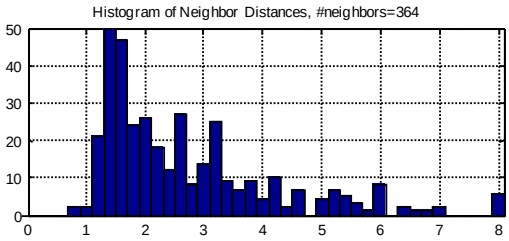


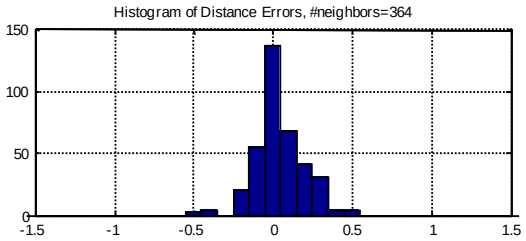


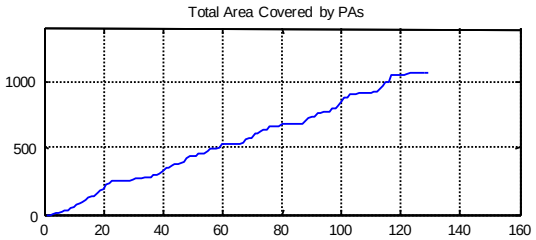


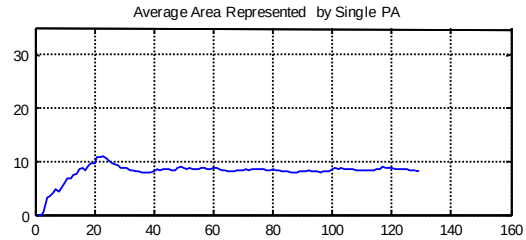




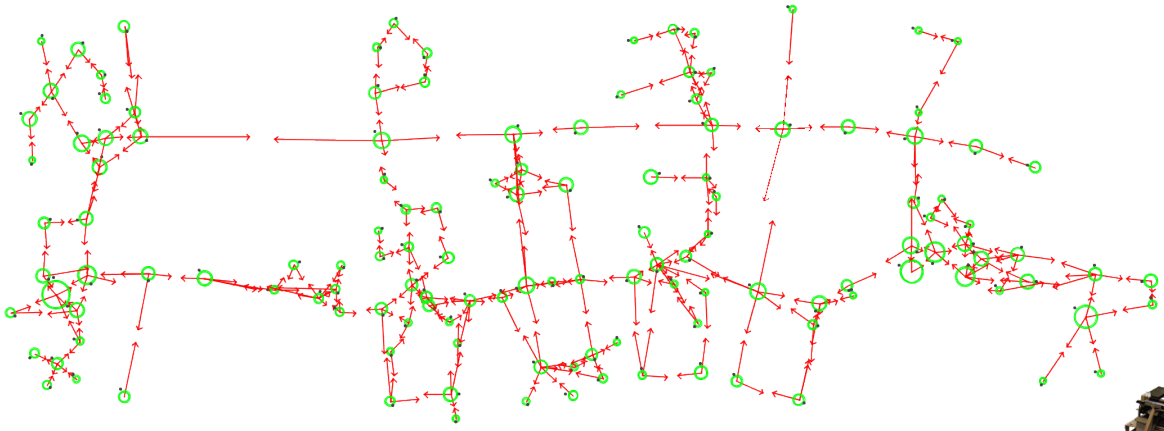


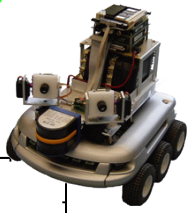


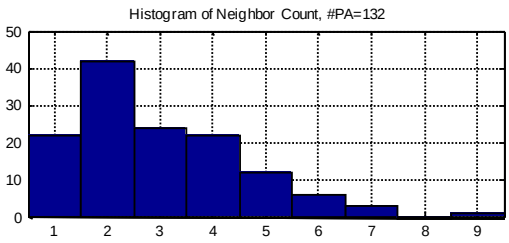


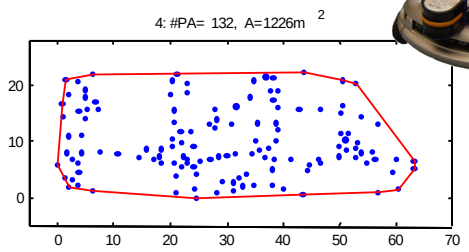


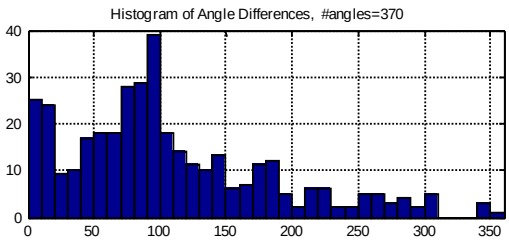
Exploration 4 (real robot)

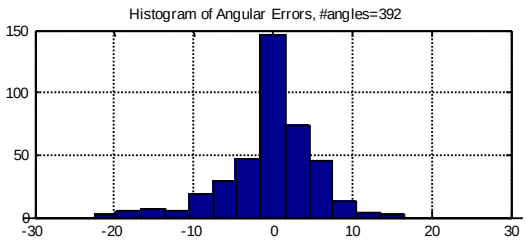


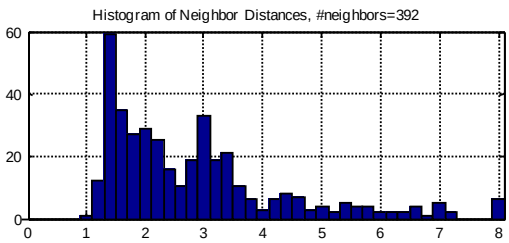


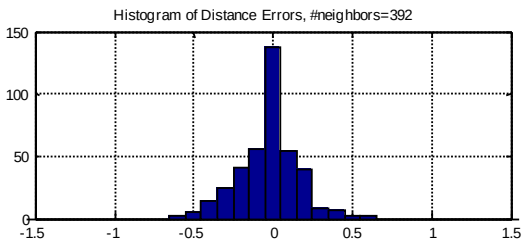


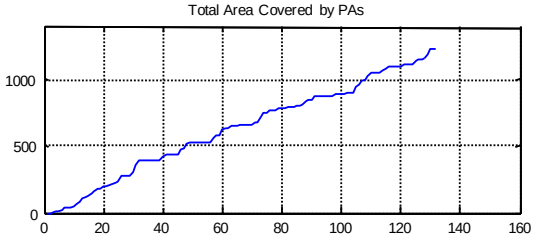


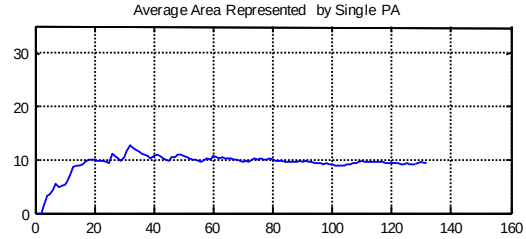




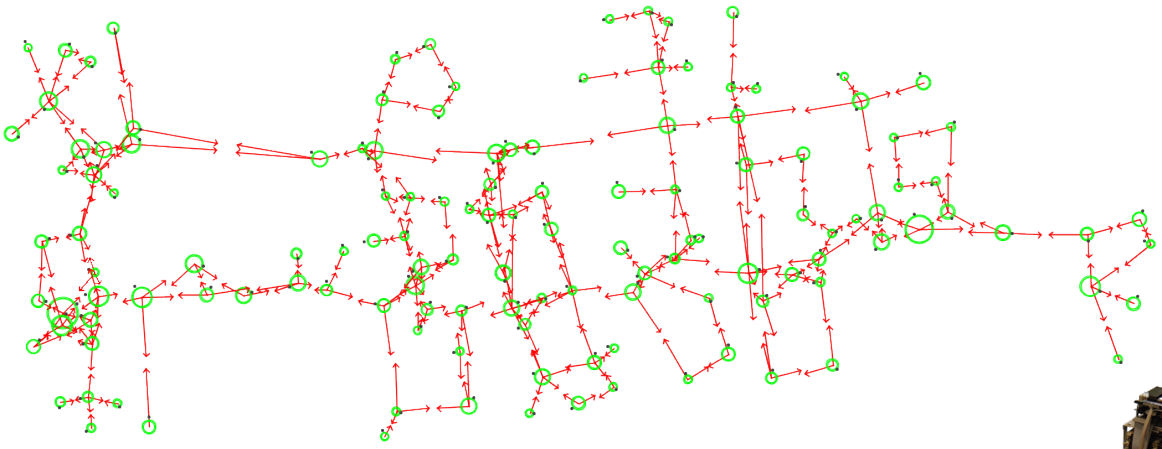


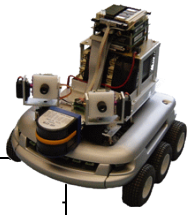


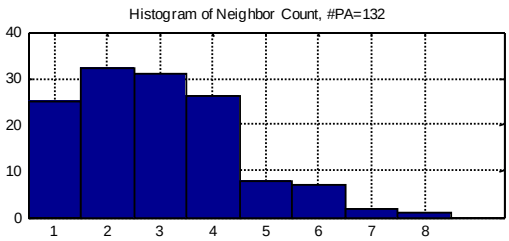


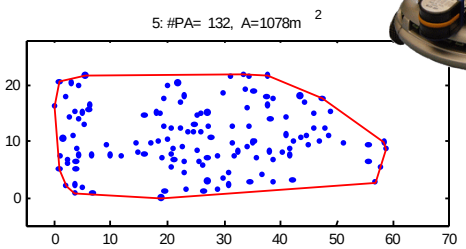


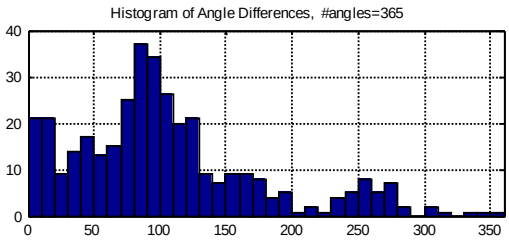
Exploration 5 (real robot)

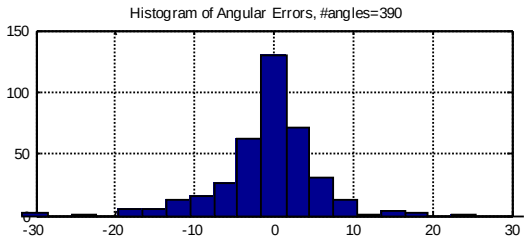


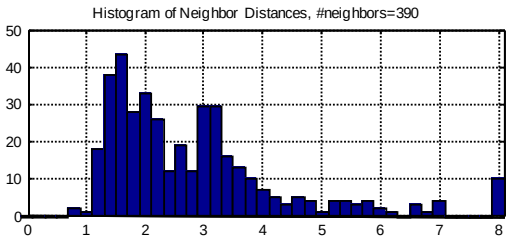


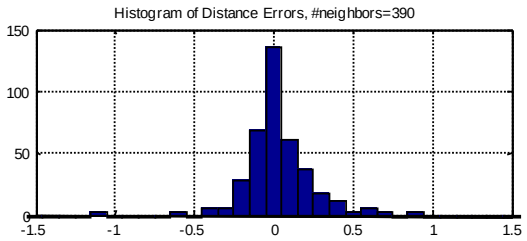


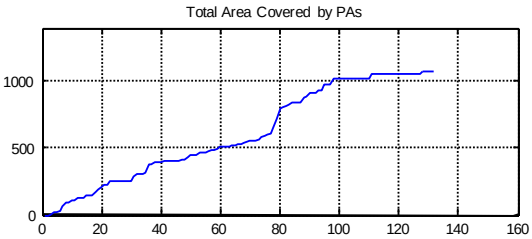


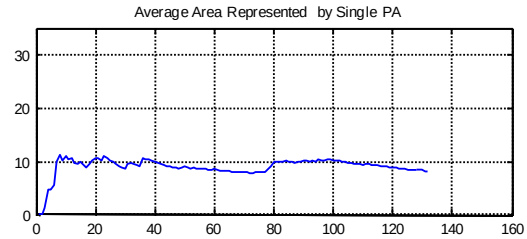




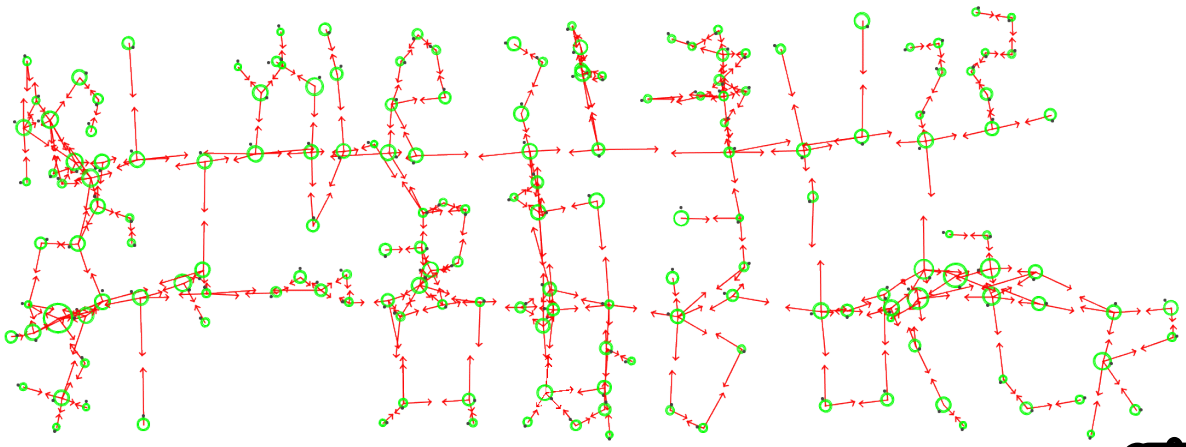


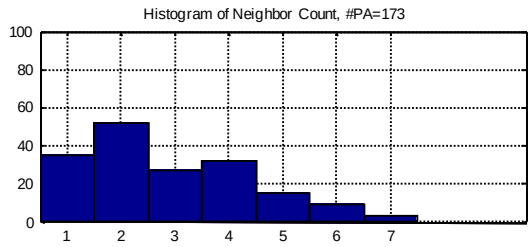


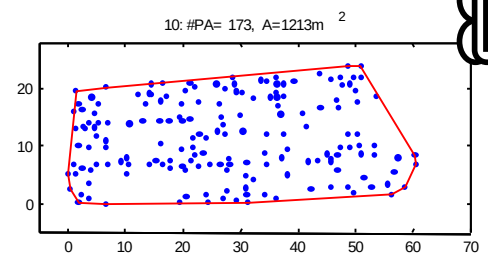


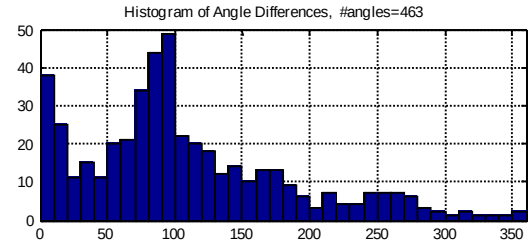


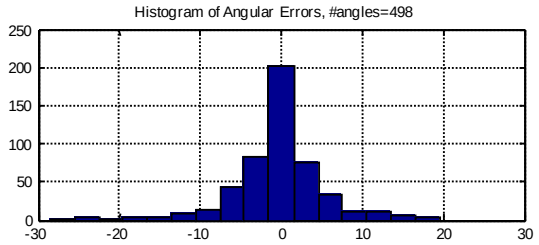
Exploration 1 (simulated)

