Obstacle avoidance and target acquisition with an event-based camera on a neuromorphic chip

Alexander Dietmüller & Hermann Blum
INILabs
ETH Zürich

January 2017

Abstract

Neuromorphic hardware (and neuromorphic computing in general) show promise to reduce computational effort and energy consumed in robotic platforms for vital tasks like obstacle avoidance and target acquisition, in comparison to conventional computation. But neuromorphic hardware still faces limitations that make its usage challenging, such as variance of components as well as their currently limited scale and precision.

In this project we use the neuromorphic chip ROLLS and the robotic platform PushBot, equipped with eDVS to provide a proof of concept implementation of obstacle avoidance and target acquisition, as well as the combination of both, completely on neuromorphic hardware.

To our knowledge, this is the first time this has been done and our project illustrates feasibility, promises and limitations of this approach.

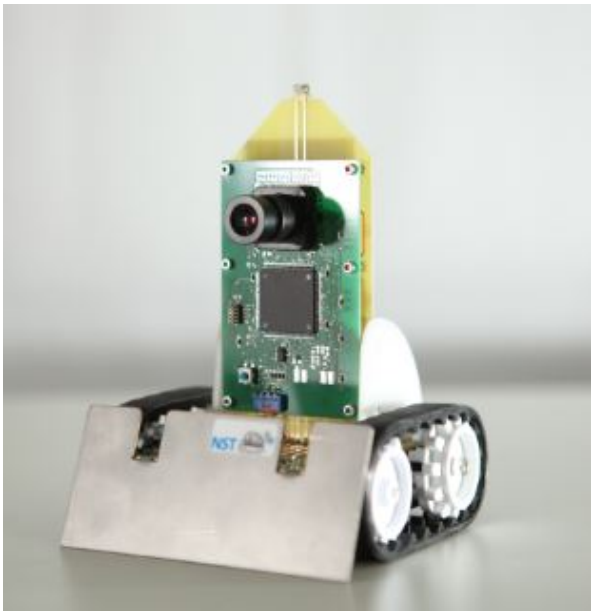

Figure 1: The pushbot platform developed by Conradt et al. [7]

1 Introduction

Collision avoidance is a key task for mobile robotic systems to ensure safety of both the robot itself and any other robot or human in it's environment. Navigation in unknown environment is another unavoidable task for many robotic applications (whether rescue missions, mars exploration or automatic vacuum cleaners) and it often does not only require obstacle avoidance but also target acquisition.
Current robots have limited capabilities and are therefore often separated from humans in safety cages or similar, because both tasks above require a lot of computational effort and energy and are therefore draining the robots power and generally taking away resources from other tasks.

Neuromorphic circuits bring interesting properties such as large-scale parallel processing, collocation of computation and memory and event-based computing from biological neural networks into classical circuit based systems. As these properties make real-time processing of large amounts of sensorial information possible in an energy-efficient way, they are particularly interesting for autonomous robotic systems.

In this project we present a proof of concept for an autonomous robotic system that performs obstacle avoidance and target acquisition in an unknown environment. All computation for this system is done on the mixed-signal 'Real-time On-Line Learning Spiking' (ROLLS) neuromorphic processor [8]. Sensory input is provided by a Dynamic Vision Sensor (DVS) [4] and other sensors, e.g. a gyroscope. Both are provided by the robotic platform "PushBot", developed by Conradt et al. [7], that was used for this project.

2 Methods

Our system runs on an semi-autonomous robotic platform. This consists of a robot with differential drive and mounted eDVS vision sensor called 'PushBot' and the neuromorphic chip 'ROLLS', which is connected to the micro-computing chip 'parallella'. Parallella and PushBot are connected via Wi-Fi and parallella is used to manage the data flows between ROLLS and PushBot. No computation is done on parallella, only transformation of data formats and routing of information.

2.1 Hardware

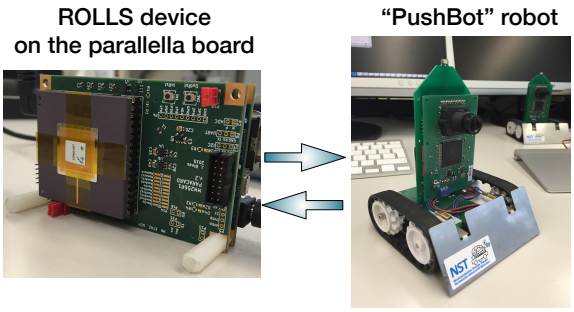

Figure 2: The robotic setup used in this work: the neuromorphic processor ROLLS is interfaced wirelessly to the PushBot through the microcomputer parallella

