Chain of Small Robots

PRACTICAL COURSE COMPUTATIONAL NEURO ENGINEERING

submitted by
Dominik Luber
Johannes Biedermann

NEUROSCIENTIFIC SYSTEM THEORY
Technische Universität München

Supervisor: Nicolai Waniek
Final Submission: 14.01.2015

In your final hardback copy, replace this page with the signed exercise sheet.

Abstract

The goal of this project is the improvement of the software for a small autonomously controlled robot, so it is able to follow another robot in front of it, which has a LED mounted on its top. This following is realized by using a tracking algorithm on the images of the flashing LED, which are collected by an on-board eDVS camera. With the resulting data, the robot commands and hence the driving trajectory is calculated.
By setting multiple of these small robots in a line, a chain of robots can be created, where only the first robot is controlled remotely and all others are following autonomously.

Contents

1 Introduction

Autonomous Systems are gaining more and more space in the modern society. Especially in logistics and public transportation, computer leaded and controlled vehicles are already very common today. With the increasing amount of different available sensors, fusion and processing of these huge data sets are getting increasingly important.
Further, the calculation of results out of the sensor data in real-time is especially for autonomous systems crucial. Besides of increasing the calculation power of the devices, the development of new algorithms provide good opportunities to manage all data in real time. The probably most promising options are biologically inspired algorithms and sensors. By using architectures and methods we learned from nature, a variety of new different application areas can be discovered.
In this project, a biologically inspired image processing algorithm [1] should be implemented on a small mobile robot. This robot should then compute the sensor data it gets into driving commands and thus follow a blinking LED. This LED is mounted on top of each of the robots. So they can either follow another robot or are controlled remotely via Wifi. The resulting chain of robots is therefore only controlled remotely by the first robot and all other robots are following autonomously, calculating their following trajectory on-board and in real time.

1.1 State of Art

The given hardware included three working pushbots with eDVS cameras and Wifi modules, as described in chapter 2.
The software part contained a Java program and simple command line tools, which allowed a Wifi connection for streaming the eDVS images and sending simple commands. Also the software, which is running on the microcontroller of the eDVS-board.
Further, some C and Matlab code was given, which should organize a working chain of robots via a cable connected UART-port, but did not work properly in our tests.

1.2 Goals

The resulting goals were in general a working chain of three or more robots, where the leading robot is remotely controlled via Wifi. The steering inputs should be given by an ordinary gamepad, transformed to robot commands, and then send via Wifi to the pushbot. The software-sided goals were a stable working following and tracking algorithm. In addition it should be possible to add or remove robots on the fly.

2 Hardware Description

The pushbot robots are equipped with a frequency-adjustable LED on top, an eDVS camera board, including the microprocessor on the front and a Wifi module connected on the back of the motherboard. Further, the motherboard is mounted on a platform which includes the motors, the gears and the power source, consisting of four rechargeable AA batteries.
Figure 2.1 and 2.2 are showing annotated pictures of the robot. The leading robot is connected via Wifi to a standard PC, running the connection program (either in Matlab or as command line tool), to which a gamepad controller is connected via USB.

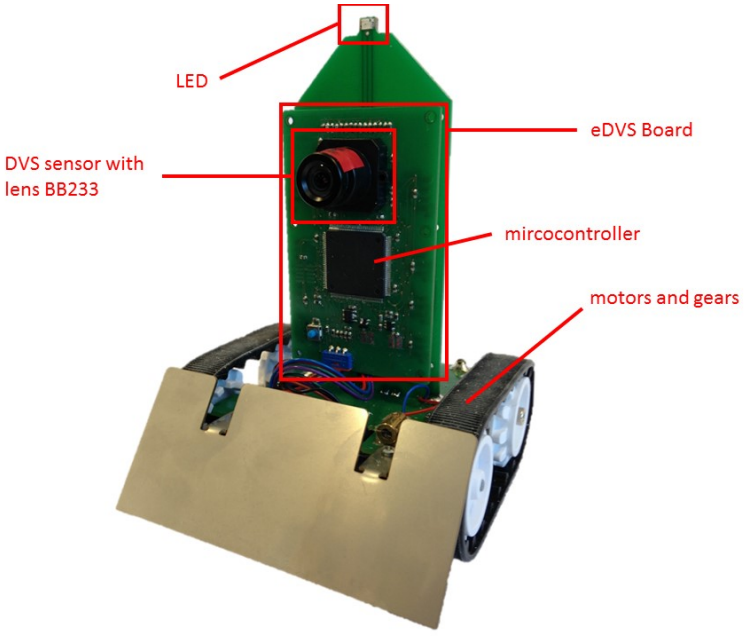

Figure 2.1: The front of a pushbot robot, consisting of a LED, eDVS-board with sensor, microcontroller, motors and gears.

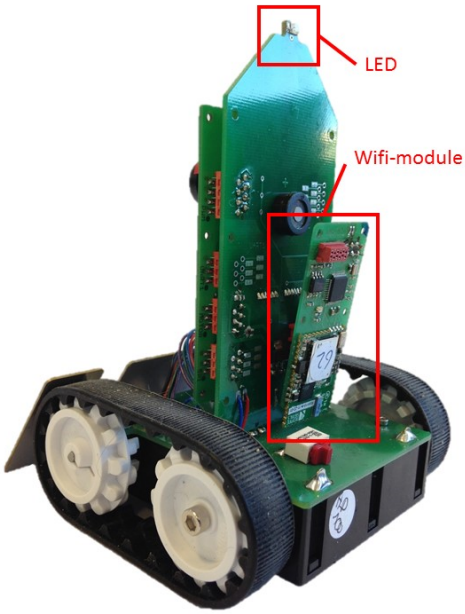

Figure 2.2: The back of the robot, also has an LED, as well as the Wifi-module.