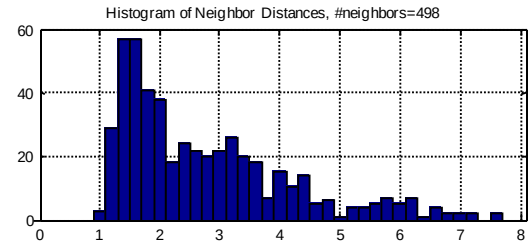


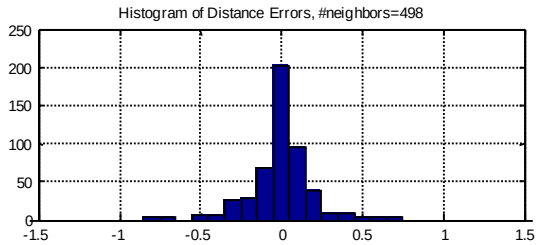


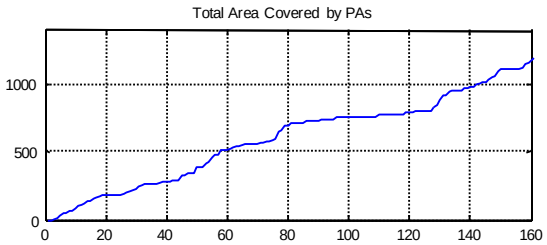


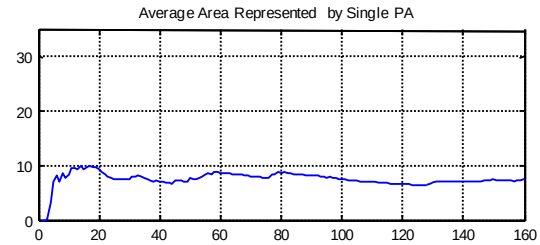




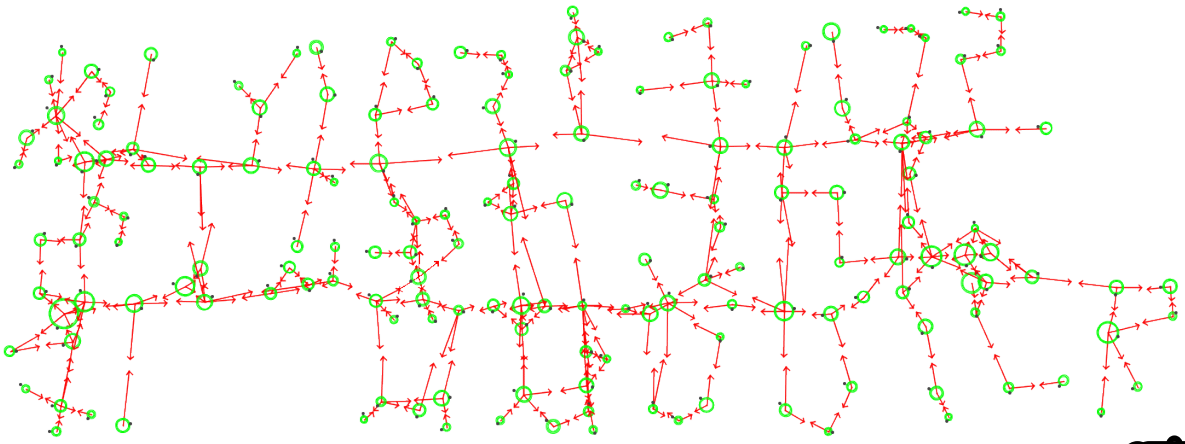


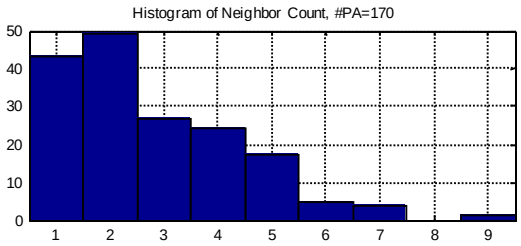


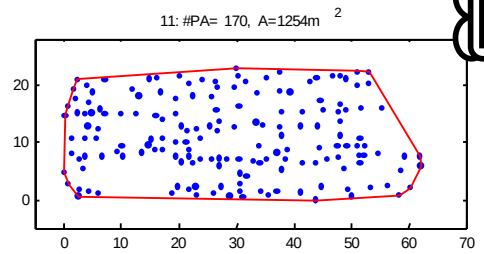


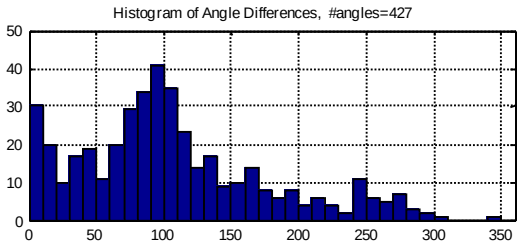


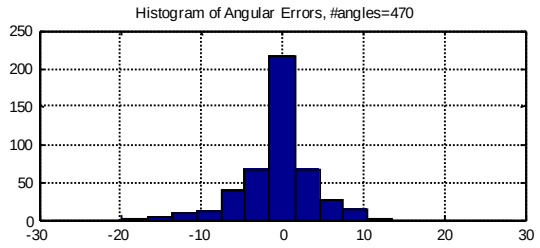
Exploration 2 (simulated)

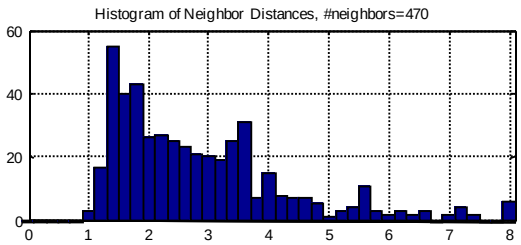


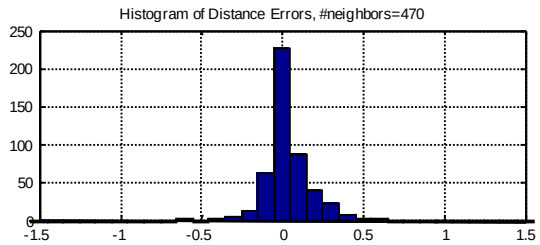


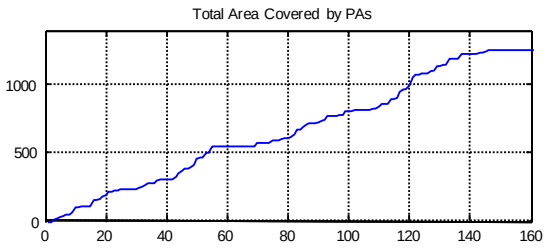


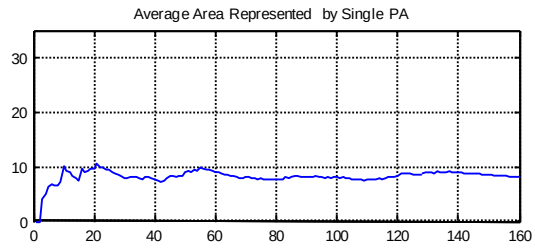




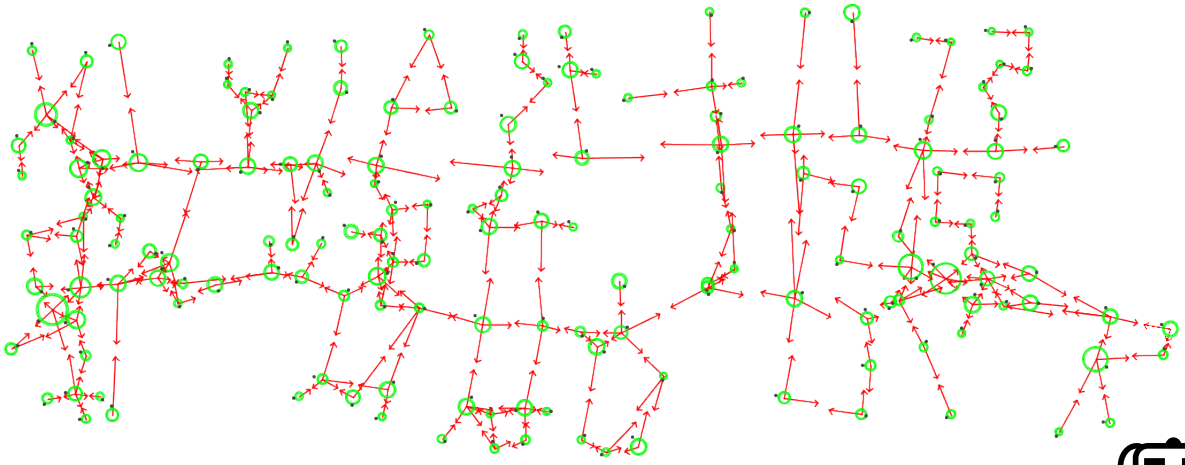


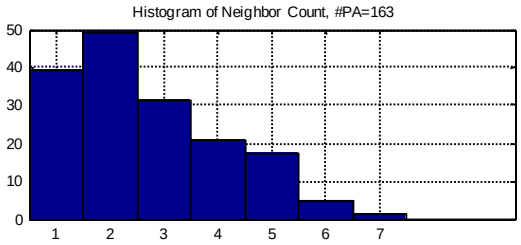


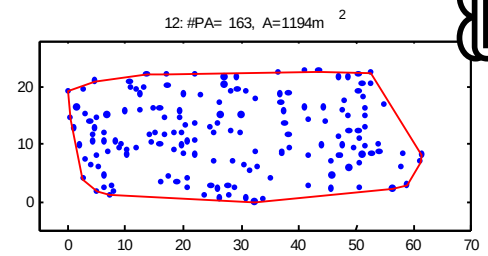


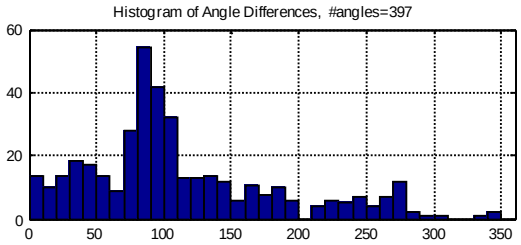


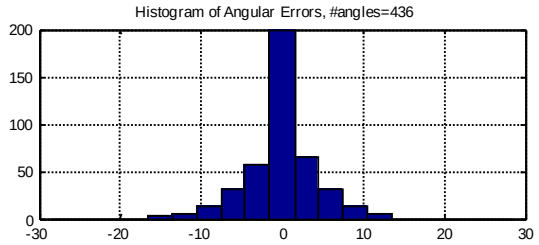
Exploration 3 (simulated)