ROLLS The ROLLS neuromorphic processor comprises 256 Adaptive-Exponential integrate and fire (AdExp IF) silicon neurons, implemented using analog electronic circuits. The neurons express biologically plausible neural dynamics including configurable refractory period, spike frequency adaptation, and time constant of integration. The 256 neurons on the ROLLS chip can be connected to each other and to external signals via three sets of synapses: each neuron has 256 programmable (non-plastic) synapses, 256 learning (plastic) synapses, and 4 auxiliary ("virtual") synapses used to stimulate neurons from the parallella. The programmable and on-chip routing on the ROLLS that supports all-to-all connectivity allows us to implement any arbitrary neural architecture. However, the synapses can assume only one of 4 possible synaptic weight values that can be programmed via a 12-bit temperature compensated bias-generator. An extra digital circuit allows the user to specify if the synapse is excitatory (positive weights) or inhibitory (negative weights).

PushBot and eDVS The PushBot mobile robot is equipped with an embedded DVS silicon retina. Each pixel of the DVS reacts asynchronously to a local change in luminance and sends out an event. Every event contains the coordinates of the sending pixel (x, y), the time of event occurrence (t), and its polarity (pol: "on-event" or "off-event"). Due to the asynchronous sampling, the DVS is characterized by an extreme low latency, which results in μs time resolution.
As the DVS detects the spatio-temporal changes in a visual scene, a static camera only perceives moving objects. However, on a moving robot, the DVS produces a continuous stream of events at the objects' boundaries, where changes are induced by the sensor motion. The fact that the DVS also emits a fairly large amount of input-dependent noise makes the use of this sensor particularly challenging in navigation scenarios.

The PushBot platform also features an IMU sensor with various components that we use to get sensory feedback of the robot's heading direction (compass) and it's turn velocity (gyroscope).

parallella The parallella computing platform is able to stimulate neurons on ROLLS, receive spike events from ROLLS and DVS events from the PushBot as well as sampling the PushBots sensory information of the IMU. It is also able to set synaptic connections on the ROLLS and change the bias settings in real time. We make use of the 'NC-

SRobotLib' Library developed at INI to interface the ROLLS chip and communicate with the robot.

The same hardware setup is used in different other projects at INI, such as a predecessor project by Michel Friesing, and we did not develop or improve any hardware components. However, the refurbishment and improvement of the software library is an integral part of our work.

2.2 Neuronal architecture

2.2.1 Robot Commands

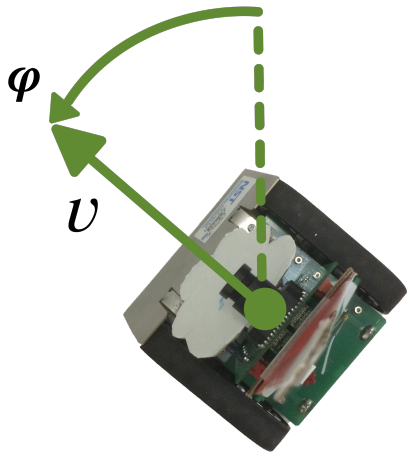

Figure 3: System representation of the PushBot. The state is defined by the heading direction φ and the forward velocity v.

One of our goals is to implement all logic in ROLLS, meaning that parallella should only forward signals. This leads to the first challenge: Controlling the PushBot with neuron populations on ROLLS.

We model the PushBot movement with forward velocity v and angular velocity $\dot{\varphi}$, following the definition from figure 3. We encode both variables with the average firing rate of a population of neurons on the ROLLS. To encode the sign of $\dot{\varphi}$, we use 2 populations of equal size that inhibit each other in a winner-takes-all (WTA) dynamic. This way, the decision of turn direction is made in ROLLS, since only one of the turning populations will ever be active at the same time.
We use three populations of 16 neurons each to represent 'speed', 'angular velocity (left)' and 'angular velocity (right)'.
On parallella, the firing rates are computed as follows: In regular sampling intervals the number of spikes in the respective population is counted. As a first approach we mapped these counts proportionally to velocities:

$$v \propto n_{\text{speed}}$$
$$\dot{\varphi} \propto n_{\text{left}} - n_{\text{right}}$$

However, this approach required relatively long sampling intervals (≈200ms) to provide sufficiently smooth output. To improve our reaction time, we implemented a first order low pass filter with parameter α to update an estimate for spikes per interval and population.