2.1 LED Marker

The LED on top of the motherboard (height above ground: $13cm$) consists of two red-shining LEDs, one frontwards and the other one backwards pointed, which can be adjusted independently both in frequency as well as in blinking duration.
In our case, tests have shown that the frequency of $1kHz$ is the most suitable for our tracking purposes under normal light situations at a distance of 20cm between the LED and the later described eDVS camera. The blinking duration was set to 50%, so that the intervals when the LED is active have the same duration as the intervals when the LED is dark.

2.2 The Embedded Dynamic Vision Sensor Board

The Embedded Dynamic Vision Sensor Board consists in general of the DVS chip and the microcontroller. The Dynamic Vision Sensor Chip (DVS, LD08) has an 128 x 128 pixel resolution and responds to relative changes in intensity. A dynamic range operation bigger than $120dB$ and an event latency of $15\mu s$ are the details to be named in this context [2].
On top of the sensor, a lens is mounted. For our purposes we used the type BB233, with F2.0 and $3.6mm$ focal length. It provides a good aperture of approximately 40^o, which is crucial for following in curves by having the LED focused at a distance of $20cm$. Using a lens which focuses on larger distances would result in having smaller

apertures which cause massive problems in following robots doing curves. Our tests evidenced that this lens was the perfect trade-off between large aperture and large focus distance.
The microcontroller (model: NXP LPC4337) has two 32 bit Arm Cortex cores with up to $204MHz$ speed and a SRAM memory of $104KB$ in size. The microcontroller is the most important piece of the whole robot: it processes the images from the camera, calculates the following trajectory and sends steering commands to the motors.

2.3 Wifi Module

To control the chain of robots, at least the leading robot needs a Wifi module to receive the steering commands. The Wifi module is, as it can be seen in figure 2.2, mounted on the backside of the motherboard and connected to the microprocessor. By establishing a TCPIP-Connection from the PC on which the later described gamepad is connected to the Wifi module, commands can be received and data streamed to the desktop PC for further analysis.
The use of a not perfectly fitting ribbon cable for the connection to the motherboard can sometimes cause connection aborts due to small vibrations which loose the plugs.

2.4 Motors and Gears

For moving the vehicle, the robot has two independently controllable motors connected to gears on its bottom. They allow the user to drive the robot in almost every desired direction. The gear reduction of the used motors is 75:1 and we have 70 velocity steps in both directions (for- and backwards).
Maximal speed is according to the documentation $85cm/s$ [2]. Due to stability reasons we only use 60 velocity steps.

2.5 Gamepad

As input device for the robot steering commands we use a Logitech Dual Shock Gamepad connected via USB to a standard PC, which can be seen in figure 2.3. For driving for- and backwards, the left joystick (also called longitudinal-joystick) is used vertically, for driving turns and curves the right joystick (lateral-joystick) is used horizontally.

Figure 2.3: Standard gamepad device

3 Software description

3.1 Tracking Algorithm

Our tracking algorithm is based on an already existing algorithm [1], basically using two circumstances of the LED, which has to be tracked.
The First one is, that the LED blinks at a given frequency, so events can be weighted according to the temporal distance between two events. The other one is, that a high frequent blinking light source produces a lot of events, so every event pulls the estimated tracking point to itself. Hence the more events are at a given area, the more they pull the tracking point towards them.

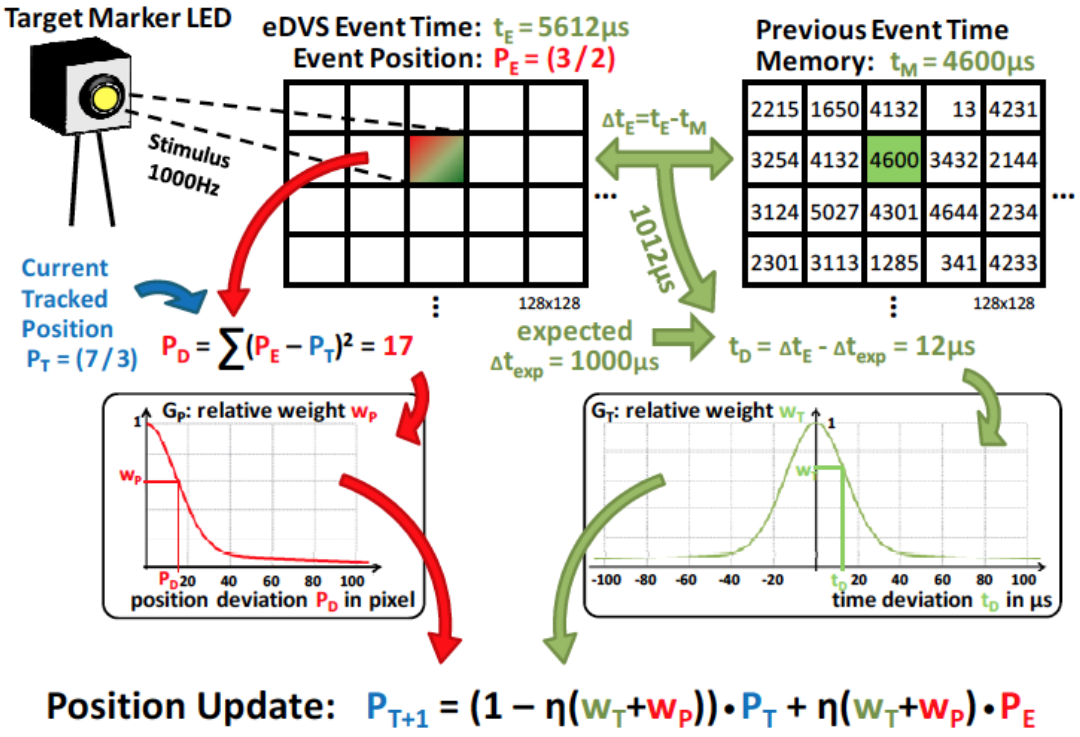

Figure 3.1: Graphical explanation of the tracking algorithm [1]