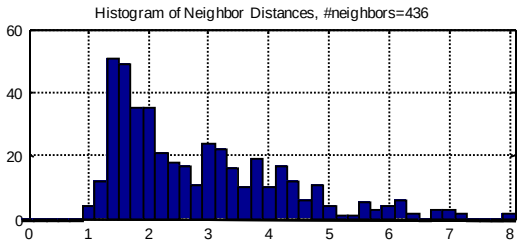


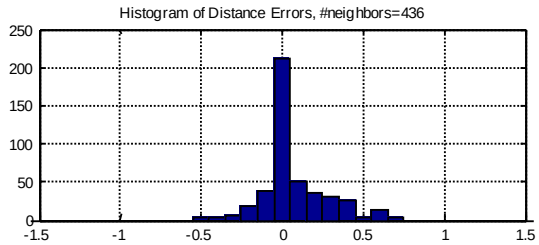


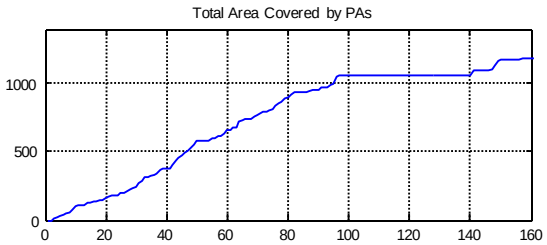


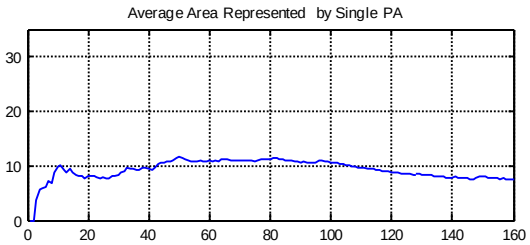




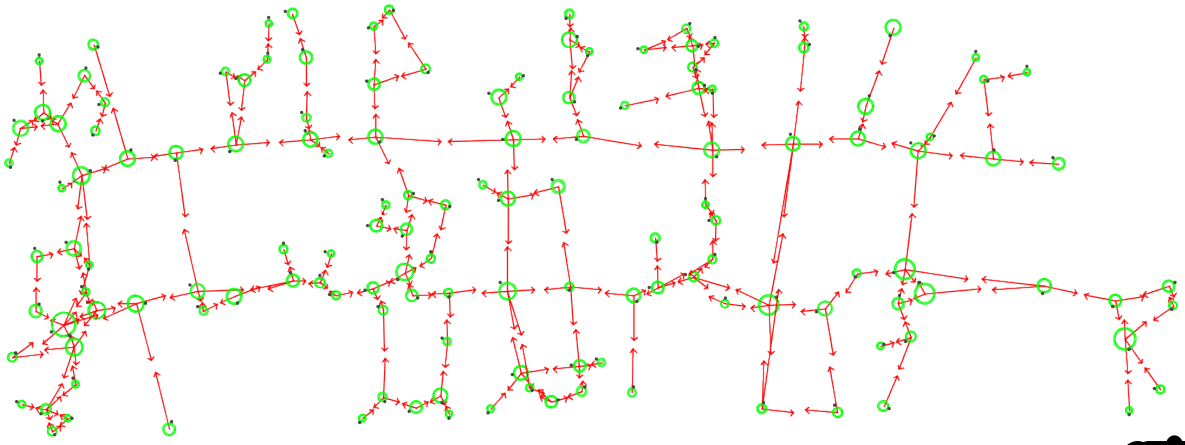


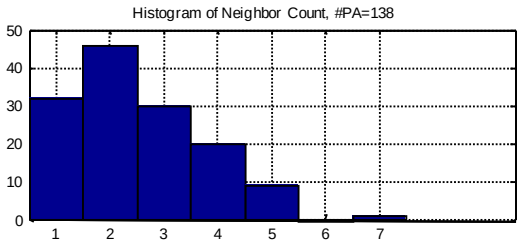


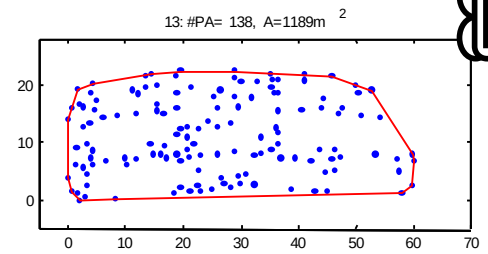


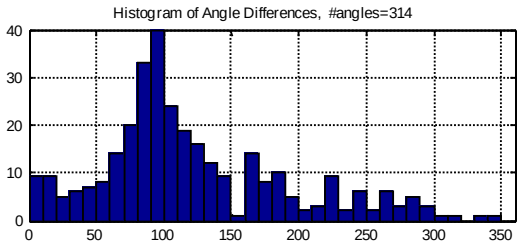


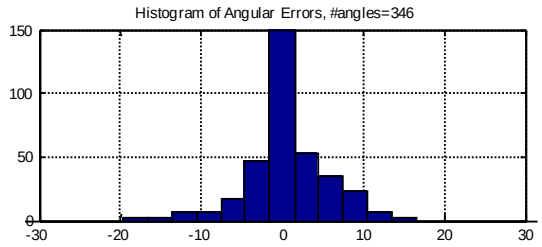
Exploration 4 (simulated)

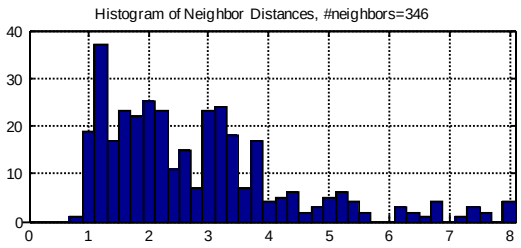


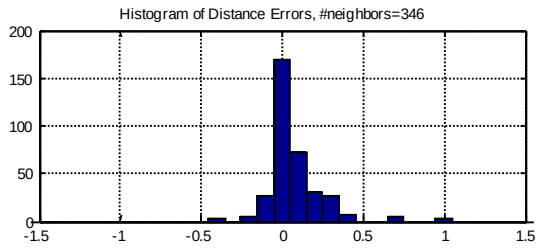

Histogram of Neighbor Count, #PA=138
50
40
30
20
10
0
1
2
3
4
5
6
7

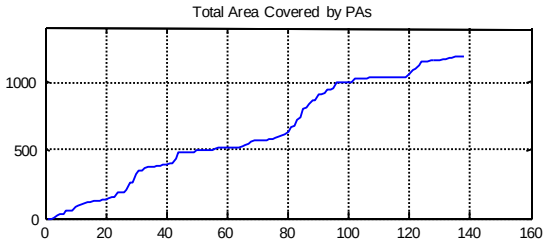

13: #PA= 138, A=1189m 2
20
10
0
0
10
20
30
40
50
60
70

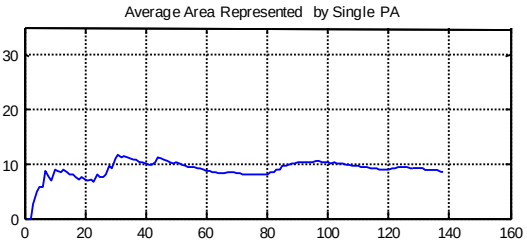

Histogram of Angle Differences, #angles=314
40
30
20
10
0
0
50
100
150
200
250
300
350


Histogram of Angular Errors, #angles=346
150
100
50
0
-30
-20
-10
0
10
20
30

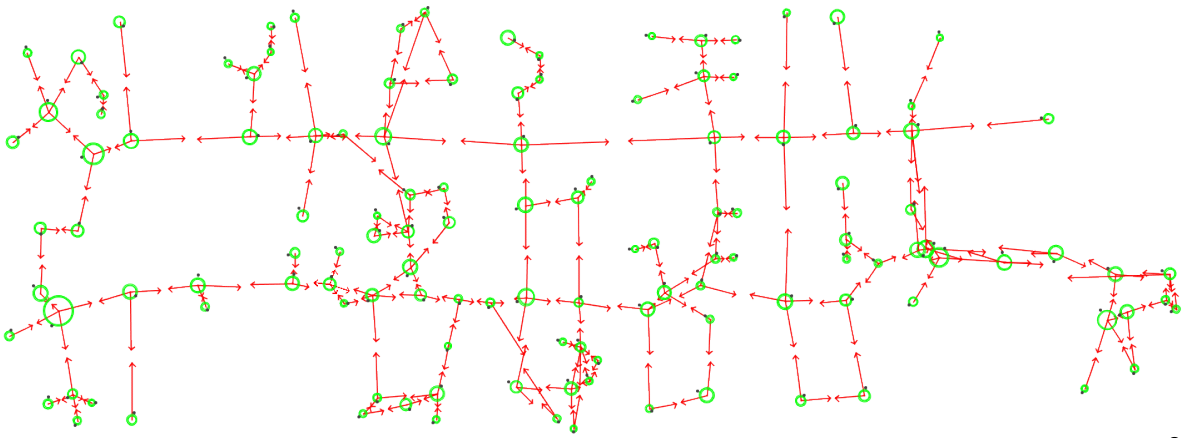

Histogram of Neighbor Distances, #neighbors=346
40
30
20
10
0
0
1
2
3
4
5
6
7
8

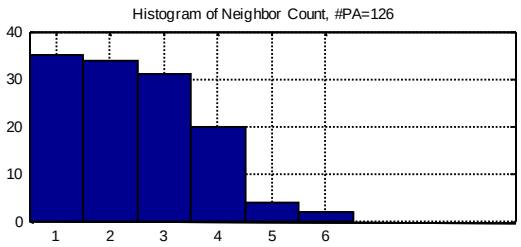

Histogram of Distance Errors, #neighbors=346
200
150
100
50
0
-1.5
-1
-0.5
0
0.5
1
1.5

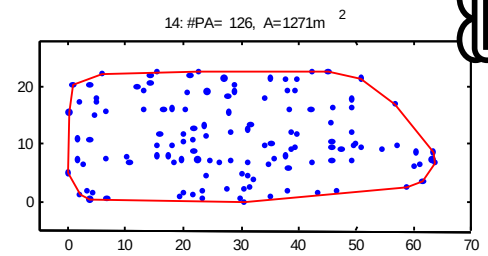

Total Area Covered by PAs
1000
500
0
0
20
40
60
80
100
120
140
160

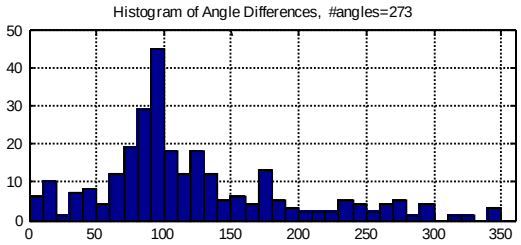

Average Area Represented by Single PA
30
20
10
0
0
20
40
60
80
100
120
140
160

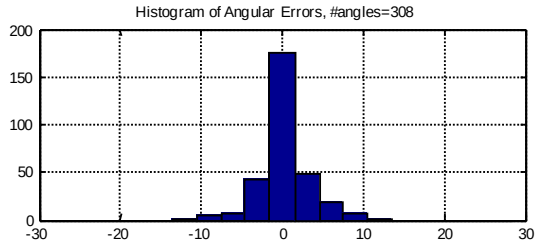
Exploration 5 (simulated)

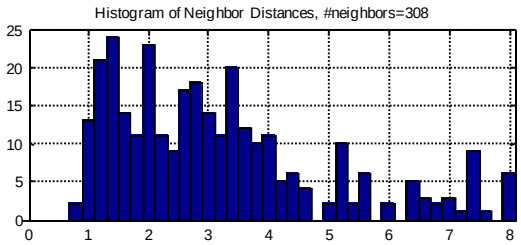


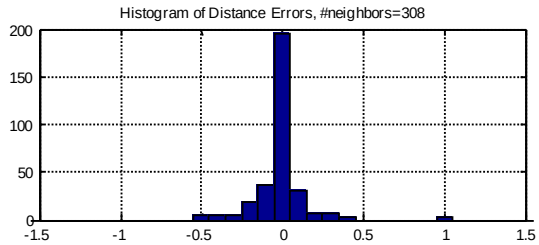


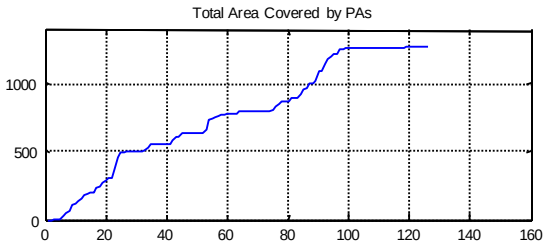


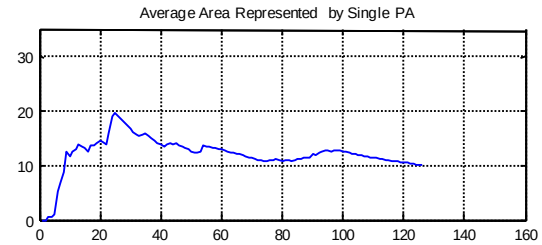




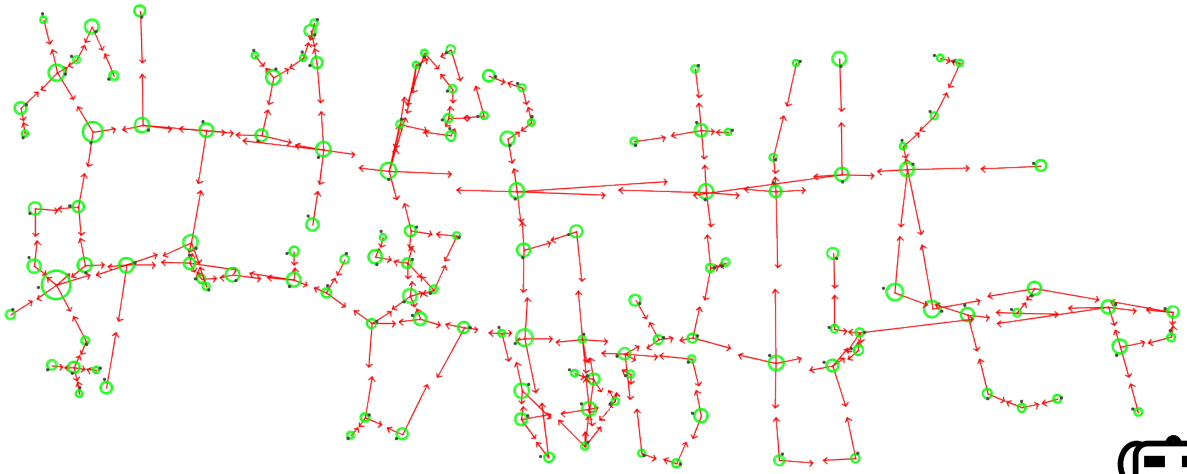


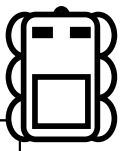


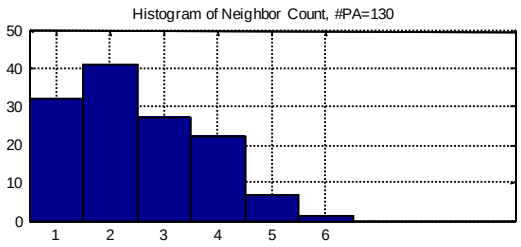


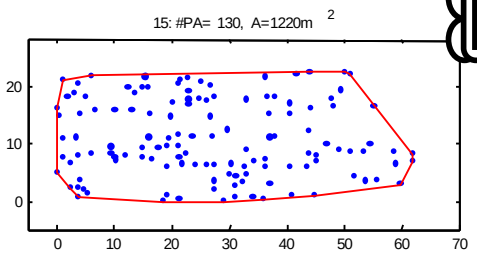


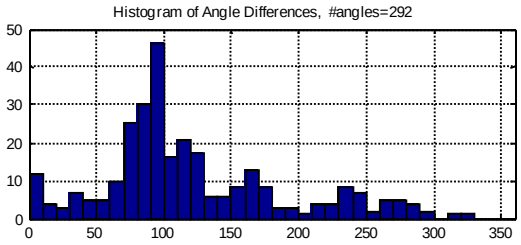
Exploration 6 (simulated)

