ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

ini

Christian P. Brändli

Driving With Spikes

Lane Detection In A Semi-Neuroinspired Manner

Bachelor Project

Institute Of Neuroinformatics
Swiss Federal Institute of Technology (ETH) Zurich

Advisors

Tobias Delbrück
Thomas Besselmann
Rodney Douglas

August 25, 2008

Abstract

The goal of this project was to create an algorithm that allows an autonomous robotic car to find its way along a chalk drawn lane by using the inputs of a spike-event-based silicon retina. This machine vision task could not be tackled with the existing frame based vision algorithms. A neuroinspired processing concept was developed whose methods are able to track a line as well as a lane. The events are filtered to increase the signal to noise ratio, undistorted and processed with a line prediction algorithm. The extracted information can be used to handle the controlling of the car in a virtual environment and in reality. The car is able to follow a line and a lane in the virtual reality and it can follow a line on a real street at walking pace.

Contents

1 Introduction

1.1 The Project

The *TmpDiff128* chip in the silicon retina camera is a powerful tool which processes visual inputs in a similar way our retina does [1]. Due to its neuromorphic architecture it has advantages compared to a classical video camera which make it suitable for a machine vision task. The cameras output are address-events which only contain the information where (spacial coordinates of the pixel), when (a timestamp) and what kind of contrast change (from dark to bright or the other way round) happened within the visual field of the camera. So it doesn't produce huge amounts of redundant information by reading out each pixel to create a frame, as a normal camera does. It just waits until something happens and then sends out these events - so if neither the camera moves nor something in front of it, there are no contrast changes and nothing is sent out. This does not only reduces the amount of data but also allows a high and adjustable (not readout frequency dependent) temporal resolution. Additionally the camera only sends out "'interesting"' informations, that come from objects that change their position relative to the camera and this makes a lot of visual preprocessing superfluous. So all in all the silicon retina camera seems to be tailor-made for dynamic machine vision tasks.

It was the goal of this bachelor project to demonstrate that one can really use the silicon retina for such a machine vision task, which in our case was to let a robotic car follow a lane with high velocity. Therefore the camera together with a micro PC, an acceleration sensor, a servo-controller micro chip and an USB-hub were mounted on a 4x4 remote controlled electric monster truck. The data that came from the camera was transmitted with an address event representation protocol (AER) via the USB port to the micro PC where it then got read out by the java project jAER [2]. So the core of the project was to develop an algorithm within the jAER project, that processes the incoming events in a way that it can send out the proper commands to the servo board which controls the motors of the RC-Car.

This project was realized in collaboration with Robin Ritz and while this part of project is about recognizing the lane and its borderlines, he developed a model of the car in a 3D-environment and set up

the controlling algorithms to steer the car. The merging and testing of the algorithms in the real world environment were then done together.

1.2 Neuroinspired Programming

Since the data we used as input was not made of images or frames but events, we couldn't use any classical approaches because converting our data into frames would have destroyed its neuromorphic nature, i.e. its advantages. The goal was therefore to develop a concept that processes the data in a way that can handle its event-based nature as well as using its advantages and so it was obvious to treat them in a similar way the brain does - in a neuroinspired way. But what does this mean?
If one wants to process data in a computer the way the brain does it, there are certain principal problems to solve and one should follow some rules. Thinking about these limitations lead me to the following conclusions:

The Paradox Of Neuroinspired Programming

The paradox of neuroinspired Programming can be reduced to the following sentence: **The brain is no Turing machine.** Since the computer and the whole idea of programming, i.e. building algorithms is based on the idea of a Turing machine, it will never be possible to build a program that works in the way our brain does because it works in a fundamentally different way. How does this come?
The brain is a huge network of neurons that work in a simple way: They let other neurons deposit charge on their membrane until the voltage across this membrane reaches a threshold which makes them fire, i.e. by sending out spikes they deposit charge on other neurons. So to speak they are nothing else than analog to digital converters which integrate analog currents to a certain threshold at which they then create a digital output pulse. In reality this is more complex of course but this main principle describes a lot very well. What makes the brain computing, is not a single neuron, it is the network with its connections: In contrast to a computer a brain does not distinguish between memory and processing unit - the network is both. It is the wiring and weighing of the connections in between the neurons that determines what the output for a certain input is. This is also the reason why a brain is able to learn - it has the ability to change its computation structure by changing the strength of the connections.
An output of one neuron is the input of many other neurons, which

is parallel and simultaneous signal transduction and leads to the fact that the whole brain computes in parallel. And this is exactly the difference between a brain and a Turing machine: While a Turing machine can only execute serial ordered commands, the brain computes in parallel through the network given structures.
The implementation of the network would not be a real problem but it is impossible to do simultaneous calculations on a computer with only one CPU. Neuroinspired Programming therefore tries to emulate the brain within the serial command structure of a computer by dividing the time into sections in which pseudo-simultaneous actions are possible.

The Rules Of Neuroinspired Programming

In order to guarantee the neuroinspired nature of the developed programming code, some rough rules about handling the data were set up:

- **Do It Like The Brain** The calculations done should also be possible for a network. This leads to a processing structure that contains different layers of 'neurons' (stored in arrays) that change their state depending on the input they get. Each neuron has a defined amount of possible input and output-neurons which can change the state of the neuron or can get changed by it. No neuron should have the possibility to change a whole layer (no 'supervision neurons')
- **No Event - No Computation** Since the brain does not know any algorithms, the job of the programmer is to prepare structures in which an event can drive a cascade of reactions. This can be done by simple arithmics or if-statements. It is forbidden to implement any supervised process that scans through an entity. The whole computing should be event-driven and organized bottom-up.
- **Simultaneous is Simulated** Supervised scanners or iterators (e.g. for-loops) are only allowed if they simulate processing steps that would be simultaneous. So a for-loop should for example be used if it represents the effect of an event that happens at the same time in an entity (e.g. a neuron that projects to a number of other neurons).