Position update formula explained in detail:
Here the points for a simpler notation stand for the 2x1 vector $[x\ y]^T$, which shows the x- and y-position in the 128x128 pixel picture, given by the eDVS camera.
P_{T+1} is the new estimated point to track, hence P_T is the old one and P_E is the point, where the current event was.
η is a factor between 0 and 1. For low values, the old tracking point has more influence, but new events can't pull the tracking point fast to them, so it gets more robust against noise, but less sensitiv for fast movements of the LED (respectively the robot with the LED on its top). For a high value of η, of course it is the other way around and new events have a bigger influence, therefore also noise.
w_T and w_P both are weighting factors, which are computed by taking the y-value of the respective gaussian curve at a certain x-value, which is determined by the temporal (t_D) or local (P_D) distance of the event.
The local distance P_D, naturally is the distance between the old tracking point and the new event. So a new event, which is very close to the tracking point is weighted higher, than a more distant event. Therefore a point source which generates a lot of events (LED with high frequency) pulls the tracking point to itself, rather than some noise at the other side of the picture.
For the temporal distance t_D, a 128x128 table (one entry for every pixel of the camera) is needed, where the timestamps of all events are stored, so a new event can be compared to an old event at this pixel and hence, a temporal difference with a value close to the estimated one, given by the LED frequency, will get more important.

3.2 Modified Tracking Algorithm

For our task, where a moving object has to track and follow another moving object, the above described algorithm had to be modified.
This is due to the facts, that it has to be fast enough to evaluate events in real time. Also the possibility exists, that the LED leaves the field of vision and thus can't be tracked. So the Algorithm has to know, if there is something it would like to track, or not. Otherwise it would lead to an unstable and uncontrollable behavior of the robots, once they would loose the robot in front of them.
For making the algorithm faster, the computation of the local distance between to points has been altered. In the original version, mathematically correct, it was calculated as the well known and therefore not in detail explained euclidean norm.
The problem is, that here a root has to be evaluated, which needs a lot of computation resources. So a faster estimation of the distance was to take the absolute value of both the x- and y-distance, get a x and y-weighting and multiply them both. This also enables to weight x- and y-distance differently, since changes in the horizontal level (x-axis) happen much faster than in the vertical (y-axis).
The next step was, to find out, whether a tracked point was actually the LED. Since

without an LED, noise or events from movements in the environment would pull the tracked point towards them, this is an important thing to make the tracking, as well as the following, stable.
To achieve this, the temporal weight factor had to be evaluated. Above a certain threshold, there was an LED with the desired frequency detected. Every time this happens, it gets counted. So in a given time window, if the LED was detected often enough, the algorithm knows that it is actually tracking the right LED. At the end of this time window, that count is reseted to zero and a new evaluation circle begins. However, the algorithm counts in how many of this time windows no LED was detected (so the temporal weight was beneath a certain threshold), thus at a defined value, there is nothing anymore to track. This count gets reseted to zero, every time a LED was detected within the time window

3.3 Following Algorithm

The following algorithm determines the whole movement behavior of the robot. Shortly explained, the tracked point attracts the robot, so it wants to center it in his view at a predefined point of sight (128x128 pixel).
To do this, it uses the knowledge of its tracking algorithm, which says, if there is a LED to track or not. If there is nothing to track, the motor speed of both chains is set to 0 and the pushbot stops moving. Otherwise, if there is something to follow, tables are used for determine further movements.
Here the values for both chains are computed in the following manner:

$$leftChain = maxMotorSpeed \cdot (yV + xV) \tag{3.1}$$

$$rightChain = maxMotorSpeed \cdot (yV - xV) \tag{3.2}$$

The variable $maxMotorSpeed$ has a value between 0 to 70 (chapter 2.4), which adjusts the relative speed values for both motors.
yV and xV are weighting factors in the range of -1 to 1. Both weighting factors are computed by taking values of the $yVTable$ (fig. 4.9), regarding to the y-postion of the tracked point and respectively of the $xVTable$ (fig. 4.8) regarding to the x-position.
The values of these table enable a smooth centering of the robot at a horizontal level and a predefined distance.

3.3.1 Delayed Following / Queue

Another thing, which should be implemented is a sort of queue, or time delay for the following robot, where the position of the tracked LED is saved with a predefined

sample frequency (fig.3.2). This should take care for a following behavior, which more or less copies the movement of the robot before them.

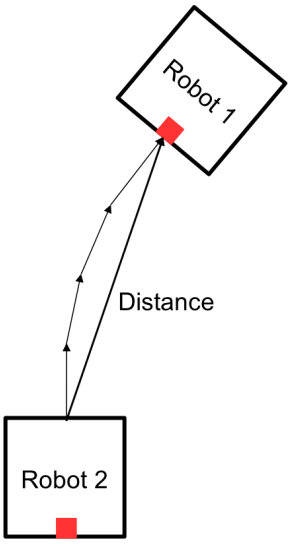

Figure 3.2: Scheme for estimating a better approximation of the actual trajectory

3.3.2 Chain Behavior

As already explained in chapter 1.2 and shown in figure 3.3, our aim was to implement a chain of robots. Therefore every pushbot has its LED activated with the same frequency, which is also the same it wants to track and follow. Thus the chain can easily be enlarged.

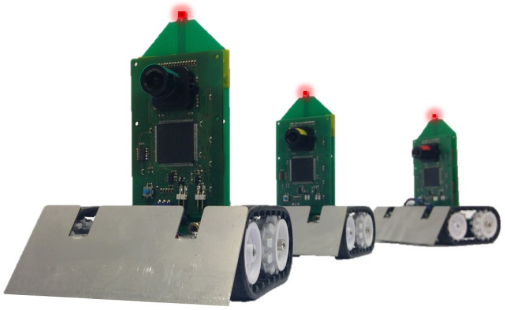

Figure 3.3: Chain of three pushbot robots

3.3.3 Changing Position Within The Chain

Another goal was, that a robot can be removed or inserted at any place of the chain. Therefore figure 3.4 provides a better understanding.
Since all the LEDs operate on the same frequency, robots easily can be changed.