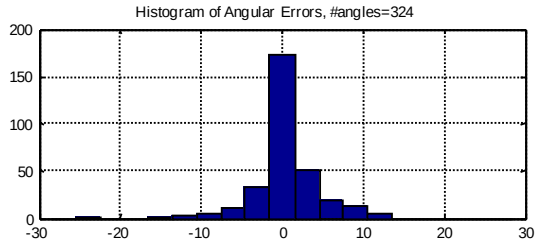


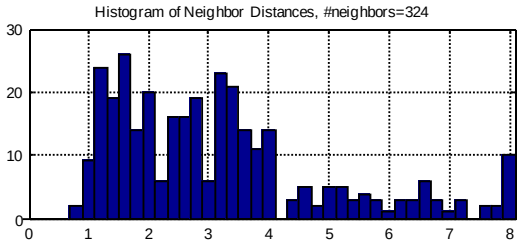


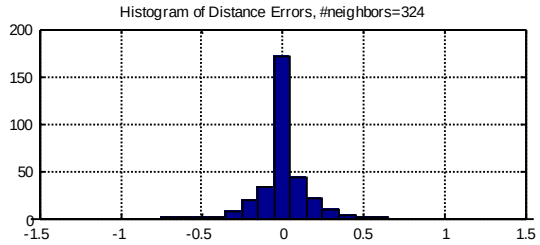


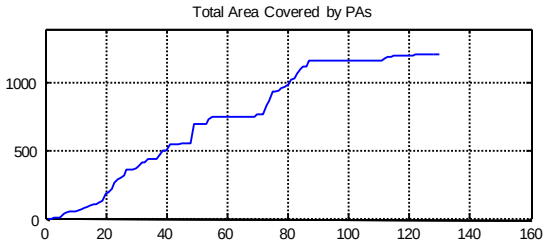


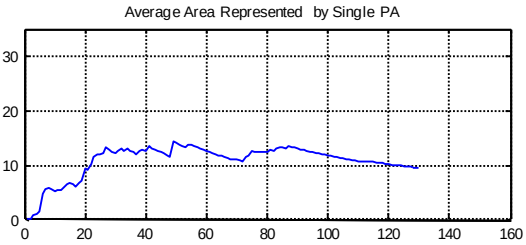




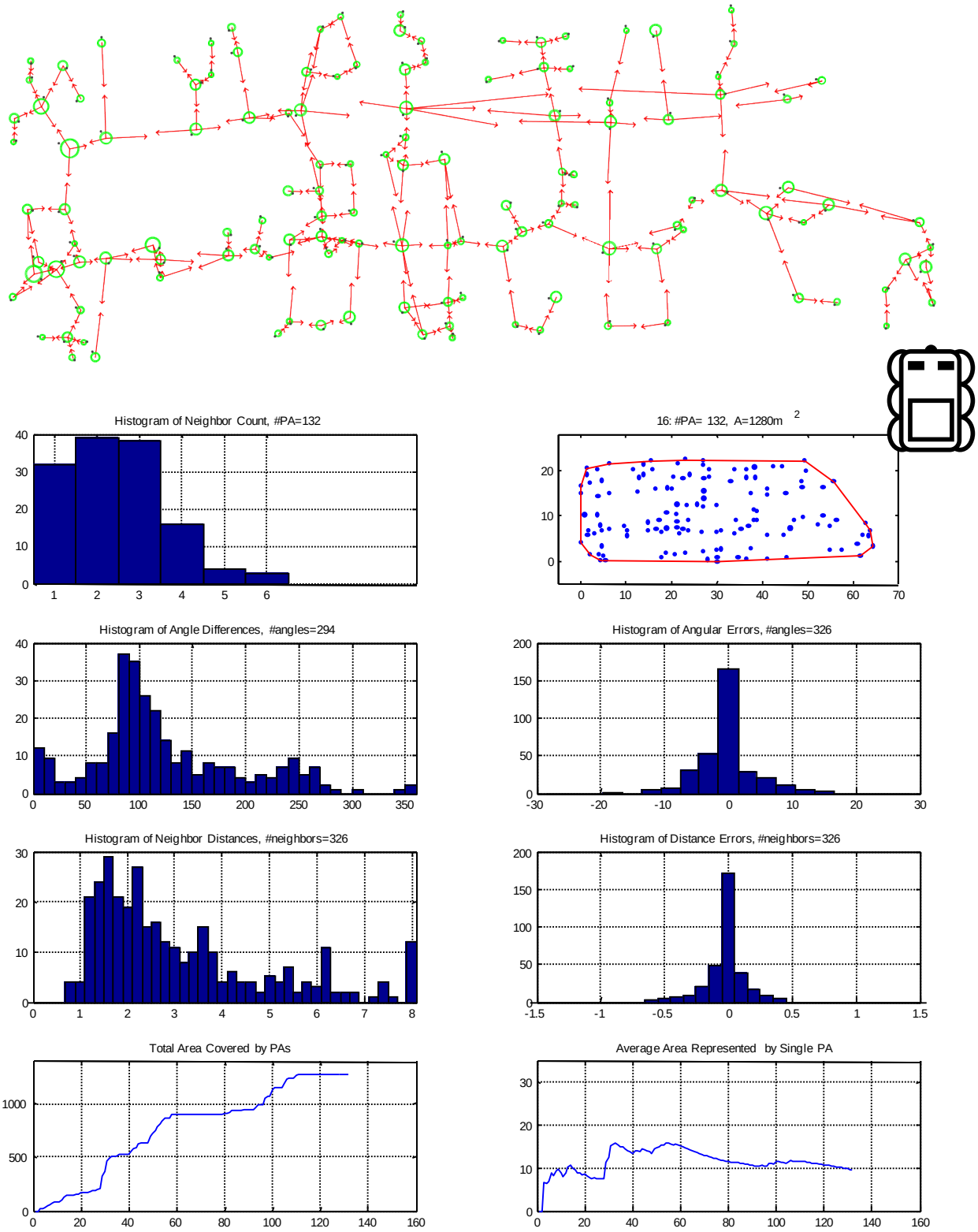




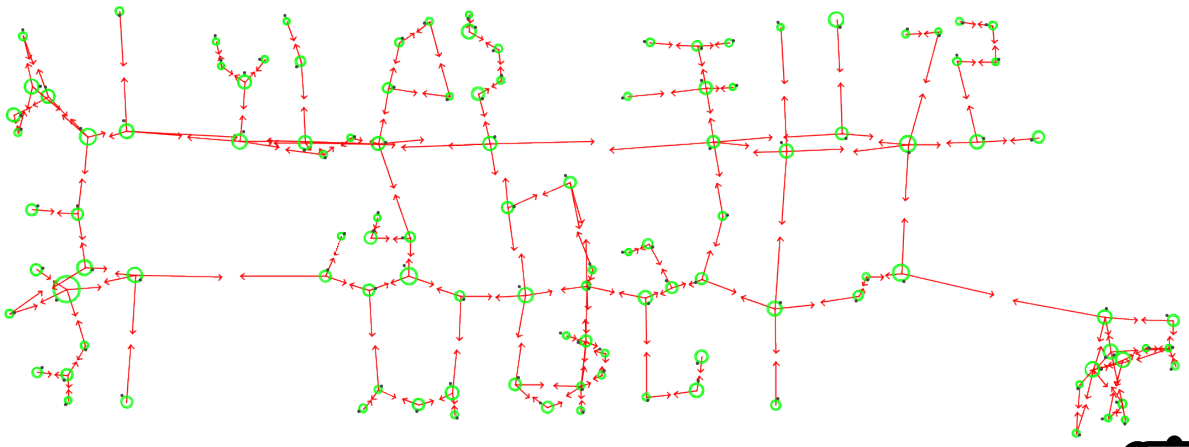


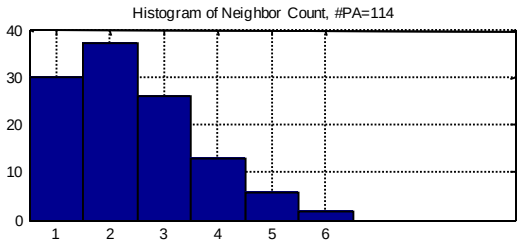


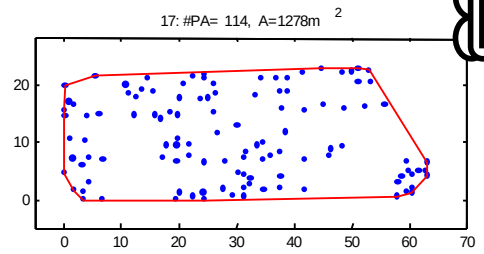
Exploration 7 (simulated)

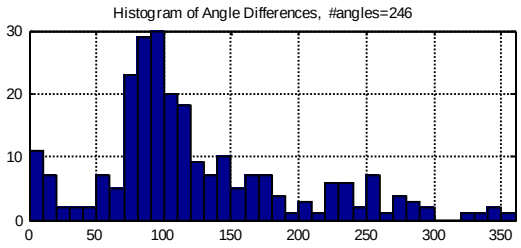


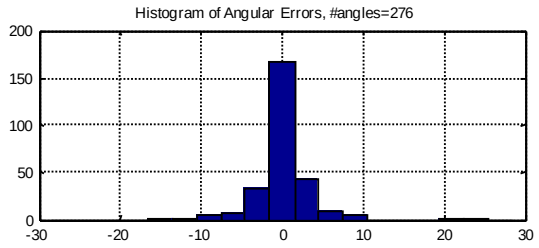
Exploration 8 (simulated)

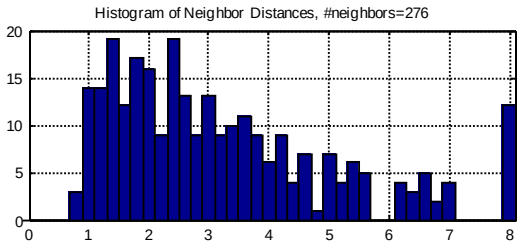


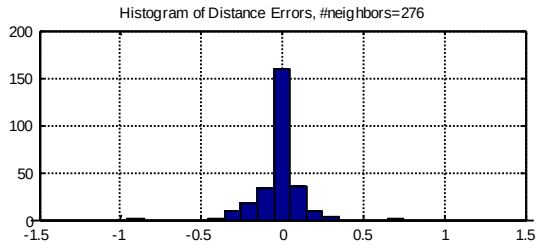


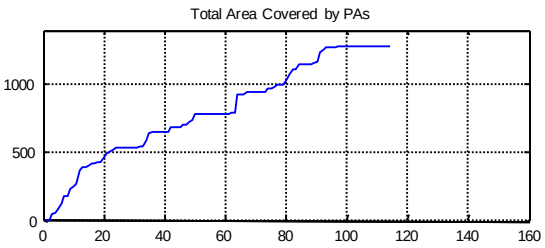


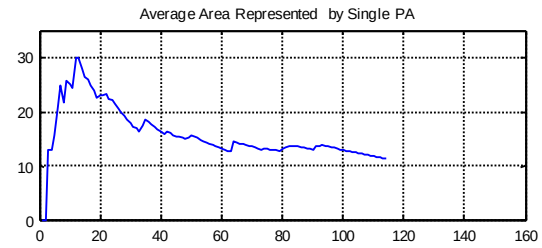




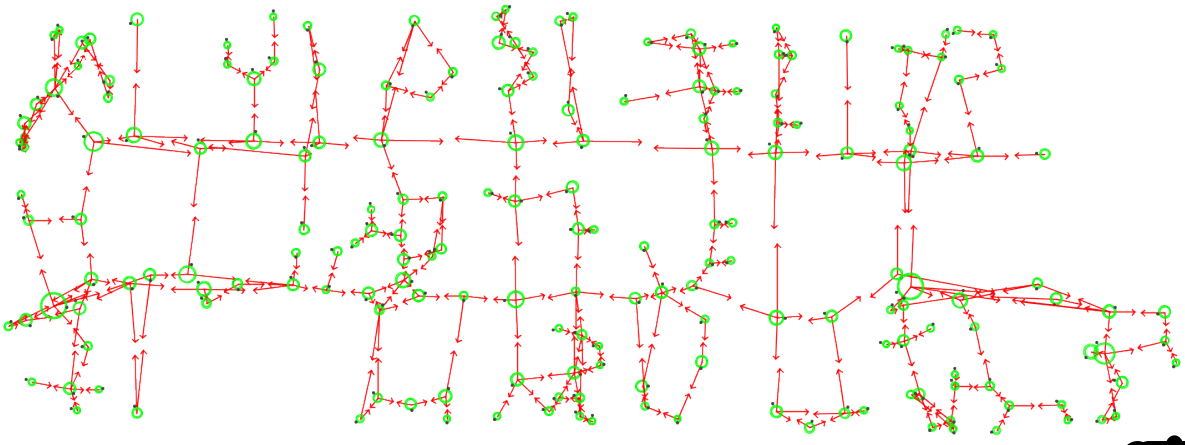


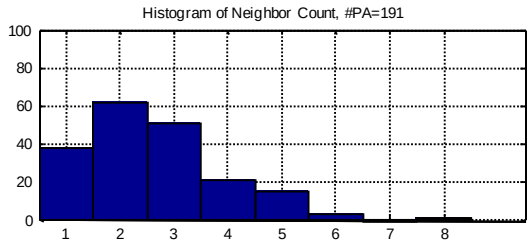


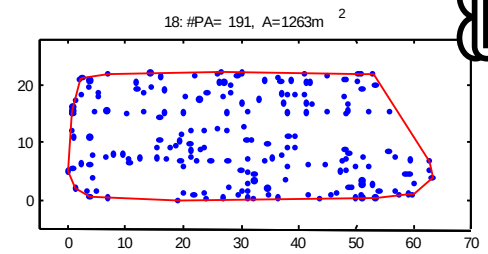


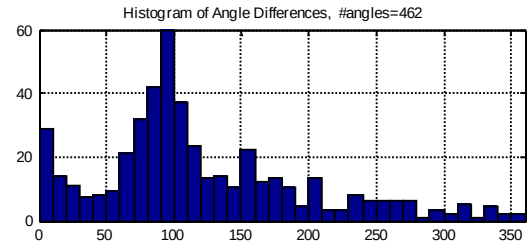


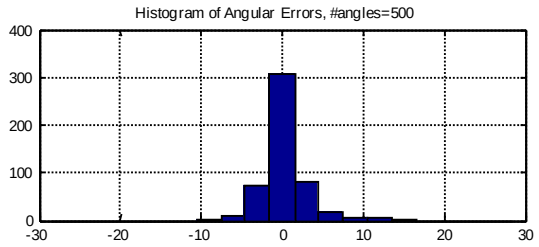
Exploration 9 (simulated)