Most of the project was implemented following these rules but certain processing steps couldn't done this way, so other methods were used which made the approach a semi-neuroinspired one.

2 System Description

In the following all methods and technical devices to realize the project used are presented.

2.1 The Hardware

Camera

The camera used was an asynchronous temporal contrast silicon retina with a TmpDiff128-chip that has a 128x128 resolution, 120dB dynamic range, 23mW power consumption, 2.1%-contrast threshold mismatch and 15 μs-latency, [10].

Micro PC

The micro PC used was a VAIO®UX 180 Micro PC with an Intel Core™2 Solo processor U2200 (1.2GHz, 2MB L2 cache) and Windows XP. It has 500 MB RAM and a 16 GB flash HD, [11].

RC Car

The RC Car used was a TRAXXAS E-MAXX™with a 14-Volt power system for two twin Titan®550 motors and a top speed over 50 kph.

Servo Board

The servo board used was a USBservo board with a Silicon Labs C8051F320, [10], capable of controling four servo motors and automatic override of the computer controls from the RC headset.

Structure

Camera Mount

A first construction to mount and protect the camera that was made of wood and used a mirror but it had several drawbacks: It cut off a very important part of the visual field of the camera and it broke after a crash after which it had to be repaired. A second construction we built which was made of wood and plexi and that used a spring and a screw to adjust the camera angle also broke after intense testdriving so P. Lichtsteiner made a solidly built, full plexi construction that also allowed to adjust the angle of view.

Acceleration Sensor And PC Mount

In the beginning the Micro PC was just put into the chassis of the Car so it was badly protected, rather

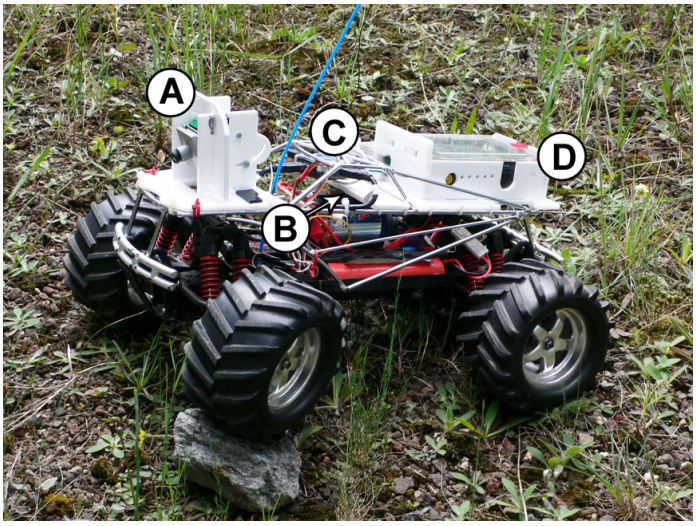

Figure 2.1: The RC-car with its components: (A) The silicon retina and its mount (B) The acceleration sensor which is mounted underneath the white plate (C) The usb-hub (D) The micro PC within its protective box

loose and hard to access. To improve the protection and fixation, a full plexi box with transparent and removable cover was built in a manner that the PC was tightly locked inside.
To ensure an accurate acceleration measurement in the center of the car, the acceleration sensor was screwed, very well protected, on a plexi piece which allows different positions and orientations of the sensor.

2.2 The Software

The jAER-project

The jAER project is a collection of objects and methods to handle AER (adress event representation) data in java. It served us as basis to develop the algorithms and its GUI, the jAER-viewer made it easy to see the results and adjust the parameters and biases.

NetBeans

All code was written in the IDE NetBeans which is an open source development environment. NetBeans was also used to profile the performance of our algorithms.

RealVNC

To see the real-time performance of our algorithms on the RC-car and to adjust the parameters we tried to use RealVNC over wireless LAN but for non-ideal WLAN conditions the connection was to slow and useless.

2.3 The Execution

The Programming

The building of the basic algorithms (Feb-May) was done by using already existing code and algorithms as pattern. Most of the programming style was taken over from the preexisting jAER objects and methods.

The Refinement & Testing

The refinement of the code, the testing and the adjusting of the parameters took far more time than planned. This had several reasons:

- **The WLAN-Connection:** The adjusting of the parameters could not be done real-time because is was impossible to set up a WLAN that was fast enough to maintain the connection to the micro PC. So the parameters had to be adjusted by hand (unplugging and taking the micro PC out of the box) and USB-Stick which took very long.
- **The Test-Track:** It was not possible to find a testing area which was large enough, covered, with a black ground and enough light. An almost ideal spot in the parking garage could not be used because the retina doesn't work with deeply-modulating sodium lighting.
- **The Power:** The power of the micro PC and the RC-car only lasted for about one hour. While the RC-car was very fast recharged, the mirco pc took very long. This made the timeframe for testing very narrow. It became even narrower because the power packs of the car became weaker and weaker.
- **The Weather:** Often when test-runs were planned, it rained or started to, which made driving impossible because the micro PC would have gotten wet.
- **The Light Conditions:** Parameters that were right for sunshine did not work anymore for overcast or twilight conditions. So changing conditions led to a lot of adjustment work.
- **The Lens:** While a small angle lens made it impossible to see the whole lane, a wide angle lens distorted the picture and reduced the resolution and both of these options caused problems hard to tackle.