$$n_{\text{estimate}} = \alpha \cdot n_{\text{old\_estimate}} + (1 - \alpha) \cdot n_{\text{count}}$$

The parameter α is related to a time-continuous low pass filter by discretizing with the respective sampling time T. The desired time constant τ of the time-continuous filter determines α accordingly:

$$\alpha = \exp\left(-\frac{T}{\tau}\right)$$

We used a sampling time of 50ms and a time constant of 100ms resulting in $\alpha = 0.6$

The number of spikes per interval and population is normalized over interval duration and population size to get the firing rate per neuron and second. Finally multiplied by a user defined scaling factor this rate is sent to the robot (also every 50ms).[1]

With these translational methods we can implement all driving logic on ROLLS and only use parallella for signal transformation and -passing.

2.2.2 Obstacle Avoidance

The first goal of our neural architecture is robust obstacle avoidance. The robot should be able

[1] We were not able to obtain documentation of the PushBot platform defining the unit of the motor velocities. Therefore, the magnitude of this scaling factor remains as arbitrary as the input the PushBot receives.

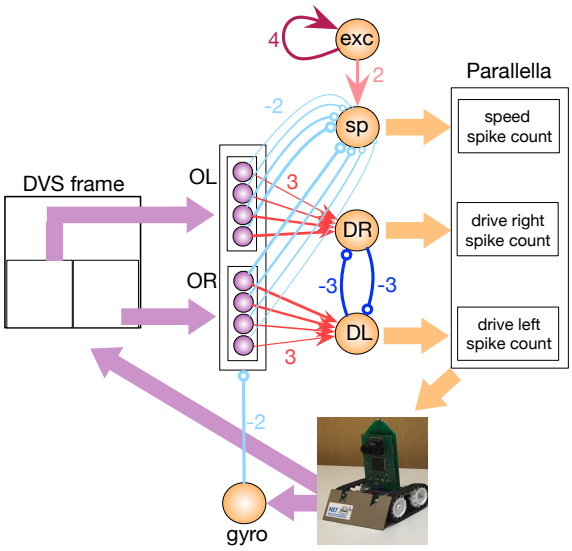

Figure 4: Connectionists' plot of the obstacle avoidance architecture.

to navigate in an unknown environment without crashing into obstacles.

We only consider the lower half of the DVS field of view (FoV) for this task, since objects in the upper half are either above the robot or very far away and therefore will not cause collisions. A population of 32 neurons on ROLLS is used to represent obstacles. Columns of 4 × 64 DVS pixels are mapped to one neuron each (therefore all of the 128 × 64 DVS pixels in the lower half of the FoV have a corresponding neuron in ROLLS). For every event in a column, the respective neuron is stimulated. After sufficient stimulation, the neuron will spike and therefore signal the detection of an obstacle.

In the next step, obstacles need to actually influence the robot's behavior. We realize this by connecting the obstacle population with the command populations described in 2.2.1. The half of the obstacle population representing obstacle on the left has excitatory connections to the turn right population and vice versa. Following a concept introduced by Braitenberg [1], we implement a simply dynamic that enables the robot to turn in response to obstacles: In absence of obstacles, we drive straigt forward. Any detection of an obstacle will slow down the robot. If there are more obstacles left than right, we turn right, otherwise left.

In order to represent the 'base speed' in the absence of obstacles we implemented a constantly excited population of 8 neurons that excites the speed population while all neurons in the obstacle population have inhibitory connections to the speed population, causing the robot to slow down in the presence of obstacles (with stronger deceleration for bigger/more obstacles).

This setup already works well and is used for the first experiments, but is overwhelmed by cluttered environments since the robot will see obstacles everywhere.
Even worse, any turn will increase the number of incoming DVS events drastically making the robot to turn on the point forever. This is because the DVS senses changes on the 'retina'. Therefore, any movement of the camera will cause DVS events. We did observe that the rate of DVS events increases approximately proportional with the angular velocity.

To cancel out this effect, we inhibit the obstacle detecting neurons while turning. This inhibition was initially realized as a connection from the turn populations to the obstacle populations. In the newer versions of the architecture we use the gyroscope of the PushBot. The implementation is described in more detail in 2.2.4.