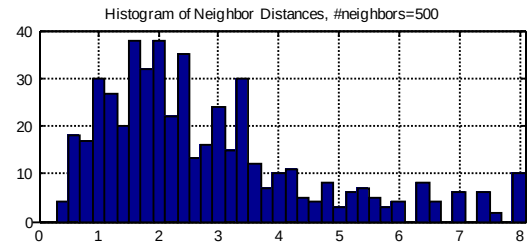


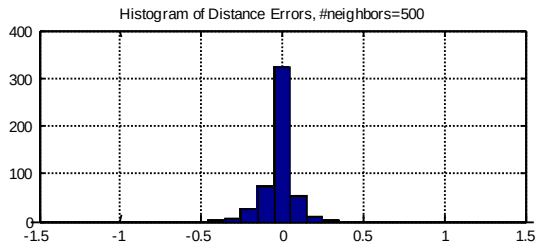

Histogram of Neighbor Count, #PA=191
100
80
60
40
20
0
1
2
3
4
5
6
7
8

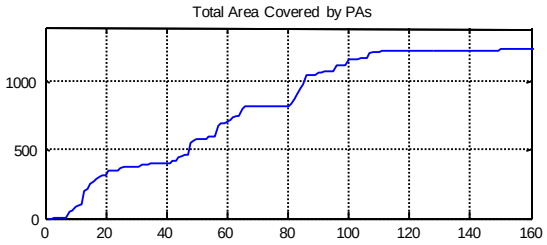

18: #PA= 191, A=1263m 2
20
10
0
0
10
20
30
40
50
60
70

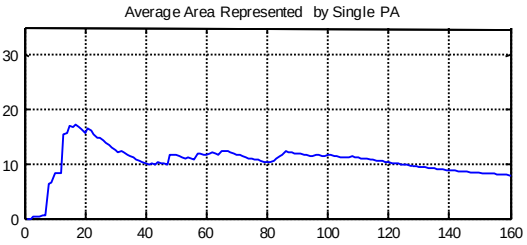

Histogram of Angle Differences, #angles=462
60
40
20
0
0
50
100
150
200
250
300
350


Histogram of Angular Errors, #angles=500
400
300
200
100
0
-30
-20
-10
0
10
20
30

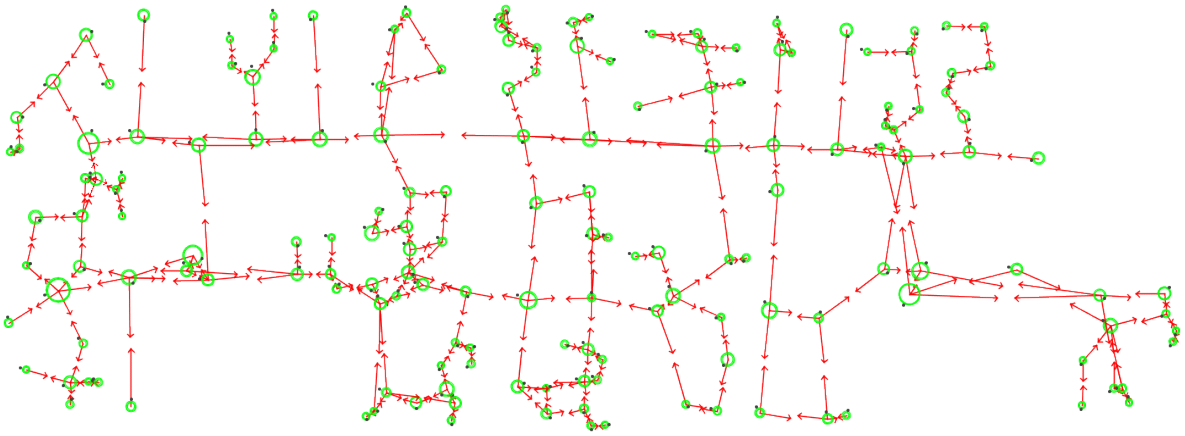

Histogram of Neighbor Distances, #neighbors=500
40
30
20
10
0
0
1
2
3
4
5
6
7
8

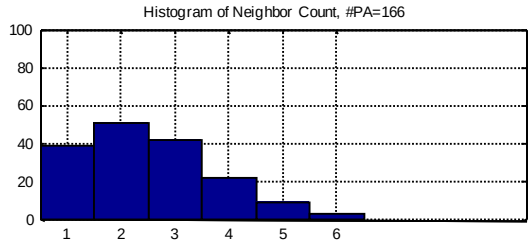

Histogram of Distance Errors, #neighbors=500
400
300
200
100
0
-1.5
-1
-0.5
0
0.5
1
1.5

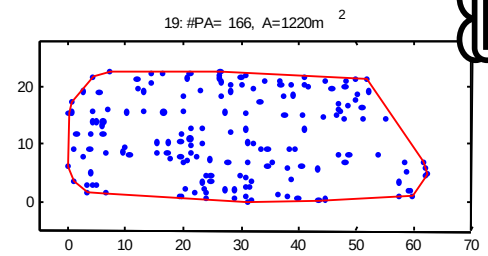

Total Area Covered by PAs
1000
500
0
0
20
40
60
80
100
120
140
160

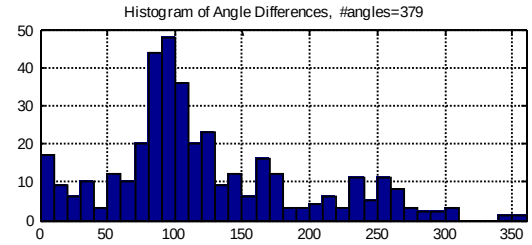

Average Area Represented by Single PA
30
20
10
0
0
20
40
60
80
100
120
140
160

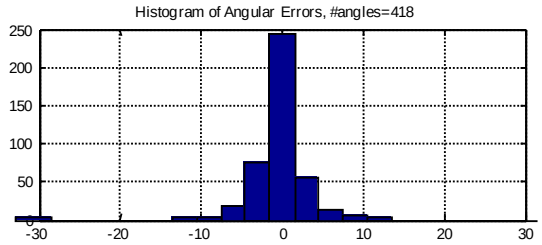
Exploration 10 (simulated)

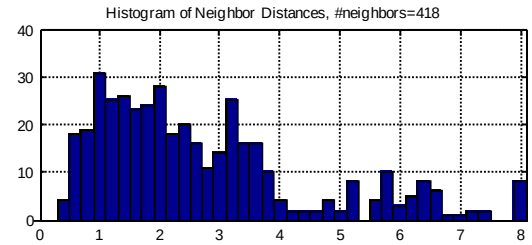


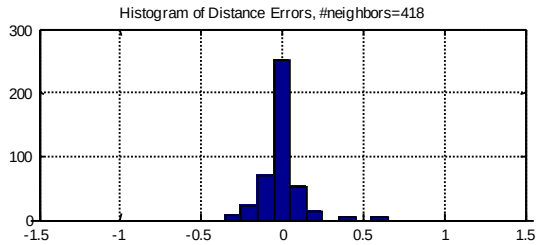


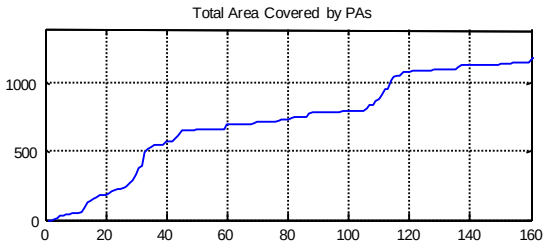


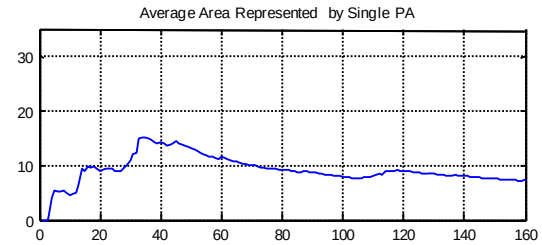












Appendix B: Token to Find on Optimal Path to a Single Best Match

Token are used by the various active threads, such as place agents (PAs) or behaviors, to acquire information from the network of PAs which is locally not available. For example, the route to the nearest coffee is typically locally unknown; hence a "search for coffee"-token explores the whole network by duplicating itself and traversing along all locally available neighbor connections. It finally determines the "best route to coffee" in the network. For exploring the network and returning information to the origin (the PA that started the search), we differentiate between *outbound* and *returning* tokens:

- *outbound tokens* explore away from the origin, remembering the 'best return neighbor' towards the origin at every PA.
- *returning tokens* carry their "best answer to the request" and return to the origin via the 'best return neighbor', as previously determined by outbound tokens.
  A *rejected token* is a special instance of a returning token, indicating that the current outbound path is worse than a previously detected, which terminates the current outbound search along a particular path.

Individual tokens have to provide evaluation functions that allows distinguishing between several possible paths (for cycles in the network), and multiple potential targets (e.g. multiple kitchens providing coffee). A PA - upon receiving an outbound token - calls a function inside that token to update an incremental penalty measure that reflects the value of the particular path the token has taken form its origin to the current PA. Similarly, upon receiving a new token, a PA calls a function inside the token to compute the local target penalty, which determines how likely this PA is the target that the token searches (e.g. if the PA offers coffee).

Only the instance that initially created the token and the current token itself know these evaluation functions; the current PA that received a token does not understand the purpose of these functions. The current PA only provides its local knowledge about the spatial environment that it represents to get inspected by the token's evaluation functions, and in return the PA obtains a (scalar) value from the token's evaluation functions. The creator of a token can provide a very simple or a very complex evaluation function, depending on the task; but the PA does not have to adapt itself to a particular token.

Due to asynchronous operation of PAs and tokens we can absolutely not make any assumption about the order in which messages arrive at PAs. It has happened that a direct neighbor of a token's origin received the first instance of that token through a loop consisting of several other PAs - instead of directly from the origin PA. While sending messages, a PA might get interrupted and rescheduled before finishing. However, we can guarantee that it will send all its messages in order, e.g. send a reject message only after sending further outbound tokens if they are scheduled in that order at a single PA.

Tokens Travelling in the Network

A PA executes the first token of a particular type it receives as a local thread. This thread remains active at least until it found its best answer to the search, and returned this reply towards the origin of the token. All new tokens of the same kind arriving at the PA along different routes are given to this active thread; the original thread thus inspects and processes further instances of that token.

Therefore, in pseudo-code below, we refer to information contained in a new-instance as *newInstance.x* (e.g. outbound distance that this instance has traveled), and information in the 'base thread' as *this.x* (e.g. a unique best returning neighbor towards origin for a particular PA).

Outbound Tokens

Every outbound token arriving at a PA computes its outbound penalty; typically an integrated value for distance traveled. The current PA does not know the total distance traveled from the origin, because it typically does not know the origin PA as one of its neighbors. However, the current PA can provide the distance to its direct neighbor the token came from, whereas the token while on the way can integrate this information at every PA it arrives, thus keeping an accumulated value of the total distance travelled.

If a new token's total outbound penalty is larger than or equal to the smallest previously seen outbound penalty at this place agent (i.e. a token along a shorter path arrived previously), the current PA replies with a 'reject' token and thereby terminates this branch of the outbound search. Alternatively, if the new token's penalty is smaller compared to all previously seen penalties, the PA

a) memorizes the sender of the new instance of the token as best return neighbor towards origin.
b) sends a reject token to the neighbor that was the previous best returning neighbor if existing, thereby notifying the old neighbor that this search-path ended.
c) propagates duplicates of the new instance to all neighbors except the current sender to further explore the network.
d) evaluates the token's local target penalty function (e.g. "does this PA have coffee?").
e) memorizes the sum of outbound penalty at this place and local target penalty (d) as current best result for the particular search this token performs.
f) in case (c) did not find any neighbors to continue exploration: returns the best (i.e. the only) solution computed in (e) to its best return neighbor, no matter how poor the solution is (e.g. even if no coffee is locally available, and thus the local target penalty computed in (d) is extremely high).

Note that outbound tokens might cause two special cases to happen at a PA:

If a PA received a second outgoing token that reports a better route to the origin, this PA will propagate the new outbound token to all its neighbors. Assume one of its neighbors, PA_N, receives a second token from this PA with a smaller outbound penalty compared to the earlier outbound token it received. The neighbor PA_N will immediately send a reject token to the PA, because the PA is either already PA_N's current best return neighbor, or PA_N has seen another better token already. But that behavior is correct, as PA has in total sent 2 tokens to PA_N, it also expects a total of two returning token: one returning a reject, whereas the other one is continuing to explore the network. PA_N will later send another return token, and PA waits for that second token before sending its return to its own best return neighbor.

The second special case occurs when the local target penalty is zero, i.e. the best possible. The PA does not need to send further outbound tokens to its neighbors, since the incremental penalty by definition is positive, and therefore further neighbors cannot find a better result. Instead of propagating the token, the current PA can immediately return its result to the neighbor that sent the token. However, a result of zero is a very special case: usually the evaluation function returns a value

depending on various locally available quantities, such as the quality of coffee, the price of coffee, etc. A penalty of zero implies that this is the absolute optimum; better coffee cannot exist anywhere else in the world. Such a result typically occurs for simple binary searches, as e.g. “find a dead-end to hide”: A place can either be a dead-end or it is not, there is no quality of dead-ends.

Returning Tokens

All returning tokens provide the best result that tokens on the outbound path have found somewhere in the network (see above, outbound token, step (e)). Arriving at a PA, the already running thread for this token compares the returning token’s best target value against the locally already known best target value. This local value is either from the local place’s target evaluation, or has earlier arrived on another return token along a different path. If the new token’s target value is smaller than the previous best penalty, the local thread remembers this new token as ‘best return token’. Once all outbound tokens have returned a result, the overall best returning token is propagated to the neighbor that was remembered as “best neighbor towards origin” amongst possibly multiple outgoing token that arrived at this PA.