3 The Implementation

In the following not only the solutions we found for the problems that arisen during tackling the task are depicted, but also some approaches that failed or seemed unsuitable for certain problems. To fully understand what the algorithms are doing one should have a look at the code which can be found on the attached CD. The `typed` words represent variables or classes of the implemented algorithm which are in some cases accessible as parameters in the jAER-viewer.

3.1 The Architecture

The idea behind the algorithm architecture was to process the visual input information and solve the problems in the same order and a similar way our brain does. So the task was split into a number of processing steps (specifically into software classes) which have an analog in the brain (fig. 3.1):

1. **Retina - TmpDiff Chip:** The preprocessing which is done in the retina is in our case undertaken by the TmpDiff Chip that produces the first signals to process in jAER.

2. **LGN - attention:** By using informations about the position of the line, the attention around it modulates the intensity of the incoming events. In the brain the lateral geniculate nucleus (LGN) has a similar function and helps focusing the attention on the important inputs. Since this attention driven modulation is not very complicated it is implemented as a part of the OrientationCluster.

3. **Visual Cortex - *OrientationCluster*** The orientation of the presented stimuli is extracted, which in the brain is done by simple cell columns and if they satisfy some criteria - in our case this is being part of a line - they are forwarded.

4. **Visual Cortex - *PerspecTransform*** The PerspecTransform then filters out the uninteresting events that do not origin from the road and it transforms the image to account for the perspective and the lens distortion. It is thought that the perspective transformation in the brain is also done in the visual cortex.

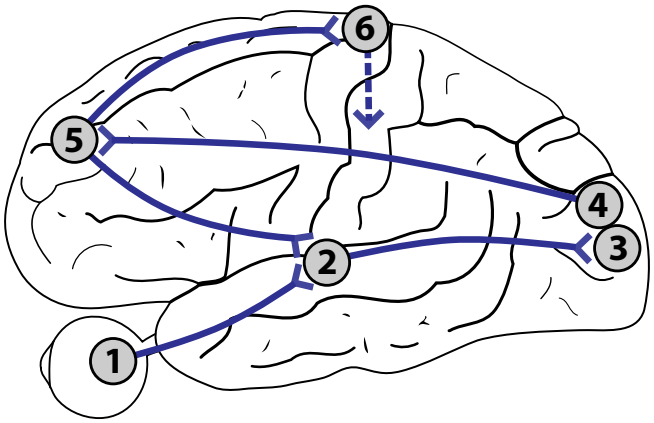

Figure 3.1: The architecture of the approach: (1) The silicon retina chip which serves as retina (2) The attention method of the OrientationCluster which serves as lateral geniculate nucleus (LGN) (3),(4) The OrientationCluster and the PerspecTransform which serve as visual cortex (5) The HingeLineTracker and the HingeLaneTracker which serve as prefrontal cortex (6) The FancyDriver which corresponds to the motor cortex from where the commands are sent to the motoric devices.

5. **Prefrontal Cortex - *HingeLine/LaneTracker*** From the visual information a line is extracted. Since this is done in a supervised, so to say cognitive way, the prefrontal cortex as home of the executive functions is considered the most appropriate spot to 'place' this algorithm.

6. **Motor Cortex - *FancyDriver*** The information about the line(s) has to be directed to the motor devices which are in an animal the muscles and in our case the electro motors. The analog for the FancyDriver is therefore the motor cortex.

3.2 The TmpDiff Chip

If we observe the world with our eyes, we do this with the rods and cones on our retina. Both types of these photo receptors show an adaptation behavior which follows in case of the cones a logarithmic behavior [3]. This adaptive loga-

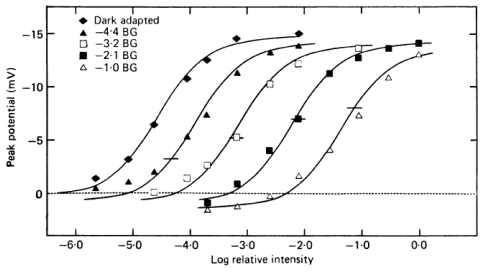

Figure 3.2: Dark- and light-adapted intensity-response curves recorded in a red cone. From [3] ©Blackwell Publishing

rithmic behavior was implemented in the retina chip by using the properties of MOSFET-transistors in subthreshold. This adaptive behavior makes that a pixel, which represents a photo receptive neuron, only sends out signals - so called spikes - if it observes an change in light intensity. Usual pixels just store the light intensity and have then to be read out by a 'scanner', this is not how it works in our retina and also not in the retina chip. In the TmpDiff chip a pixel autonomously sends a spike to the USB-port without any supervised scanner. Furthermore the chip distinguishes, like the bipolar cells in our eyes, the polarity of the events that happen: If the light intensity changes from dark to bright, it sends out an 'ON' spike and if changes from bright to dark an 'OFF' event is sent out. So the output of the camera are events which contain when (timestamp), where (coordinates) and what kind (polarity) of brightness change happened.

3.3 The OrientationCluster Class

To consider only the events that might come from a line, each event has to pass a line selective filter. To build such a filter one has to find a property of an event that correlates to the fact that it is originating from a line. This property is in our case that the orientation of an event from a line is similar to the one of its spacial and temporal neighbors. The development of this class was based on the preexisting SimpleOrientationCluster class [4].