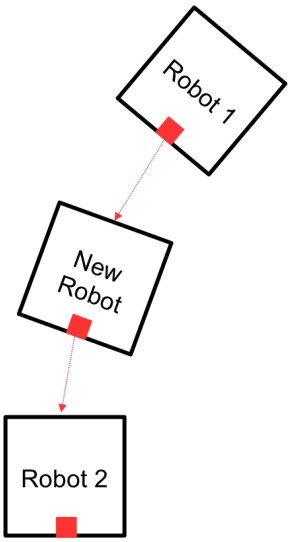

Figure 3.4: Scheme of a new robot getting inserted between two other ones

3.4 gorrc

The leading robot of the chain has to be controlled remotely by PC, therefore some of the given C-code was improved (similar to the MATLAB code) to steer the leading robot via gamepad. This was realized by collecting the signals given from both joysticks from the DualShock Gamepad Controller (chap. 2.5) and transforming them to commands for the two gears.
In the first versions we had one Joystick for each gear (similar to the tank-steering), so that turns could be driven by reducing the speed of a specific gear. But now we have one joystick for longitudinal movement and one for lateral movements. Because this enables a much easier and more natural handling.
The commands given from the right lateral-joystick are either added to or reduced from the commands of the left longitudinal-joystick to receive the specific gear velocity.

4 Results

4.1 Modified Tracking Algorithm

As previous described (chap. 3.2) our modified tracking algorithm evaluates incoming events and pulles the tracking point towards them, according to temporal and local weighting. Also it guesses, whether there actually is something we would like to track or not.
Figure 4.1 actually shows the first person view of a robot, tracking another one in front of it.

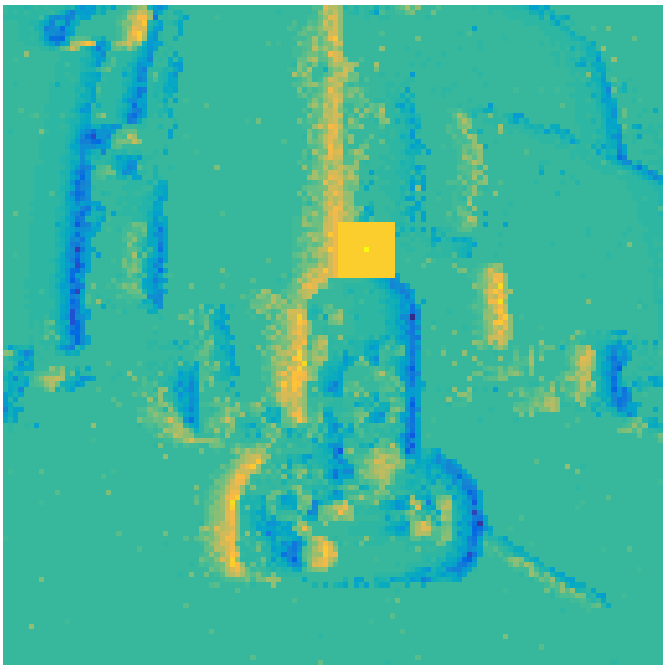

Figure 4.1: First person view of the pushbot, tracking another one

Therefore a lot of parameters have to be tuned. Some of them were received by analyzing data, but most had to be found by driving around with the robots, look at their behavior and find values, which suit its purpose in an acceptable manner.

4.1.1 Parameters "By Inspection"

LED frequency: $1kHz$ as already explained in chapter 2.1.

$\eta = 0.8$. At this rate, the tracking point is more pulled towards new events, rather than to the old tracking point. Although it gets more sensitive for noise, we need a high value, since fast movements of the robot lead to quicker changes of the LED's position and therefore, the updates have to follow accordingly.

Temporal weighting table $WtTable$ (fig. 4.2): formed by the positive side of a gaussian curve with the mean at 0 and a standard deviation of 3. So, only a small deviation around the desired frequency gets weighted.

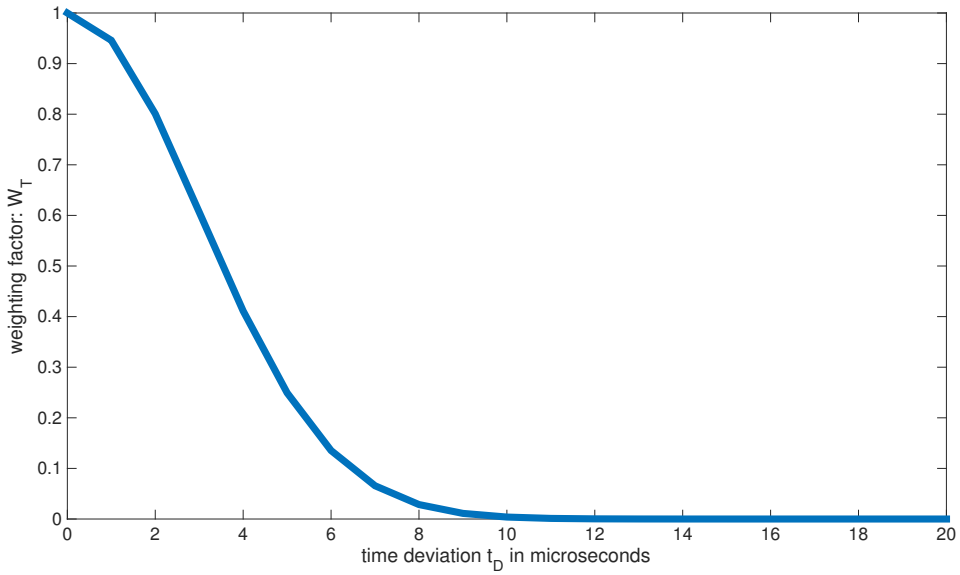

Figure 4.2: Temporal weighting table $WtTable$

Local weighting table $WpTable$ (fig. 4.3): Here a gaussian function was also choosen, which covers the whole range of the camera (128x128 pixel), with the mean in the middle and a standard deviation of 5.
However, since the vertical values change slower than the horizontal ones, at the computation, the y-distances get doubled before evaluation of the weighting factor, so changes have a smaller influence, than in the x-axis.