DETAILLED PSEUDO-CODE

OUTGOING Token arriving at node

```
 1   compute outbound penalty delta, add to newInstance.outboundPenalty

 2   if newInstance.outboundPenalty < this.bestOutboundPenalty

 3     send RejectToken to previous this.bestNeighbor (if existing)

 4a    copy and propagate newInstance to
            all neighbors except of NewInstance.getSender()
 4b    increase waitForReturningTokensCounter by 1 for each Token sent

 5a    this.bestOutboundPenalty = newInstance.getOutboundPenalty()
 5b    this.bestOriginNeighbor = newInstance.getSender()

 6     compute newInstance.PAIsTargetPenalty()

 7     if newInstance.PAIsTargetPenalty < this.bestTargetPenalty

 8a      this.bestReturningToken = newInstance
 8b      this.bestTargetPenalty = newInstance.PAIsTargetPenalty
 8c      this.bestTargetNeighbor = null
       end

 9     if waitForReturningTokensCounter == 0
10       send this.bestReturningToken to this.bestOriginNeighbor
       end

     else

11     SendRejectToken to newInstance.getSender()

     end
```

RETURNING Token arriving at node

```
 1   if NOT newInstance.isRejected

 2     if newInstance.getTargetPenalty() < this.bestTargetPenalty
 3a      this.bestReturningToken = newInstance
 3b      this.bestTargetPenalty = newInstance. getTargetPenalty()
 3c      this.bestTargetNeighbor = newInstance.getSender()
       end
     end

 4   decrease waitForReturningTokensCounter
 5   if waitForReturningTokensCounter == 0
 6     send this.bestReturningToken to this.bestOriginNeighbor
     end
```

Proof: Tokens find the Shortest Path to a Global Optimal Target:

Assumptions:

0a. The distance between two adjacent nodes reflects the penalty for a token to travel between these two nodes (incremental penalty)

0b. All costs along edges are positive

Proof:

1. If two nodes lie on an optimal path, then the shortest distance between these nodes equals their distance as measured along the path.
2. All optimal paths (from a starting PA to a remote target PA) have the same length.
3. Any outbound token A traveling along an optimal forward path can be blocked only by another such token B, which goes farther than A (uses (0b)).
4. An outbound token traveling along an optimal forward path must reach the end of said path. From (3), consider the token that goes the farthest: It must not be blocked, so it must go the distance of (2).
5. The trail left by a token traveling along an optimal forward path will not be disturbed (uses (1) and (0a)).
6. An outbound token will traverse a complete optimal path and return to the robot. Consider the token of (4): Due to (5), its return token can only be blocked by another return token of (4). Same reasoning as for outbound tokens ((3), (4)), one of these return tokens must return to the origin.
7. The trail left by a token returning along an optimal path will not be disturbed (uses (1)).

When the algorithm finishes, there is an intact optimal path (by (6) and (7)).

Appendix C: Invitations establish a Gradient towards Distal Targets

Establishing a Gradient in the Network

All messages between neighboring PAs are “invitations”. Each such an invitation provides an identifier for its commodity, an evaluation function to be run locally at a PA, and the currently best result (CBR) of the evaluation function. For a commodity, each PA keeps track only of the CBR and which of its neighbors (or itself) provides the best CBR.

If a PA wants to find coffee, it just sends invitations for coffee to all its neighbors, providing its current CBR which will be very high as the PA cannot provide coffee. When a node receives an Invitation, it computes a possibly new better CBR based on the information it received from its neighbor, and does one of the following three things, which can be thought of as accepting the Invitation, doing nothing, or sending back a counter-Invitation:

- If the PA finds the invitation to be a better deal than what it was previously aware of (a lower CBR), then it "converts". It changes its knowledge about best CBR and best neighbor, and it sends invitations based on this new CBR to each of its neighbors except the sender.
- If the PA finds the new CBR to be much worse than what it already knows, then it returns a counter-invitation. "Much worse" here means that the receiver's CBR + the penalty along the way from PA to sender would be better for the sender than what the sender was already offering. The current PA can compute the CBR that the sender will compute upon receiving its invitation, as the current PA knows the distance between itself and the sender. If that imagined CBR is still better then what the sender previously offered, the current PA return a counter invitation knowing that it can provide a lower CBR.
- If neither of these two cases applies, then the PA does nothing. This case means that the PA already knows a CBR that is at least as good as going to the sender, but not so much better that the sender should switch over to the PA’s source.

If a PA receives the first invitation ever for a particular commodity, it sends out invitations all around.

The result of this is that the graph self-organizes, so that each source is the root of a tree covering the region in which it is the closest source. No PA will ever know if the whole graph has been queried, and indeed, if there is a slow link somewhere out in the network, it doesn't have to wait for that node to sort out its reply. Indeed, even if there is a completely lazy node that never replies, the algorithm still works correctly on the graph minus that node.

Maintaining Consistence in Changing Networks

Since the gradient is self-organized and stores information that is meaningful independent of the location the search started, it can fix itself locally. Assume the network has established a gradient towards coffee sources in the network. If a connection between neighboring PAs disappears, or its length increases, then both PAs at the end of that connection need to:

- If the edge was a PA's preferred route to coffee, then the PA, having lost its best route to coffee, simply starts the process over. It sends an invitation to each of its neighbors and

ultimately receives their replies for coffee. Otherwise, if the connection is still there, the PA sends a new invitation on that edge.

- Note that now we need a special rule for what to do when a PA receives an invitation from a neighbor, but that neighbor is how the PA already prefers to get coffee - and the new invitation is worse than what the PA already knows about. This means that something bad has happened to the coffee source in that direction. In this case the PA does the same thing as if the length of the edge had increased: it sends an invitation either for the neighbor's new coffee or for its own bad coffee (whichever is better) to every neighbor. The PA is publicizing that it is no longer the good source of coffee that it used to be, and also hoping for a better counter-invitation.
- If a new connection appears, or its length decreases, then the network has gotten better, and the PAs at either end simply send each other invitations. They will determine if one of them can offer coffee that convinces the other PA to switch.
- If a node appears or disappears in the network, then some edges must also appear or disappear, so that case is already handled.
- If the quality of coffee at a PA improves (the kitchen gets a new espresso machine - and thus a better result in the CBR measure), then it simply sends out invitations to all its neighbors, if this new coffee remains or becomes its favorite.
- If the quality of coffee at a PA decreases, and the PA had previously thought of itself as having the best coffee, then it sends out invitations to all neighbors.

In general, any time a PA becomes uncertain about its information, it can just send out invitations for its own coffee, and its neighbors will soon determine where the good coffee is. The worst that happens is a wave of panic among those PAs that had previously used this uncertain PA on their route to coffee; but this will be followed by a wave of reassurance as the route to coffee is re-propagated. The maximum disturbance from such uncertainty is confined to the sub-tree that depends on the PA that got uncertain.

Summary

In summary, the methods based on invitations solves the problem for every node in the network (not just the initial node), it is robust to local failures, and only localized areas of the network need to be updated if local changes occur in the topology. If there are multiple robots on a single network, then simultaneous inquiries do not interact in a bad way but rather speed each other up.

DETAILLED PSEUDO-CODE

Data Structure: Invitation

```
  invitation.commodity
  invitation.evalFunction
  invitation.currentCost
```

Data Structure: Memory at PA

```
  node.commodity.bestCost
  node.commodity.bestNeighbor
```

Initiating a Gradient for Coffee:

```
  Create new invitation I:
  I.commodity = 'coffee'
  I.currentCost = infinity
  I.evalFunction = 'distance to neighbor + 1/best quality of coffee'
  for each neighbor N send invitation to N
```

Receiving an Invitation (I) at a place agent (PA):

```
  SendToSource = no;
  SendToOthers = yes;
  newLocalCBR = evalFunction(based on I);

  if (newLocalCBR < PA.coffee.bestCost)
                              // accept invitation
    PA.coffee.bectCost = newLocalCBR
    PA.coffee.bestNeighbor = I.sender

  else if (PA.coffee.bestNeighbor == I.sender)
                              // invitation is from our coffee source!

    if (newLocalCBR < I.evalFunction(based on local))
                              // invitation terms are better
                              // than using local coffee
      PA.coffee.bectCost = newLocalCBR    // remember sender as source
      PA.coffee.bestNeighbor = I.sender

    else
                              // use this PA as source
      PA.coffee.bectCost = I.evalFunction(local)
      PA.coffee.bestNeighbor = null
      SendToSource = yes;

    end

  else                        // we have received a poor invitation from
                              // a neighbor who is not our source
    SendToOthers = no;
    if (newLocalCBR < CBR as computed in sender's perspective)
      SendToSource = yes;
    end

  end
  if (SendToSource) send invitation to sender;
  if (SendToOthers) send invitations to all nPA that are not sender;
```

Abstract in German

Eine verteilte Kognitive Karte zur räumlichen Navigation, die auf grafisch organisierten „Orts-Agenten“ basiert

Jörg Conradt
Institut für Neuroinformatik, UZH/ETH-Zürich

Tiere können sich extrem schnell räumliches Wissen aneignen und sich damit innerhalb grosser Reviere sicher bewegen. Ein solche natürliche Methode ist allen heutigen technischen Ansätzen zur Roboternavigation bei Weitem überlegen, da letztere entweder nur kleine räumliche Bereiche abdecken (Arleo 2000) oder aber inakzeptabel hohe Rechenleistung benötigen, um eine global konsistente Karte zu verwalten (Thrun 2002). Wir haben in dieser Arbeit ein neues Modell zur Navigation entwickelt, das wie sein biologisches Vorbild erkundete Gebiete in ein verteiltes System einzelner aber verknüpfter verhaltensrelevanter Plätze zerlegt. Jeder solche Platz wird durch einen eigenen unabhängigen „Orts-Agenten“ (OA) repräsentiert, der aktiv verhaltens- und ortsrelevantes lokales Wissen verwaltet, um Navigation an diesem Ort zu ermöglichen.

Die Gesamtheit solcher Ortsagenten verwaltet Wissen über die Umgebung nicht in einer global konsistenten Datenstruktur wie das heute gängige topologisch- und metrisch basierte Ansätze machen (Abb. 1, links und mittig). Stattdessen entwickelt unser System inkrementell ein grafisches Netzwerk aus eigenständigen OAs, die jeweils nur ihre lokale Umgebung sowie ihre nächsten Nachbarn kennen. Keiner dieser OA kennt seine Position im Netzwerk, eben so wenig die absolute Position des Ortes den er repräsentiert. Die Netzwerktopologie - die die Struktur der befahrbaren Umgebung widerspiegelt - existiert somit nur implizit im System durch Kommunikation der Orta-Agenten mit ihren jeweils direkten Nachbarn (Abb. 1, rechts). Kein Teilnehmer in unserem System kann global auf dem Netzwerk operieren, daher ist auch das Einhalten von räumlicher Konsistenz nur lokal erforderlich.

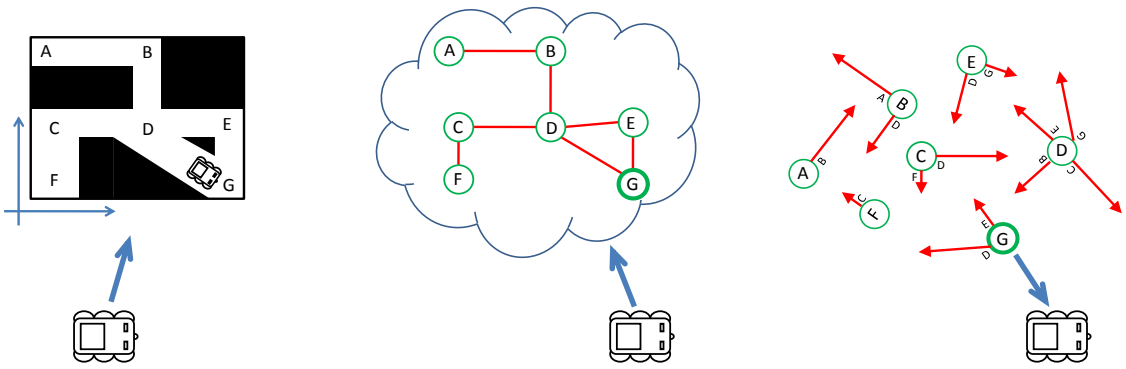


Abbildung 1: Links: Eine globale Kartesische Karte in Aufsicht, von einem einzelnen Roboter oder Programm verwaltet. Mitte: Eine global konsistente topologische Repräsentation der selben Umgebung, wieder von einem einzelnen Roboter verwaltet. Rechts: Eine Ansammlung eigenständiger verteilter Programme (Ortsagenten), die jeweils nur ihren eigenen Ort und ihre direkten Nachbarn kennen. Die tatsächlich unterliegende räumliche Struktur ist nur implizit enthalten.

Solch eine einfache Strategie reduziert Rechenanforderungen drastisch, ist robust gegen lokale Störungen, skaliert gut mit der Grösse der zu erkundenden Umgebung, und erlaubt einem Roboter, autonom und in Echtzeit grosse unbekannte Umgebungen zu erkunden, zu erlernen, und darin zu navigieren. Unser System hat mit Hilfe eines mobilen Roboters bereits unser gesamtes Stockwerk am Institut für Neuroinformatik (60x23m) erkundet und durch etwa 150 Ortsagenten abgebildet. Das resultierende verteilte Netzwerk kann - ohne globalen Akteur - global konsistentes Verhalten erzeugen, wie z.B. den Roboter zu einem früher erkundeten Stimulus zurückführen. Wir erwarten, dass unser System auch problemlos deutlich grössere Umgebungen erkunden kann, da alle Ortsagenten unabhängige Programme sind die sich ohne weiteres auf verschiedene Computer verteilen lassen.

Neuronale Verschaltungen sind gut geeignet, um eine solche „verteilte kognitive Karte“ umzusetzen, da unser Gehirn an sich ein verteiltes Rechensystem darstellt, in dem es keinen globalen Zugriff auf alle Speichereinheiten gibt - was allerdings von heutigen Navigationsprogrammen erfordert wird. Daher sind wir überzeugt, dass das vorgestellte System biologische Informationsverarbeitung wesentlich besser widerspiegelt als andere heutige Methoden zur räumlichen Navigation.