The VectorMap Array

In order to handle the orientation of an event, a vector-array (`vectorMap`) is used that containes 'orientation cells' which stores the timestamps, the orientation (θ) and the corresponding intensity (i.e. vector length) of the most recent incoming events . This array corresponds to the orientation columns that were discovered in the cat as well as in the monkey by Hubel and Wiesel ([7] and [8]). So the length of the vector would be the firing strength and the orientation its receptive field. To calculate the orientation, meaning the vector belonging to an event, the algorithm iterates through the neighboring orientation cells within the receptive field (defined by `width` and `depth`) of the orientation cell to calculate it. In each step the distance vector (from the actual event to the neighboring orientation cell) gets decayed by an 1/t function (its slope can be adjusted by `factor`) and then the normalized x- and y-components are added up. The resulting total components are used to extract the angle (θ) and the length of the ori-

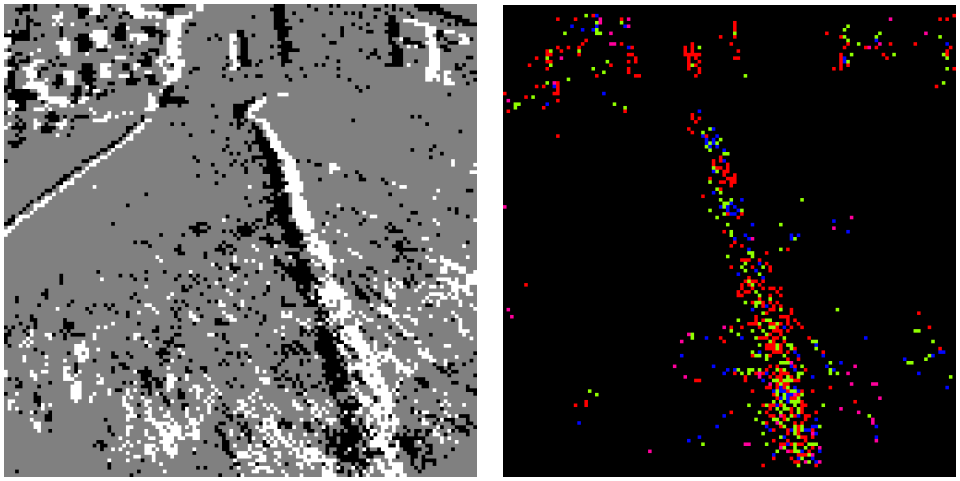

Figure 3.3: The effect of the OrientationCluster: On the left an image of a lane as it comes from the retina chip and on the right the result of the filtering by the OrientationCluster. The colors are distributed the following: Theta from zero degrees (vertical) to the ori-angle (in this case 45°) - red, green, blue, purple.

entation vector. To avoid that the two components of a line above and below an event get subtracted, all coordinates of orientation cells below the event get point inverted, so all orientation vectors have a positive y-component. Additionally one can choose if the VectorMap entries of the opposite polarity should be used or not to calculate the orientation of an event (`useOppositePolarity`). If they are used their coordinates get rotated by 90° (in a way that the y-component afterwards is positive) because the contrast gradient is rectangular to the course of a line.

The Selection

To decide whether an event can pass, the orientation of the neighbors also have to be known. Therefore during the neighborhood iteration steps described above, the vector components of the respective orientation cells in the VectorMap are added up to a neighborhood vector (with `neighborX`, `neighborY`, `neighborTheta`, `neighborLength`). The actual selection then consists of three criteria:

- **The Orientation Difference:** The difference between the orientation of the event and the orientation of the neighborhood vector has to be lower than a given `tolerance` value. This ensures that the orientation of the event and its environment are the same as it is in the picture of a line.
- **The Right Orientation:** The orientation angle θ has to

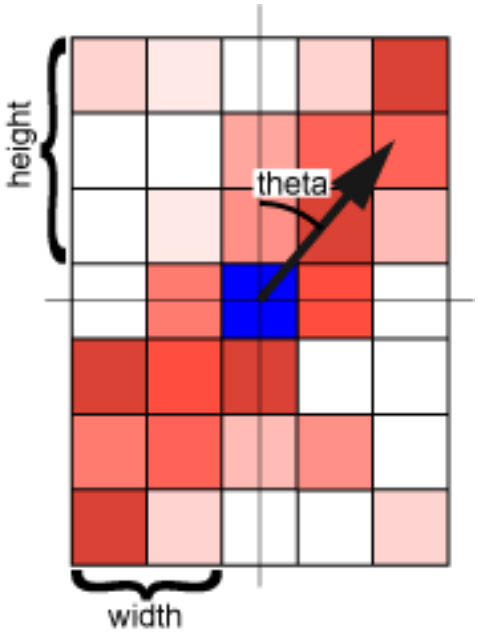

Figure 3.4: The orientation vector of an event (blue): The length and angle are determined by iterating through the neighborhood (given by `width` and `length`) and by adding up the normalized x- and y-components of the time decayed position vectors of the neighbors. The darker the red, the more recent the timestamp, i.e. the more important (longer) the position vector. White means that the timestamp of the most recent event with this coordinates was below the time threshold `dt` and so it does not account for the total vector.

has to be smaller than a given `ori` angle so that horizontal lines, which are not interesting in a line-/lane-following task, are filtered out.

- **The Length Of The Neighborhood Vector:** The neighborhood vector length has to satisfy a certain threshold (`neighborThr`) to filter out single events without any neighborhood activity and events from a noisy environment, that have random events all around them which lead in summation to a small vector.