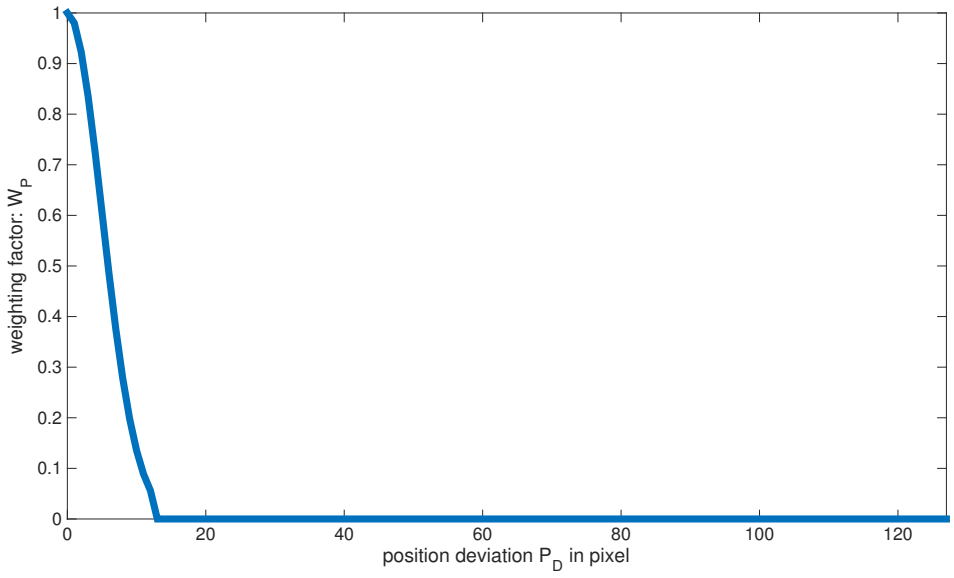

Figure 4.3: Local weighting table $WpTable$

Time window: the time window was set to $20ms$, this enables a fluent updating of the robot and also fits well to the value of $WtCount$ (as explained further below in chapter 4.1.2).

$lostTrackingCount = 10$. This yields after all in $200ms$ (10 times no tracked LED within a time window) to see quick enough, if the LED was lost for good, or it just got lost temporarily due to fast movements.

4.1.2 Parameters By Analyzing Data

$WtCount$: Here overtime is counted, when Wt (temporal weighting factor) was greater than 0 within the time window of $20ms$. Given the date (fig. 4.4 to 4.7) and by testing, we came to the conclusion, that after reaching a $WtCount$ value of 2, it yields the best performance.
In the following figures 4.4 - 4.7, the y-Axis always stands for counted $Wt > 0$ within the time window of $20ms$ and the x-Axis for the total time of the experiment. Since all four cases were observed sequentially within the same experiment, the time axis shows the total time values of the experiment.

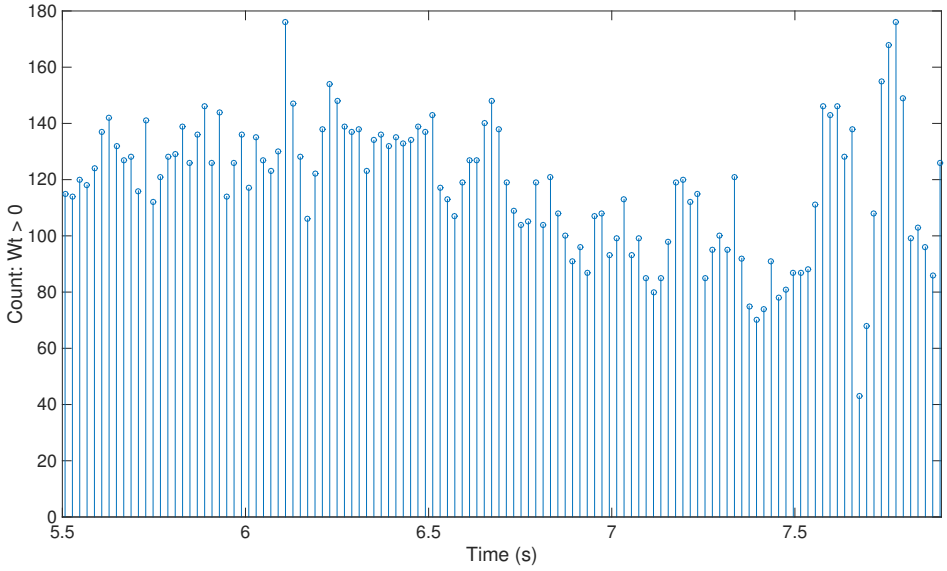

Figure 4.4: Here, the results for the first test case is shown: a stationary LED blinks with a frequency of 1kHz.