Curriculum Vitae

Jörg-Alexander Conradt

WORK ADDRESS:
Institute of Neuroinformatics
ETH / University of Zürich
Winterthurerstrasse 190
CH-8057 Zürich
Telephone: +41-44-635-3023
eMail: conradt@ini.phys.ethz.ch
Web: http://www.ini.ethz.ch/~conradt

EDUCATION

Jun. 2001 – present	Ph.D. Student at the Institute of Neuroinformatics, ETH / University of Zürich, Switzerland, Advisor: Rodney J. Douglas
Feb. 2001	Diploma in Computer Engineering, Technische Universität Berlin, Germany. Advisor: G. Hommel, GPA: 1.0 / 1.0 (with honors and distinction)
May 2000	Master of Science in Computer Science with Specialization in 'Robotics and Automation', University of Southern California, USA. Advisor: St. Schaal, GPA: 4.0 / 4.0 (with honors and distinction)
Feb. 1999	'Vordiplom' (compares roughly to a BS.) in Computer Engineering, Technische Universität Berlin, Germany, very good

INTERNSHIPS AND PROFESSIONAL APPOINTMENTS

Jan. – Mar. 2001	Associate Internship, McKinsey & Company, Inc., Strategic Management Consulting
Jun. – Aug. 2000	Internship, Kawato Dynamic Brain Project, ATR, Kyoto, Japan. Project: Visual Flow Attraction for Humanoid Robots
Jan. – May 2000	Research Assistant, Prof. Schaal's Laboratory on Computational Learning and Motor Control, University of Southern California. Project: Learning of Inverse Dynamics Models for Compliant Robots
Feb. 1997 – Aug. 1999	Research Assistant, GMD-first (German National Research Center for Information Technology). Projects in Computer Architecture and on Support Vector Machines
Sep. 1995 – Sep. 1996	Mandatory Civil Service, 'Kindergarten Kulleberga', Berlin, Germany
Oct. 1994 – Mar. 1995	Practical Training in Mechanical Engineering and Project Management, Zeiss-IKON-AG, Berlin, Germany

PROFESSIONAL SERVICES

Journal Reviewer:	"IEEE Robotics & Autonomous Systems", "IEEE Transactions on Robotics", "Neural Networks", "IEEE Transactions on Control Systems Technology"
Oct. 2003 –	Member of the Fulbright German National Selection Committee

Oct. 2005	
Dec. 1997 – Aug. 1999	Students' Representative and Chairman of the Educational Committee for Computer Engineering, Technische Universität Berlin
Aug. 1991 – Jun. 1994	Students' President in high school

HONORS, AWARDS, AND GRANTS

Jul. 2002, 2003	Invited Participant of the NSF-Workshop on 'Neuromorphic Engineering' Telluride, Colorado, USA. Received US$ 12.000,- grants to continue projects started during the workshop.
Feb. 2002	Erwin-Stephan-Price of the Technische Universität Berlin, awarded for the best Diploma in 2001 (€ 4.000,-)
Jul. 2001 – Aug. 2001	Invited participant of the EU Advanced Course in Computational Neuroscience, Trieste, Italy
May 2000	Academic Excellence Award, University of Southern California
Aug. 1999 – May 2000	Fulbright Full Grant (one of eight for all German students), includes all fees and maintenance to study one year in the USA (total about US$ 42.000)
Jun. 1999 – Jul. 1999	Invited Participant of the NSF-Workshop on 'Neuromorphic Engineering' Telluride, Colorado, USA
May 1999 – Feb. 2001	Fellow of the Daimler-Chrysler 'Studienförderung Forschung und Technologie' (Research and Technology Merit Foundation)
Oct. 1998 – Jun. 2004	Fellow of the 'Studienstiftung des deutschen Volkes' (German National Merit Foundation)
May 1993, 1994, 1995	Multiple participations in the Young Scientific Contest 'Jugend forscht', ranked 5th, 4th, and 5th over all Germany

PERSONAL

Citizenship:	German
Date of Birth:	25th December 1974
Gender:	Male

Language proficiency:

- German (native)
- English (fluent)
- French (conversational skills)

Memberships:

- Fulbright Alumni Association
- German National Merit Foundation
- Mensa International

Non-professional interests:

Volleyball, Mountainbiking, Snowboarding, Classical Music, Politics

PUBLICATIONS

Birdwell JA., Solomon JH., Thajchayapong M., Taylor MA., Cheely M., Towal RB., Conradt J., and Hartmann MJZ. (2007). Biomechanical Models for Radial Distance Determination by the Rat Vibrissal System. *J. Neurophysiol.* 98(4):2439-55, October 2007. pdf

Hipp J., Arabzadeh E., Zorzin E., Conradt J., Kayser C., Diamond ME., and König P. (2006). Texture signals in whisker vibrations. *J. Neurophysiol*. 95(3):1792-9, March 2006.

Conradt, J. (2005). Helping Neuromorphic sensors leave the designer's desk. *The Neuromorphic Engineer*, 2:(1) 8-9, Mar 2005.

Hipp J., Einhäuser W., Conradt J., and König P. (2005). Learning of somatosensory representtations for texture discrimination using a temporal coherence principle. *Network: Computation in Neural Systems* 16(2-3):223-238.

Conradt, J., Varshavskaya, P., Preuschoff, K., and Douglas, R. (2003). A CPG-driven Autonomous Robot. *Neural Information Processing Systems Conference (NIPS) (Demonstrations Track)*, Vancouver, Canada, Dec 2003.

Bermudez Badia, S., Pyk, P., Conradt, J., and Verschure, PFMJ. (2003). Artificial Moth (Insect-based Neuronal Models of Course Stabilization and Obstacle Avoidance applied to a Flying Robot), *Neural Information Processing Systems Conference (NIPS) (Demonstrations Track)*, Vancouver, Canada, Dec 2003.

Conradt, J., Varshavskaya, P. (2003). Distributed Central Pattern Generator Control for a Serpentine Robot, *Joint International Conference on Artificial Neural Networks and Neural Information Processing (ICANN/ICONIP)*, Istanbul, Turkey, Jun 2003.

Conradt, J., Simon, P., Pescatore, M., and Verschure, P.F.M.J. (2002). Saliency Maps Operating on Stereo Images Detect Landmarks and their Distance, *International Conference on Artificial Neural Networks (ICANN2002)*, Madrin, Spain, August 2002.

Planta, C., Conradt, J., Jencik, A., and Verschure, P.F.M.J. (2002). A Neural Model of the Fly Visual System Applied to Navigational Tasks, *International Conference on Artificial Neural Networks (ICANN2002)*, Madrin, Spain, August 2002.

Vijayakumar, S., D'Souza, A., Shibata, T., Conradt, J., and Schaal, S. (2002). Statistical Learning for Humanoid Robots, *Autonomous Robotics* 12:(1) 55-69, Jan 2002.

Shibata, T., Vijayakumar, S., Conradt, J., and Schaal, S. (2001). Humanoid Oculomotor Control Based on Concepts of Computational Neuroscience, Humanoids2001, *Second IEEE-RAS International Conference on Humanoid Robots*, Waseda University, Japan.

Vijayakumar, S., Conradt, J., Shibata, T., and Schaal, S. (2001). Overt Visual Attention for a Humanoid Robot, *International Conference on Intelligent Robots and Systems (IROS 2001)*, Maui, USA, October/November 2001.

Conradt, J., Tevatia, G., Vijayakumar, S., & Schaal, S. (2000). On-line Learning for Humanoid Robot Systems, *International Conference on Machine Learning (ICML2000)*, Stanford, USA June 2000.

Bibliography

Abbott, L. F. (1994). "Decoding Neuronal Firing And Modeling Neural Networks." Quaterly Review Biophysics **27**: 291-331.

Adams, M. (2007). "SLAM — algorithmic advances, loop closing, measurement classification and outdoor implementations." Robotics and Autonomous Systems **55**(1): 1-2.

Allen, G. L. (2006). Applied Spatial Cognition, Lawrence Erlbaum Associates Inc,US.

Alwan, M., M. B. Wagner, et al. (2005). Characterization of Infrared Range-Finder PBS-03JN for 2-D Mapping. ICRA, Barcelona, Spain, IEEE

Applegate, D. L., R. E. Bixby, et al. (2007). The Traveling Salesman Problem: A Computational Study, Princeton University Press.

Arbib, M. A. (2003). The Handbook of Brain Theory and Neural Networks. Cambridge, MA, USA, MIT Press.

Arleo, A. (2000). Spatial Learning and Navigation in Neuro-Mimetic Systems, Modeling the Rat Hippocampus. Lausanne, Ecole Polytechnique Federale Lausanne. **Ph.D.**

Arleo, A. (2005). The Neural Bases of Spatial Cognition and Information Processing in the Brain. Paris, University Pierre & Marie Curie. **Habilitation**.

Arleo, A. and W. Gerstner (2000). "Spatial cognition and neuro-mimetic navigation: A model of hippocampal place cell activity." Biological Cybernetics **83**: 287-299.

Balakrishnan, K., O. Bousquet, et al. (1998). "Spatial Learning and Localization in Animals: A Computational Model and its Implications for Mobile Robots." Adaptive Behavior **7**(2): 173-216.

Beeler, T. (2004). Eukalyptus - Koala Environment Simulator. Computer Science. Rapperswil, Institute of Applied Science. **Diplom**.

Bellingham, J. G. and K. Rajan (2007). "Robotics in Remote and Hostile Environments." Science **318**(5853): 1098-1102.

Benhamou, S. (1996). "No evidence for cognitive mapping in rats." Animal Behaviour **52**(1): 201-212.

Bennett, A. T. (1996). "Do Animals have Cognitive Maps." The Journal of Experimental Biology **199**: 219-224.

Blancheteau, M. and A. L. Lorec (1972). "Raccourci et détour chez le rat: durée, vitesse et longueur des parcours [Short-cut and detour in rats: duration, speed and length of course]." L'année psychologique **72**(1): 7-16.

Borenstein, J. (1994). Internal Correction of Dead-reckoning Errors With the Smart Encoder Trailer. International Conference on Intelligent Robots and Systems, Munich.

Bosse, M., P. Newman, et al. (2004). "Simultaneous Localization and Map Building in Large-Scale Cyclic Environments Using the Atlas Framework." The International Journal of Robotics Research **23**(12): 1113-1139.

Bosse, M., P. M. Newman, et al. (2003). "SLAM in Large-scale Cyclic Environments using the Atlas Framework." International Journal of Robotics Research.

Bures, J., A. A. Fenton, et al. (1997). "Place cells and place navigation." Proceedings of the National Academy of Sciences of the Unites States of America **94**: 343-350.

Burgess, N., K. J. Jeffery, et al. (1999). The Hippocampal and Parietal Foundations of Spatial Cognition. Oxford, Oxford University PRess.

Castellanos, J. A. and J. D. Tardos (2000). Mobile Robot Localization and Map Building: A Multisensor Fusion Approach, Kluwer.

Chapuis, N., M. Durup, et al. (1987). "The role of exploratory experience in a shortcut in golden hamsters (*Mesocricetus auratus*)." Animal Learning and Behavior **15**(174-178).

Chokshi, K., S. Wermter, et al. (2005). Image Invariant Robot Navigation Based on Self Organising Neural Place Codes. Biomimetic Neural Learning for Intelligent Robots, Springer.

Collett, T. S., M. Collett, et al. (2001). "The guidance of desert ants by extended landmarks." Journal of Experimental Biology **204**(9): 1635-1639.

Collett, T. S., P. Graham, et al. (2007). "Novel landmark-guided routes in ants." The Journal of Experimental Biology **210**: 2025-2032.

Consi, T. R. and B. Webb (2001). Biorobotics - Methods & Applications, AAAI Press.

Cook, W. "http://www.tsp.gatech.edu/." Retrieved April, 2008.

Cormen, T. H., C. E. Leiserson, et al. (1990). Introduction to Algorithms. Cambridge, Massachusetts, The MIT Press.

Cressant, A., R. U. Muller, et al. (2002). "Remapping of place cell firing patterns after maze rotations." Experimental Brain Research **143**: 470-479.

Cuperlier, N., M. Quoy, et al. (2007). "Neurobiologically inspired mobile robot navigation and planning." Frontiers in Neurorobotics **1**: 1-14.

Dissanayake, M. W. M. G., P. Newman, et al. (2002). "A Solution to the Simultaneous Localization and Map Building (SLAM) problem." IEEE Transactions on Robotics and Automation **17**(3): 229-241.

Downs, R. M. (1981). Maps and mapping as metaphors for spatial representation. Spatial Representation and Behavior across the Life Span. L. S. Liben, A. H. Patterson and N. Newcombe. New York, NY, USA, Academic Press**:** 143-166.

Downs, R. M. and D. Stea (1973). Theory. Image and Environment. R. M. Downs and D. Stea. Chicago, Il., Aldine.

Duckett, T., S. Marsland, et al. (2000). Learning Globally Consistent Maps by Relaxation. IEEE International Conference on Robotics and Automation, 2000, ICRA '00., San Francisco, CA, USA.

Dyer, F. C. (1991). "Bees acquire route-based memories but not cognitive maps in a familiar landscape." Animal Behavior **41**: 239–246.

Eichenbaum, H., P. Dudchenko, et al. (1999). "The hippocampus, memory, and place cells: is it spatial memory or a memory space?" Neuron **23**(2): 209-226.

Ekstrom, A. D., M. J. Kahana, et al. (2003). "Cellular networks underlying human spatial navigation." Nature **425**: 184-188.

Evolution-Robotics (2008). "http://www.evolution.com/core/navigation/vslam.masn." Retrieved April, 2008, from http://www.evolution.com/core/navigation/vslam.masn.

Fent, K. (1986). " Polarized skylight orientation in the desert antCataglyphis." Journal of Comparative Physiology A: Neuroethology, Sensory, Neural, and Behavioral Physiology **158**(2): 145-150.

Filliat, D. and J.-A. Meyer (2002). Global localization and topological map learning for robot navigation. From Animals to Animats 7. Proceedings of the Seventh International Conference on Simulation of Adaptive Behavior (SAB'02), Edinburgh, UK.

Foster, D. J. and M. A. Wilson (2006). "Reverse replay of behavioural sequences in hippocampal place cells during the awake state." Nature **440**: 680-683.

Fox, D. (1998). Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation. Institute of Computer Science III Bonn, University of Bonn. **Ph.D.**

Franz, M. O., B. Schölkopf, et al. (1998). "Learning View Graphs for Robot Navigation." Autonomous Robots **5**: 111-125.

Fyhn, M., T. Hafting, et al. (2007). "Hippocampal remapping and grid realignment in entorhinal cortex." Nature **446**: 190-194.

Fyhn, M., S. Molden, et al. (2004). "Spatial Representation in the Entorhinal Cortex." Science **305**: 1258 - 1264.

Gallistel, C. R. (1993). The Organization of Learning. Cambridge, MA, MIT Press.

Gaussier, P., J. P. Banquet, et al. (2007). "A model of grid cells involving extra hippocampal path integration and the hippocampal loop." Journal of Integrative Neuroscience **6**(3): 447-476.

Gaussier, P., C. Joulain, et al. (2000). "The visual homing problem: an example of robotic/biology cross fertilization." Robotics and Autonomous Systems **30**: 115-180.

Gillner, S. and H. A. Mallot (1998). "Navigation and Acquisition of Spatial Knowledge in a Virtual Maze." Journal of Cognitive Neuroscience **10**(4): 445-463.

Golfarelli, M., D. Maio, et al. (2001). "Elastic Correction of Dead-Reckoning Errors in Map Building." IEEE Transactions on Robotics and Automation **17**(1): 37-47.

Golledge, R. G. (1999). Wayfinding Behavior. Baltimore and London, The Johns Hopkins University Press.

Gothard, K. M., W. E. Skaggs, et al. (1996). "Dynamics of Mismatch Correction in the Hippocampal Ensemble Code for Space: Interaction between Path Integration and Environmental Cues." The Journal of Neuroscience **16**(24): 8027-8040.