Additional Features

The outputs of the filter are OrientationEvents which can have a discrete orientation value from one to four which is translated into a color code. The value is relative to the range in which the orientations have to be: from zero degrees (vertical) to the maximum value (`ori`) the colors are: red, green, blue, purple. This allows to use all four possible colors instead of sorting some of them out.
By activating textttuseAttention the vector length of an orientation Vector gets enlarged by the `attentionFactor` times the attention from the HingeLine/LaneTracker. This allows some kind of concentration by feeding back the result of the tracking algorithm to the filter (like denoted under 2 in the architecture).
Similar to the SimpleOrientationCluster [4] one can also turn on a function that allows the previous orientations to influence the actual orientation (`oriHistoryEnabled`) by a certain factor (`oriHistoryFactor`)

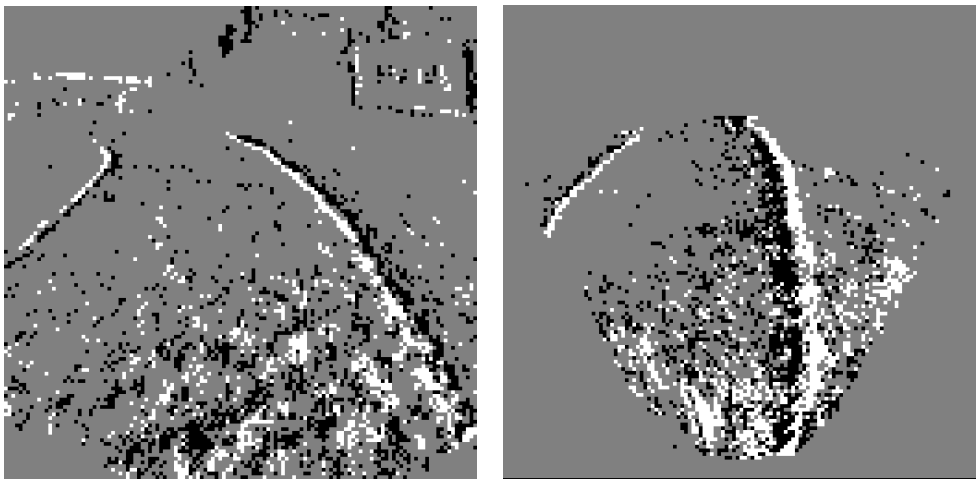

Figure 3.5: The effect of the PerspecTransform filter: An perspective, lens distorted image as it comes from the chip on the left and on the right a perspec-transformed image where this distortion is corrected and the events from above the horizon are cut away.

3.4 The PerspecTransform Class

If one wants to extract steering commands from visual inputs, the distortions that arise between the original image and the 'seen' image have to be considered. In our case there were two main sources of distortion which we tried to knock out with the transformation filter PerspecTransform:

3.4.1 The Perspective Distortion

Since the camera is not mounted perpendicular to the lines that it records, there is a distortion originating from the angle in which the camera looks on to the road. This distortion is linear, which makes it easy to handle and what is done, is linearly squeezing the picture from top to bottom by a given `ratio`. So with stronger squeezing (smaller `ratio`) one can account to flatter angles between the camera and the observed surface. This squeezing is done in the algorithm by setting up a look-up table that tells each event how far it has to be shifted.

3.4.2 The Lens Distortion

There is another distortion that originates from the lens. A first self made implementation of a transformation algorithm was not successful so the approach of Vass and Perlaki [5] was used which is built on the model of Sassenberg [6] and assumes a first order radial correction (adjusted with `k1`).
With the knowledge of how the

image gets distorted we set up look-up matrices which contain the amount of distortion in the x- and y-direction. This technique of redirecting the input via an existing structure (look-up matrix) not only represents the interconnection of two neural layers, it is also a very powerful and inexpensive method of calculation.

3.5 The HingeLineTracker Class

Before the development of an algorithm that tracks a lane which consists of two borderlines, the problem was simplified to one line and this lead to the implementation of the HingeLineTracker:
After sorting out the uninteresting events, the remaining ones have to be assembled to a line. A first neuroinspired approach should, by an inheritance method, assign to each incoming event a fuzzy value that represents the probability of being part of a line. This approach did not work because the right inheritance was hard to adjust which made the algorithm instable and after diverse approaches of fixing it, it was jettisoned. The actual algorithm is a supervised one that is no more event driven but it still has somehow neuroinspired structures. What it does is the following:

3.5.1 The AccumArray Object

Instead of assuming the line to be straight (as it was done in the existing HoughLineTracker approach [2]), it is described as a series of hinges. To place these hinges, events that are within some horizontal slices (number of silces: `hingeNumber`, lowest slice location: `bottomHinge`, highest slice location: `topHinge`) are directed to row cells (stored in the `AccumArrray`) where their intensities get added up and decay with time (controled by `hingeDecayFactor`). Each row cell in the AccumArray gets inputs from a certain `width` and `height` within a slice so the AccumArray is a kind of selective subsampler.