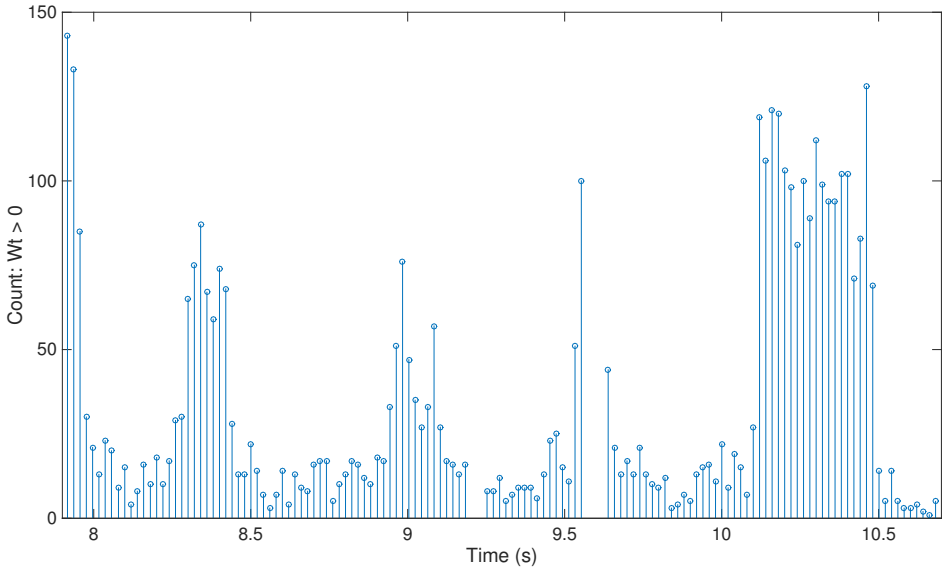

Figure 4.5: Here, the results for the second test case is shown: a moving LED blinks with a frequency of 1kHz.

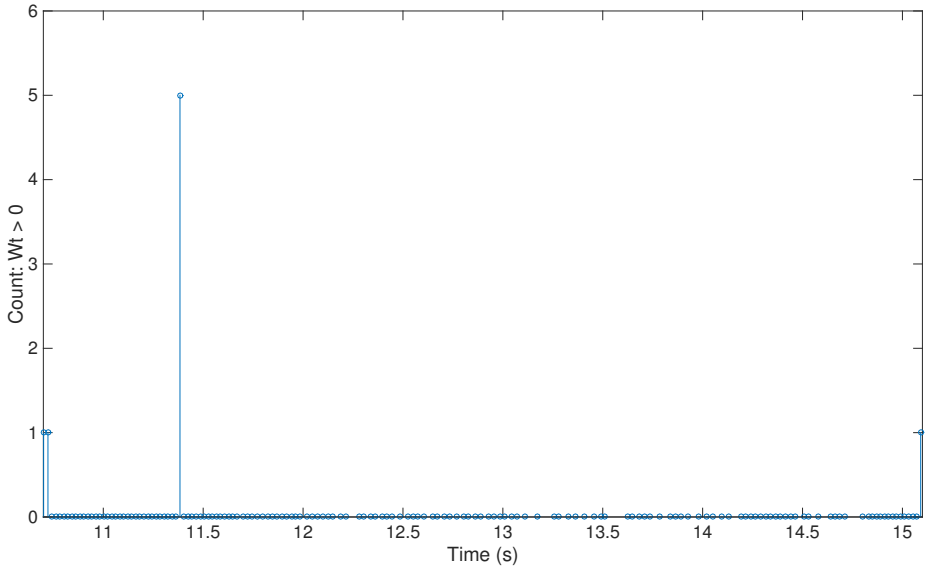

Figure 4.6: Here, the results for the third test case is shown: the camera stands still and sees no moving objects, so basically normal background noise is observed.

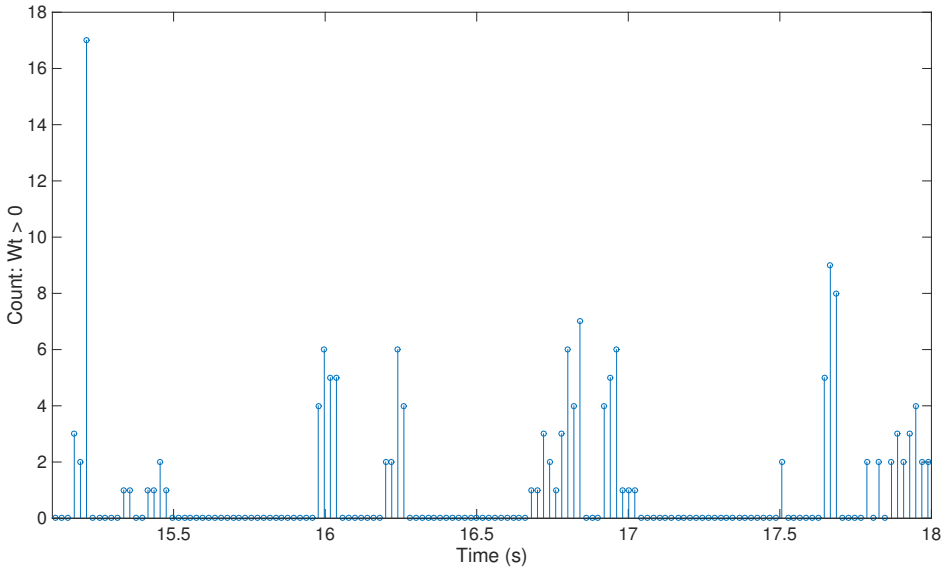

Figure 4.7: Here, the results for the fourth test case is shown: a mixture of moving camera and a waving hand in front of it generates many events, therefore some are mistaken for an LED.