Gould, J. L. (2004). "Animal navigation." Current Biology **14**: R221-R224.

Gutierrez-Osuna, R. and R. C. Luo (1996). "LOLA: Probabilistic Navigation for Topological Maps." AI Magazine **17**(1): 55-62.

Hafner, V. V. (2000). Learning Places in Newly Explored Environments. SAB2000 Proceedings Supplement Book, Honolulu, International Society for Adaptive Behavior.

Hafner, V. V. (2008). Robots as Tools for Modelling Navigation Skills - A Neural Cognitive Map Approach. Robot and Cognitive Approaches to Spatial Mapping. M. E. Jefferies and W.-K. Yeap. Heidelberg/Berlin, Springer. **38:** 315-324.

Hafting, T., M. Fyhn, et al. (2005). "Microstructure of a spatial map in the entorhinal cortex." Nature **436**: 801-806.

Hahnloser, R., X. Xie, et al. (2002). A Theory of Integration in the Head-Direction System. Neural Information Processing Systems, Vancouver, Canada.

Harwood, J. (2006). To the Ends of the Earth: 100 Maps That Changed the World David & Charles.

Healy, S. (1998). Spatial Representation in Animals. Oxford, Oxford University Press.

Hok, V., P.-P. Lenck-Santini, et al. (2007). "Goal-Related Activity in Hippocampal Place Cells." The Journal of Neuroscience **27**(3): 472-482.

Jacobs, L. F. (2003). "The Evolution of the Cognitive Map." Brain, Behavior and Evolution **62**: 128-139.

Jefferies, M. E., J. Baker, et al. (2008). Robot Cognitive Mapping - A Role for a Global Metric Map in a Cognitive Mapping Process. Robot and Cognitive Approaches to Spatial Mapping. M. E. Jefferies and W.-K. Yeap. Heidelberg/Berlin, Springer. **38:** 265-280.

Jefferies, M. E. and W.-K. Yeap (2008). Robotics and Cognitive Approaches to Spatial Mapping. Berlin, Springer.

K-Team-SA. "http://www.k-team.com/kteam/index.php?site=1&rub=22&page=18&version=EN."

Kandel, E. R., J. H. Schwartz, et al. (2000). Principles of Neural Science, Appleton & Lange.

Kitchin, R. and s. Freundshuh (2000). Cognitive Mapping. Lodon, Routledge.

Kitchin, R. M. (1994). "Cognitive Maps: What are they and why study them?" Journal of Environmental Psychology **14**: 1-19.

Knaden, M., C. Lange, et al. (2006). "The importance of procedural knowledge in desert-ant navigation." Current Biology **16**(21): R916-R917.

Koene, R. A., A. Gorchetchnikov, et al. (2003). "Modeling goal-directed spatial navigation in the rat based on physiological data from the hippocampal formation." Neural Networks **16**(5-6): 577-584.

Kortenkamp, D. and T. Weymouth (1994). Topological mapping for mobile robots using a combination of sonar and vision sensing. 12th national conference on Artificial intelligence, Seattle, Washington, USA.

Krichmar, J. L., D. A. Nitz, et al. (2005). "Characterizing functional hippocampal pathways in a brain-based device as it solves a spatial memory task." Proceedings of the National Academy of Sciences of the Unites States of America **102**(1026): 2111-2116.

Kuipers, B. (1978). "Modeling Spatial Knowledge." Cognitive Science **2**: 129-153.

Kuipers, b. (1983). The Cognitive Map: Could It Have Been Any Other Way? Spatial Orientation: Theory, Research, and Application. H. L. Pick and L. P. Acredolo. New York, Plenum Press**:** 345-360.

Kuipers, B., J. Modayil, et al. (2004). Local Metrical and Global Topological Maps in the Hybrid Spatial Semantic Hierarchy. IEEE International Conference on Robotics and Automation, Proceedings., New Orleans, IEEE.

Kuipers, B. J. (1977). Representing knowledge of large-scale space. Mathematics Department. Boston, Massachusetts Institute of Technology. **Ph.D.**

Kuipers, B. J. (2000). "The Spatial Semantic Hierarchy." Artificial Intelligence **119**: 191-233.

Kulvicius, T., M. Tamosiunaite, et al. (2007). Development of place cells by a simple model in a closed loop context. Sixteenth Annual Computational Neuroscience Meeting: CNS*2007, Toronto, Canada.

Lagoudakis, M. G. (1998). Spatial Knowledge in Humans, Animals and Robots.

Lambrinos, D., R. Möller, et al. (1999). "A Mobile Robot Employing Insect Strategies for Navigation." Robotics and Autonomous Systems, special issue: Biomimetic Robots **30**: 39-64.

Leonard, J. J. and H. F. Durrant-Whyte (1992). Directed Sonar Sensing for Mobile Robot Navigation. Dordrecht, Kluwer Academic Publishers.

Leutgeb, S., J. K. Leutgeb, et al. (2005). "Place cells, spatial maps and the population code for memory." Current Opinion in Neurobiology **15**: 738-746.

Lynch, K. (1960). The Image of the City. Cambridge, MIT Press.

Mackintosh, N. J. (2002). "Do not ask whether they have a cognitive map, but how they find their way about." Psicológica **23**: 165-185.

Maguire, E. A., D. G. Gadian, et al. (2000). "Navigation-related structural change in the hippocampi of taxi drivers." Proceedings of the National Academy of Sciences of the Unites States of America **98**(8): 4398-4403.

Maguire, E. A., K. Woollett, et al. (2006). "London Taxi Drivers and Bus Drivers: A Structural MRI and Neuropsychological Analysis." Hippocampus **16**: 1091-1101.

Mallot, H. A., H. H. Bülthoff, et al. (1995). View-based cognitive map learning by an autonomous robot. International Conference on Artificial Neural Networks.

Mallot, H. A., S. Gillner, et al. (1998). Behavioral Experiments in Spatial Cognition Using Virtual Reality. Lecture Notes In Computer Science. London, UK, Springer-Verlag. **1404:** 447-468.

Markus, E. J., Y. L. Qin, et al. (1995). "Interactions between location and task affect the spatial and directional firing of hippocampal neurons." Journal of Neuroscience **15**(11): 7079-7094.

McNaughton, B. L., F. P. Battaglia, et al. (2006). "Path-integration and the neural basis of the "cognitive map"." Nature Reviews Neuroscience **7**: 663-678.

Menzel, E. W. (1973). "Chimpanzee spatial memory organization." Science **182**: 943-945.

Menzel, R., R. Brandt, et al. (2000). "Two spatial memories for honeybee navigation." Proceedings of the Royal Society B **267**(1447): 961-968.

Menzel, R., U. Greggers, et al. (2005). "Honey bees navigate according to a map-like spatial memory." Proceedings of the National Academy of Sciences of the Unites States of America **102**(8): 3040-3045.

Milford, M. J. (2008). Robot Navigation from Nature. Berlin/Heidelberg, Springer.

Möller, R., D. Lambrinos, et al. (2001). Insect Strategies of Visual Homing in Mobile Robots. Biorobotics - Methods and Applications. B. Webb and T. Consi. Menlo Park, CA, AAAI Press/MIT Press**:** 37-66.

Montemerlo, M. and S. Thrun (2007). The FastSLAM Algortihm for Simultaneous Localization and Mapping. Berlin/Heidelberg, Springer.

Morris, R. G. M. (1981). "Spatial localization does not require the presence of local cues." Learning and Motivation **12**: 239-260.

Moser, E. I., E. Kropff, et al. (2008). "Place cells, grid cells and the brain's spatial representation system." Annual Reviews of Neuroscience.

Müller, M. and R. Wehner (1988). "Path Integration in desert ants, Cataglyphis fortis." Proceedings of the National Academy of Sciences of the Unites States of America **85**: 5287-5290.

Muller, R. (1996). "A quarter of a century of place cells." Neuron **17**(5): 813-822.

Newman, P. M., J. J. Leonard, et al. (2005). Towards Constant-Time SLAM on an Autonomous Underwater Vehicle Using Synthetic Aperture Sonar. Robotics Research. P. Dario and R. Chatila, Springer. **15**.

O'Keefe, J. and J. Dostrovsky (1971). "The hippocampus as a spatial map: preliminary evidence from unit activity in the freely moving rat." Brain Research **34**(171-175).

O'Keefe, J. and L. Nadel (1978). The Hippocampus as a Cognitive Map. Oxford, UK, Oxford University Press.

Olthof, A., J. E. Sutton, et al. (1999). "In search of the cognitive map: Can rats learn an abstract pattern of rewarded arms on the radial maze?" Journal of Experimental Psychology: Animal Behavior Processes **25**(3): 352-362.

Olton, D. S. and R. J. Samuelson (1976). "Remembrances of places passed: spatial memory in rats." Journal of Experimental Psychology: Animal Behavior Processes **2**: 97-116.

Patnaik, S. (2007). Robot Cognition and Navigation. Berlin, Springer.

Poucet, B. (1993). "Spatial Cognitive Maps in Animals: New Hypotheses on Their Structure and Neural Mechanisms." Psychological Review **100**(2): 163-182.

Ranck, J. B. (1984). Head-direction cells in the deep cell layers of the dorsal presubiculum in freely-moving rats. Soc. Neurosci. Abstr.

Redish, A. D. (1999). Beyond the Cognitive Map - From Place Cells to Episodic Memory, MIT Press.

Robotshop-Canada. "http://www.robotshop.ca/home/products/robot-parts/sensors/infrared-sensors/hokuyo-pbs-03jn-infrared-rangefinder.html."

Rosenzweig, E. S., A. D. Redish, et al. (2003). "Hippocampal map realignment and spatial learning." Nature Neuroscience **6**(6): 609-615.

Samsonovich, A. and B. L. McNaughton (1997). "Path Integration and Cognitive Mapping in a Continuous Attractor Neural Network Model." The Journal of Neuroscience **17**(15): 5900-5920.

Sargolini, F., M. Fyhn, et al. (2006). "Conjunctive representation of position, direction, and velocity in entorhinal cortex." Science **312**(5774): 758-62.

Schölkopf, B. and H. A. Mallot (1995). "View-based cognitive mapping and path planning." Adaptive Behavior **3**(3): 311-348.

Se, S., D. Lowe, et al. (2002). "Mobile Robot Localization and Mapping with uncertainty using Scale-Invariant Visual Landmarks." The International Journal of Robotics Research **21**(8): 735-758.

Shatkay, H. and L. P. Kaelbling (1997). Learning Topological Maps with Weak Local Odometric Information. International Joint Conference on Artificial Intelligence.

Smith, L., A. Philippides, et al. (2007). "Linked Local Navigation for Visual Route Guidance." Adaptive Behavior **15**(3): 257-271.

Spiers, H. J. and E. A. Maguire (2007). "A Navigational Guidance System in the Human Brain." Hippocampus **17**: 618-626.

Srinivasan, M. V., S. Zhang, et al. (2000). "Honeybee Navigation: Nature and Calibration of the "Odometer"." Science **287**(5454): 851-853.

Stringer, S. M., E. T. Rolls, et al. (2002). "Self-organizing continuous attractor networks and path integration: two-dimensional models of place cells." Network: Computation in Neural Systems **13**(2): 217-242.

Tapus, A. (2005). Topological SLAM - Simultaneous Localization and Mapping with Fingerprints of Places. Computer Science and Systems Engineering Department. Lausanne, Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland. **Ph.D.**

Tapus, A., F. Battaglia, et al. (2006). The Hippocampal Place Cells and Fingerprints of Places: Spatial Representation Animals, Animats and Robots. Intelligent Autonomous Systems Conference.

Tapus, A. and R. Siegwart (2006). A Cognitive Modeling of Space using Fingerprints of Places for Mobile Robot Navigation. IEEE International Conference on Robotics and Automation, Orlando Florida.

Tapus, A. and R. Siegwart (2006). Fingerprints of Places: A Model of Hippocampal Place Cells. IEEE International Conference on Robotics and Automation. Orlando, Florida.

Taube, J., R. Muller, et al. (1990). "Head-direction cells recorded from the postsubiculum in freely moving rats. I. Description and quantitative analysis." Journal of Neuroscience **10**: 420-435.

Thrun, S. (1998). "Learning Metric-Topological Maps Maps for Indoor Mobile Robot Navigation." Artificial Intelligence **99**(1): 21-71.

Thrun, S. (2000). "Probabilistic algorithms in robotics." AI Magazine **21**(4): 93-109.

Thrun, S. (2002). Robotic mapping: A survey. Exploring Artificial Intelligence in the New Millenium. G. Lakemeyer and B. Nebel, Morgan Kaufmann.

Thrun, S. (2008). Simultaneous Localization and Mapping. Robot and Cognitive Approaches to Spatial Mapping. M. E. Jefferies and W.-K. Yeap. Heidelberg/Berlin, Springer. **38:** 13-42.

Thrun, S., W. Burgard, et al. (2005). Probabilistic Robotics. Cambridge, MA, The MIT Press.

Todd, E. T. (1997). "Virtually Lost in Virtual Worlds Wayfinding without a Cognitive Map." ACM Computer Graphics **31**: 3.

Tolman, E. C. (1948). "Cognitive Maps in Rats and Men." The Psychological Review **55**(4): 189-208.

Tolman, E. C. and C. H. Honzik (1930). ""Insight" in rats." University of California Publications in Psychology **4**(14): 215-232.

Tolman, E. C., B. F. Ritchie, et al. (1946). "Studies in Spatial Learning. I. Oriantation and the short-cut." Journal of Experimental Psychology **36**: 13-24.

Tomatis, N., I. R. Nourbakhsh, et al. (2001). Simultaneous Localization and Map Building: A Global Topological Model with Local Metric Maps. IEEE/RSJ International Conference on Intelligent Robots and Systems, Maui, USA.

Tomatis, N., I. R. Nourbakhsh, et al. (2003). "Hybrid simultaneous localization and map building: a natural integration of topological and metric." Robotics and Autonomous Systems **44**: 3-14.

Touretzky, D. S., H. S. Wan, et al. (1994). Neural Representation of Space in Rats and Robots. IEEE World Congress on Computational Intelligence) Computational Intelligence: Imitating Life, Piscataway, NJ.

Vardy, A. and R. Möller (2005). "Biologically Plausible Visual Homing Methods Based On Optical Flow Techniques." Connection Science **17**: 47-89.

Wehner, R., M. Boyer, et al. (2006). "Ant Navigation: One-Way Routes Rather Than Maps." Current Biology **16**: 75-79.