3.5.2 The Winner-Take-All Algorithm

To set a hinge for each slice the coordinates of the row cell with the highest activity are chosen to set the hinge. This maximum function corresponds to a winner-take-all (WTA) circuit which is considered to play an important role in the brain's way of computing [9]. To really set a hinge, the activity has to be higher than a threshold value (`hingeThr`) to avoid that even smallest activity due to noise can lead to a hinge (mis-)placement.
If no row cell of an hinge reaches the threshold the hinge is sent to a

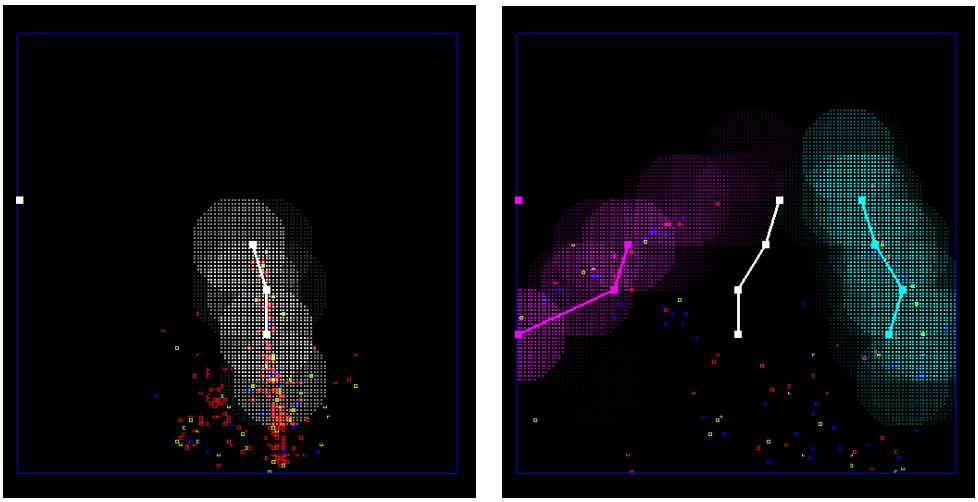

Figure 3.6: The HingeLineTracker (left): The events that passed the OrientationCluster and the PerspecTransform filters are used to set up a line approximation with hinges which are denoted by the white squares. The the activity in the highest slice is not high enough to set the highest hinge which is waiting on the left border of the image. The white surrounding of the line approximation is the attention: the whiter the higher the attention that is paid to a certain pixel. The HingeLaneTracker (right): The concept is the same as in the HingeLineTracker but white denotes the hinges of the seperator and the borderline approximations are pink and turquoise.

waiting state until a cell reaches the threshold (it is no PArt Of a LIne, so `isPaoli`=false)

3.5.3 The Concept Of Attention

To make the line prediction algorithm more stable, attention was introduced: Each pixel has an attention value which decays with time (`attentionDecayFactor`) and which directly influences how strongly an event on this pixel excites the corresponding row cells (tuned by `attentionFactor`. The attention can be increased in two ways: Either a pixel is within a `attentionRadius` of where an already set hinge is predicted to be if one continues with its lateral movement or it is predicted to be near a possible hinge which is predicted by an extrapolation of two already placed hinges. The attention around an already existing hinge makes the hinge less dependent on the actual activity because events with attention activate the row cells more, even if they are less than events without attention. By placing the attention on spots where a missing hinge is predicted the line prediction gets completed and extended to regions with less activity.

An additional mechanism to avoid instability is the condition that an already set hinge can only jump from one position to another

within a `shiftSpace`. This makes hinges stay on a already predicted line, even if there would be more activity outside of it.

3.5.4 The Output

The HingeLineTracker delivers two values:

- **The Phi-value:** This value is an weighted average of the angles between the hinges where the angles count more the higher they lay in the image. The first angle counts once, the next higher twice and so on. This has the following purpose: The higher a hinge, the further away and the more important it is for the future. Weighing the future higher leads to an anticipating error estimation. The higher hinges are also more reliable since they are affected by less noise less if the horizon is cut away properly. If phi is 0, the line is vertical and if it is 1 the line is horizontal. Positive values come from a road pointing to the right (top hinges more right) and vice versa.

- **The X-value:** The second value which is produced as output is the averaged x-position of the hinges. The averaging leads to a more stable behavior because the values of misplaced hinges get dampened. The value ranges from -1 at the full left to 1 on the full right.

3.6 The HingeLaneTracker Class

The Hinge Lange Tracker can be used to let the RC-car follow a lane which consists of two borderlines. The main difference between the LineTracker and the LaneTracker is the fact that there are two series of hinges (in the `hingeArray`: even numbers for the right, odd for the left line). This leads to the problem of assigning events to the two lines, so to simplify the task, the image is split by a mobile and flexible separator.

3.6.1 The Separator

Incoming events to the right of this separator only contribute to the accumArray of the right hinges and the other way round. Since the line is not always at the same location in the image, the separator has to be adjusted all the time. If the attention of one line is coming close to the separator, the separator gets pushed away until the attention of both lines is equal. So if for example the lane is leaving the image to the left and the right borderline is shifting more and more in this direction, than it pushes the separator to the left until all hinges have left the frame and all events

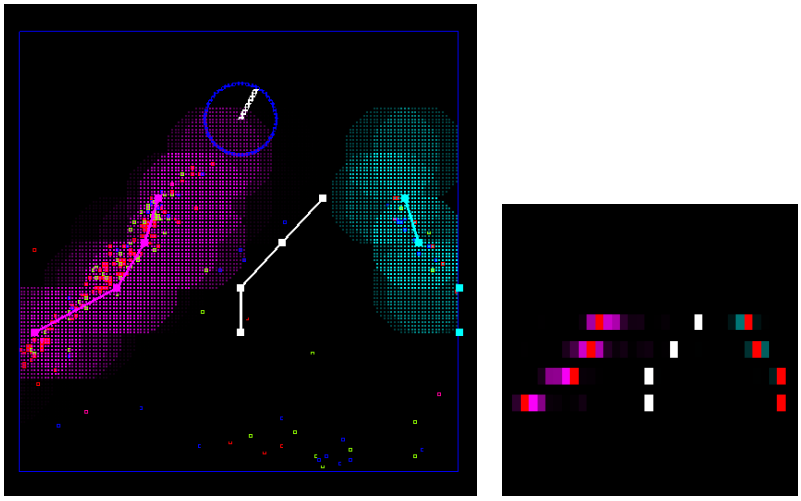

Figure 3.7: The FancyDriver (left): the visual output is similar to the line trackers but in addition a steering wheel denotes the actual steering angle. On the right the AccumArray, which represents the activity of the row cells, is shown.