As it was shown, the LED produces many events with a temporal weighting factor grater than zero (fig. 4.4), even while moving (fig. 4.5).
On the other hand, if there is no LED, nearly no such events are generated (fig. 4.6). However if there is no LED, but a lot of movement (fig. 4.7), some events are generated which can be mistaken for the LED, but since this only happens occasionally and not constantly over a long period of time, our modified tracking algorithm does not care enough about it to produce an unstable behavior in the robot.
After all the parameters were tuned in a fashion, where the tracking works stable and serves it purpose. It works at different light conditions, however better at darker than bright conditions. With different sources of noise: normal noise, waving hands between camera and LED, different frequencies from monitors or lamps in the background or simply by moving objects, which generate a lot of events.
In addition, as already stated, the algorithm got faster by computing the distance between two points in a more simple way. Tests in Matlab with different test data sets, with and without noise/movement/LED and different durations/number of events (data not shown) revealed a computation time, which is accelerated by roughly the factor of 5 in comparison to the original algorithm [1].

4.2 Following Algorithm

The behavior of the following robot is totally defined by both, $xVTable$ (fig. 4.8) and $yVTable$ (fig. 4.9). Both tables have a range of 1 to 128, covering every pixel of the cameras view and were acquired through a lot of tests via Matlab, where single parameters were tuned, so the following looks nice and smooth, but also takes care about the robot does his best to not loose the LED it is following.
In our experiments this allowed the robot even rather harsh maneuvers, like sharp turning or s-curves. However this does not hold, if the robot moves too fast, so the maximum Speed $maxMotorSpeed$ was set to 60, to guaranty a stable following. Again this value was found by testing.

$xVTable$ (fig. 4.8) is a parabola, normalized to a height of 1 and multiplied with a constant factor of 0.6. Also the left side got flipped around the horizontal axis. This is due to how the speed of the two motors is computed (eq. 3.1 and 3.2).
Also this constant factor was obtained by tests. For lower values, the robot does not turn fast enough and loses the LED. But for higher values, it moves too fast, so the behavior gets unstable.

$yVTable$ (fig. 4.9) consists of two gaussian curves with a mean of 45. This serves the purpose of holding the robot at distance of $20cm$, which did a good job while tests. The left gaussian curve has a standard deviation of 10, its y-values have been adjusted to be between -1 and 0. So the robot slows down or drives backwards, if it is too close.

The gaussian curve on the right side of the mean has a standard deviation of 3, and its y-values are between 0 and +1. In addition it is flipped around the horizontal axis. So if the distance to the robot in front gets too big, the robot quickly accelerates.

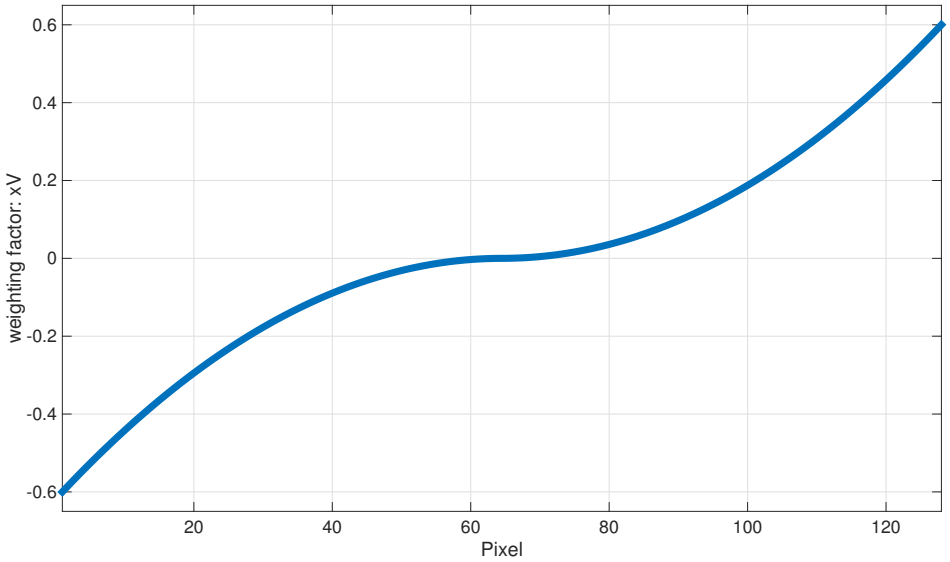

Figure 4.8: *xVTable* gives a weight, according to the x-position of the tracked point.

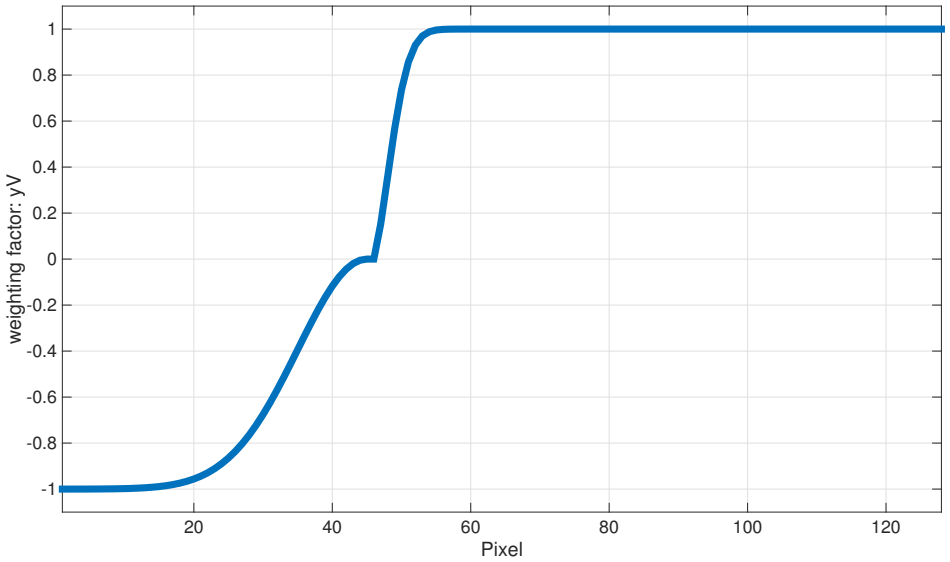

Figure 4.9: *yVTable* gives a weight, according to the y-position of the tracked point.