excite the row cells of the right line. Like this the left row cells do not get excited anymore and the left hinges stay unset. So if the right line moves to the right again while the right hinges follow it, the separator follows as soon as the right hinges passed the middle of the image.
Additionally the separator slowly (dependent on number of input event packets) slides into the middle if neither the right nor the left hinge are part of a line. If a hinge is placed in between two hinges, it gets pushed out.

3.6.2 Entering Or Leaving Lines

In the case that the whole lane is leaving the image, the algorithm is sent into a waiting state (`isWaiting`) which makes the hinges align at the side where the lane left the image and it also makes the events of the other half of the image not excite the row cells until the hinges are set again.
To avoid that low hinges of a line, which in the lower part already partially left the image, are set to some noise, they are considered as not being part of the line as long as an upper hinge is close to the frame of the image.

3.6.3 The Output

Since there are two lines, the output is now coming from both line approximations and instead of averaging the x-values of the hinges only over one line, it is done for the two, which results in about the middle of the lane. The phi is handled

similarly: because we are interested in the curvature of the whole lane. It is possible to draw the output-values in form of a line (activated by `drawOutput`.

3.7 The FancyDriver Class

The output of the HingeLine-Tracker or the HingeLaneTracker is used to extract an error about where the car is at the moment and where it should be. The ideal state where the error is zero is the one where the phi- and the x-value are zero. To reach this state two controllers are implemented: a PID-controller that works with the actual error (P), its derivative (D) and the integrated error (I) and a LQR-controller that works with the phi-, the x-value and their derivatives. These controllers are further described in the companion Bachelor thesis by Robin Ritz.

4 Results

To see the algorithms at work there are some movies on the attached CD which show some exemplary performance of them on a test-track.

4.1 The OrientationCluster Class

With the right parameters the OrientationCluster was able to filter out the events originating from a line. This highly increased the signal to noise ratio which allowed an increased performance of the HineLine/LaneTracker. The movie on the CD shows how much of the noise, i.e. events that do not originate from a line are sorted out. It is not possible to sort out all noise because not matter what criteria is chosen as filter-criteria, there always will be some statistical outliers that fulfill them.

4.2 The PerspecTransform Class

The PerspecTransform helped to tackle the distortion problem. It is still not working perfectly but it made the output values of the lane trackers, especially the phi value, more reliable. The transformation was set up to avoid that the phi values of a line which is away from the image center, don't point away from the line in reality (because of the perspective narrowing) and this goal could more or less be reached. The problem that still remains, is the one of tackling the lens distortion where the transforming filter may have improved the image but it is still distorted - especially for the wide angle lens.

4.3 The HingeLineTracker Class

If the two filters in front of the HineLaneTracker were tuned the right way, it was able to track a line in a robust way. Thanks to the built-in attention it could complete an uncomplete line prediction and it was also able to ignore noise even when its activity was higher than the one of the line.

4.4 The HingeLaneTracker Class

The HingeLaneTracker ran stable and was able to track the two borderlines unless the lines did not leave the image or suddenly changed the lateral position. If the algorithm once lost the lines it had almost no chance to find them again in the right order. If the car drove slowly the tracker was able to stay on the lines but sudden changes e.g. narrow curves often made the algorithm mixing up the lines. Even though the algorithm already has some exception handling this couldn't handle all the variations of exceptions but at least the separator hinges could be set to a satisfying behavior.

4.5 The FancyDriver Class

After some fiddling around with the interface between the trackers and the FancyDriver the error needed for the controlling model could be extracted from the trackers and sent to the driver. Some more fiddling around with the driver parameters then allowed following the line and the lane. In the computer model the lane could be followed at moderate speed and the line even at some higher speeds. In reality where we draw some lines on the asphalt ground behind the INI the lane could be followed if the car was pushed manually and for the line tracking it was possible to let the car drive at moderate speed (a little more than walking pace).

5 Conclusion And Outlook

5.1 Conclusion

The project showed that the camera certainly has the power to simplify machine-vision tasks in a way that less computational resources have to be used. It also demonstrated that it is possible to use the events of the retina and process them in a neuroinspired manner.
Unfortunately the car following a lane could only be realized in the virtual model and for the reality it only worked when the car was slowly pushed manually. This and the fact that it followed the line only at moderate speed have two main causes:

- **The small visual field:** The camera only recorded a small part of the street in front of the car which narrowed the range of positions and orientations that 'see' the line(s). This made it hard to control the steering - especially at high speed - because already small failures led to a loss of the line(s). The wider lens could also not solve this problem because it introduced others.
- **The wasteful parameter-tuning conditions:** Since it was not possible to do the parameter-tunig in real-time, it was a very costly process and so we couldn't adjust the algorithms and parameters to an optimal performance.

5.2 Outlook

To increase the performance and its stability, i.e. to make the make the car run faster and longer, there are several areas one should work on:

The OrientationCluster

The OrientationCluster works quite well but it requires a lot of computation time, so the calculation steps might be optimized. For example by subsampling the events as it is done in the SimpleOrientationFilter.
It would also be useful to implement a learning algorithm that adjusts the parameters of the filter to the light and contrast conditions.

The PerspecTransform Filter

Since the image is still distorted after the transformation of the PerspecTransform Filter, one would have to think about implementing

second order corrections or a totally new transformation approach. If the actual approach is not replaced, one would should fix it and avoid the shrinking of the image for high k1-values.

The HingeLineTracker

If the line once left the image the HingeLineTracker should somehow try to retrieve it but this couldn't be implemented yet. Instead of retrieving the line the hinges track some noise which makes it impossible for the FancyDriver to recover it.

The HingeLaneTracker

The HingeLaneTracker would need some more elaborate exception handling to avoid the mixing up of the lines or to increase the retrieving rate of lines that left the visual field.

The Parameter-Tuning And Test-Running

If one wants to make the car driving at really high speeds it would be very useful to make real-time parameter adjustment possible which would heavily decrease the error recognition and removal time and the fine-tuning. One should also respect that even for a human it is impossible to react on narrow curves at high speed with such a small visual field, so the curves should be rather wide and this would mean that big testing area is needed.

Additional Devices

A second camera with a wider field could make it easier to see the line and stay on it. A local GPS-system could be used to retrieve the track after a loss or it could be used to train an artificial neural network approach. A higher resolving Retina could be used with a wider lens to see more of the lane.

A Acknowledgements

I would like to thank Robin Ritz for the very nice collaboration, Tobi Delbrück for the supervision of the project and the support, Thomas Besselmann for the inputs and support, Patrick Lichtsteiner for the support, the intense test-driving and the following construction of a new camera mount and Rodney Douglas for being official godfather of my bachelor project as grading professor.

B Attachment

On the attached CD is some of our work in digital form:

B.1 Code

The folder Code contains the latest version of the code we developed.

B.2 Movies

Each folder contains a movie that shows the performance of the algorithm on a example sequence compared to the unprocessed sequence, the raw data (recored AER-events) in form of a .dat file and the settings of the algorithm saved in a .xml file. This allows a reproduction of the result in the jAER-viewer.

OrientationCluster

The sequence to demonstrate the effect of the orientation cluster was recorded with a chalk drawn lane on the 11th of March 08 behind the INI on dry street ground. The sequence was recorded during the car was manually driven along the lane. During the development phase of the algorithms this sequence served as template to test the developed code.

PespecTransform

This sequence which demonstrates the PerspecTransform performance was recorded under the same conditions as the one of the Orientation-Cluster - it also served as development template.

HingeLaneTracker

Also this sequence was recorded under the same conditions as the other two and also served as development template.

FancyDriver

This sequence was recorded on the 15. Jun. 08 on the street behind the INI. The reason for the slow driving at the end, is the fact that the power packs run dry.

B.3 Blueprints

camera_mount.pdf

This is the laser cutter blueprint for the second version of the camera mount. The upper parts were used to screw on a hinge and the lower one to screw on the camera.

screw_nut.pdf

This is the laser cutter blueprint for the screw-nut holder in the the second version of the camera mount.

PC_box.pdf

This is the laser cutter blueprint for the protective box of the micro PC. the wholes for fresh air and for charging the pc are missing because they were drilled afterwards.

AccSensor_mount-pdf

This is the laser cutter blueprint for the acceleration sensor mount which allows to screw on the sensor in different directions and on different positions.

B.4 Literature

This folder contains all referenced literature numbered the same way as in they were referenced.

B.5 Presentation

This folder contains the slides of the project presentation in the hardware group meeting at the INI on 19. June 2008.

Bibliography

[1] P. Lichtsteiner, C. Posch, T. Delbrück, A 128x128 120 dB 15 μ s Latency Asynchronous Temporal Contrast Vision Sensor , *IEEE Journal Of Solid State Circuits*, 2008, VOL. 43, NO. 2

[2] Available: http://jaer.wiki.sourceforge.net/

[3] R.A. Normann, I. Perlman, The Effects Of Background Illumination On The Photoresponse Of Red And Green Cones, J. Physiol., 1979, 286, pp. 491-507, Authorized to print figure 7.

[4] T. Delbruck, Frame-free dynamic digital vision , Proceedings of Intl. Symposium on Secure-Life Electronics, Advanced Electronics for Quality Life and Society, University of Tokyo, Tokyo, Japan, Mar. 6-7, 2008, pp. 21-26.

[5] G. Vass, T. Perlaki, Applying and removing lens distortion in post production, The Second Hungarian Conference on Computer Graphics and Geometry 2003, Budapest, available: http://www.vassg.hu/pdf/vass_gg_2003_lo.pdf

[6] U. Sassenberg. "'Lens distortion model of 3DE V3"', available: http://www.3dequalizer.com/sdv_tech_art/paper/distortion.html (2001).

[7] D.H. Hubel, T.N. Wiesel, Shape and arrangement of columns in cat's striate cortex, J. Physiol. 1963, 165, pp. 559 - 568

[8] D.H. Hubel, T.N. Wiesel, Receptive fields and functional architecture of monkey striate cortex, J. Physiol. 1968, 195, pp. 215-243

[9] W.Maass, On the Computational Power of Winner-Take-All, Neural Computation 2000, 12, 2519-2535

[10] Available: http://siliconretina.ini.uzh.ch/

[11] Available: http://www.sonystyle.com/webapp/wcs/stores/servlet/CategoryDisplay?cata
10551&storeId=10151&langId=-1&categoryId=577&parentCategoryId=16154

List of Figures