Since the hardware dependent maximum motor speed is at a value of 70, we get following speed distribution (fig. 4.10) for each motor, given the x- and y-position of the tracked LED:

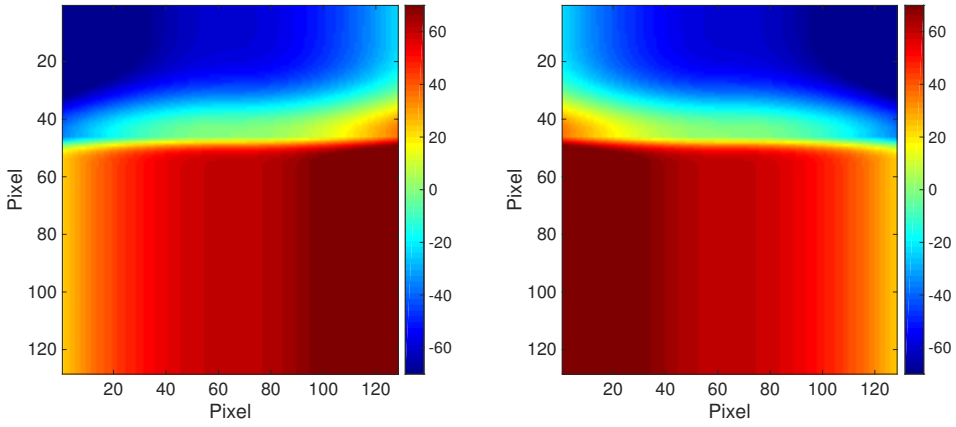

Figure 4.10: On the left: speed table for the left chain. On the right: speed table for the right chain.

As our test results have shown, this tables care for a nice following of the robot. The movements look mostly fluent and the trajectory of the leading robot is copied relatively well, having in mind, that the LED may not be lost.

4.2.1 Delayed Following / Queue

As already explained in chapter 3.3.1, we wanted to implement a sort of queue, to better mimic the movement of the robot in front, so complex tasks like driving around a corner would work better.
We did implement such a algorithm in Matlab. The current position of the tracked LED was stored every $50ms$ and was then evaluated by the following algorithm in delay steps of 1 up to 10. Hence a delay of $50ms$ to $500ms$ in 10 steps.
For a low delay of $50ms$ and $100ms$ you could basically not distinguish a difference, compared to the case without this cue. From a delay of $150ms$ upwards, you could see the desired behavior, of course more for bigger delays.
But it was highly impractical, since the robot did loos the robot in front of it very quick. An addition where it was assured, that the LED does not leave the boundaries of sight, simply brought the robot back, to behave as there was no queue at all. Hence we discarded the whole idea with the time delay and approached the problem exclusively with the following tables $yVTable$ (fig. 4.9) and $xVTable$ (fig. 4.8).

4.2.2 Chain Behavior

All robots drive in a quite clean chain. In some situations the little inaccuracies regarding the following path lead to an adding of the small errors so that the last chain number is not absolutely driving the same path as the leading robot. In general, in relation to the driving distance these errors are quite small. It is even possible to make complex movements, like turning on a point and driving the same way back, without distraction the other robots.

4.2.3 Implementation

We implemented the code for the image processing, the trajectory calculation and the driving commands both in MATLAB and in C.
The MATLAB Code is working on a PC, streaming the sensor data, calculating the trajectory and sending commands back to the robot. This code is more for debugging and evaluating new changes on the algorithm. Sometimes problems occurred regarding time delays caused by the wifi-connection.
To avoid those connection delays we wrote the final C code, which is running directly on the robot's microcontroller, so that they are completely autonomous. This code is, as things stand today, not already working stable due to the latest changes regarding the lenses and the resulting missing adaption of the tracking and following parameters. But we are confident to adjust those parameters in the next days and have a working version at the end of this project.
Also the newest eDVS software version 7.1 does not work with our matlab software. Until now we have used version 6.1, which worked fine. Nevertheless finding the reason should also be manageable within the next days.

5 Conclusions

In this project, the implementation of a tracking and following algorithm for the pushbots could be shown successfully.
Due to todays status a minimum of two robots can follow autonomously another remotely controlled robot, by using LED markers and eDVS Sensors. By using more eDVS cameras, our results allow the conclusion that more than two robots could follow autonomously. However, we could not test it yet due to a lack of enough eDVS-boards.
The tuning of the different parameters to adapt the tracking algorithm on our project was successfully and proves the variety of different application scenarios. Further, a stable and reliable following algorithm was developed. On the other hand this algorithm has also some drawbacks, regarding sharp curves and edges to bypass.
Here, ultrasonic-sensors for detecting objects and a turnable standing motherboard (similar to a neck) for tracking LEDs out of the field of vision could fix those problems. Also, the used LED-marker had not the large bandwidth and brightness that would have given space for identifying the right robot to follow in difficult circumstances. The use of better LEDs and more filters on the received image could probably improve this concept. Nevertheless, if those little drawbacks can be handled, the system may get very attractive for logistics and public transportation due to the simple and cheap structure of the system. Also, it is already very attractive as a technology demonstrator for using eDVS cameras in robotics and automation.

List of Figures

Bibliography

[1] Georg R. Müller and Jörg Conradt. A Miniature Low-Power Sensor System for Real Time 2D Visual Tracking of LED Markers. Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference, pages 2429 – 2434.

[2] Jose Alberto Soler Garcia and Dami`a Viana Casals. Chain of small mobile robots controlled by a smartphone. 2014.

[3] Georg Müller, Jörg Conradt. Self-calibrating Marker Tracking in 3D with Event-Based Vision Sensors, International Conf. on Artificial Neural Networks (ICANN), pages 313-321, Lausanne, Switzerland, 2012.