Up to this point, obstacles in the FoV only differ in the rate of DVS events they produce. An obstacle at the edge of the FoV would have the same influence as one directly in the way of the robot. In order to increase the general speed of the robot and improve the behavior in cluttered environments, obstacles in front of the robot should have a greater influence on speed and turning than obstacles on the edge of the FoV. Therefore, connections to the turn and speed populations from neurons representing obstacles in the center need to be stronger. Since our available synapse weights on ROLLS are limited (as described in 2.1) we achieve this by varying the number of connecting synapses to the respective population. Neurons representing obstacles in the center of the FoV are connected to all neurons in the command populations, neurons representing obstacles on the edge of the FoV are connected to only one neuron in the command populations, with the number of connections decreasing linearly in between.

In conclusion we were able to implement obstacle avoidance using raw DVS input with just 32 additional neurons and carefully linking the differ-

ent parts to be able to distinguish obstacle positions and react accordingly: Strong reactions for obstacles in front of the robot, weaker reactions for others.

2.2.3 Target Acquisition

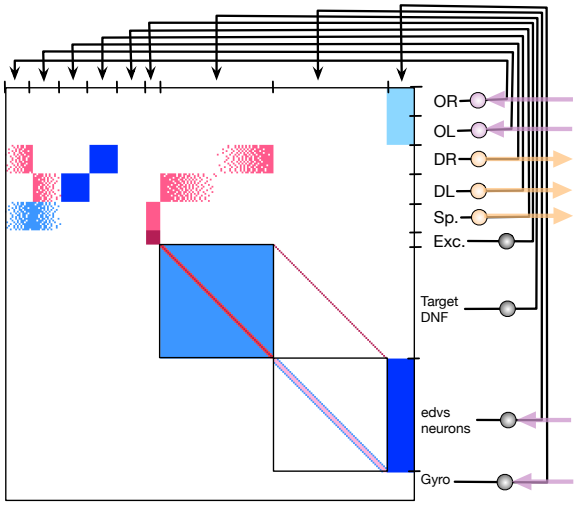

Figure 5: Connectivity matrix of the full neural architecture with obstacle avoidance, target acquisition and gyroscope

We set the requirements for target acquisition with respect to the limited number of neurons on ROLLS in order to ease the recognition of the target. We represent the target by a (second) PushBot with a LED blinking at 200 MHz. This will generate DVS events of high rate, thus sticking out from the high background activity. Our goal is to recognize this target and keep it in memory if the target vanishes for short periods of time.

However, in comparison to obstacle avoidance, this recognition task was still much more complex as the walls of the robotic arena were lower than the LED of the target PushBot, making it necessary to filter out a great amount of background DVS activity as shown in figures 8 or 11.

We realize target acquisition with two populations of 64 neurons each. The first population is used as a filter for DVS input. Similar to obstacle avoidance, every neuron in this population receives input from columns of 2 $\times$ 64 DVS pixels, this time using the upper half of the image. Since the neurons require a steady stream of events to emit spikes themselves, we effectively filter out background noise with this population.
Besides it's filtering property the neural population features a WTA dynamic which enables it to find local maxima.

The second layer represents the target position. Every neuron in the filter population excites exactly one neuron in the memory population. In this population we use global WTA dynamics: Every neuron excites their close neighbors while inhibiting all others. This way we pick the global maximum of the local maxima the filter layer provides and also get sticky behavior in order to keep the target position in memory.

Analog to the implementation of obstacle avoidance, the neurons in the memory population are connected to the turn population. Neurons representing a target on the left excite a turn to the right right and vice versa. Since we need to turn stronger if the target is on the edge of the FoV compared to a target in the center, we use the same connection principle from obstacle avoidance, but inverted: Neurons representing a target in the center of the FoV are connected to one neuron in the turn population, with the number of connections linearly increasing to both sides up to target neurons representing the edge of the field of view with all exciting all neurons in the turn population.

Since we have no effective knowledge about the distance to the target (which would enable us to stop in front of it), we only use target acquisition for turning.

Both target acquisition and obstacle avoidance are connected to the command populations. To ensure that obstacle avoidance is always prioritized over following the target, the connections from target acquisition are weaker than those from obstacle avoidance.

With this architecture we enable the robot to follow the target and avoid an obstacle, if necessary. While we can keep the target in memory, we are not able to adapt the 'remembered position' while we are turning which can lead to undesired behavior (see 3.7). (Ultimately unsuccessful) attempts to improve this are described in 2.2.5.

2.2.4 Proprioception

As described above we receive many more events from DVS while turning, which can lead to turning movements which are longer than necessary, since the additional events are recognized as obstacles and keep the robot turning. Therefore, we need some degree of proprioception to recognize that we are turning and, as described above, inhibit the neurons receiving DVS events as a coutermeasure, similar to how saccadic suppression works in the mammalian eye.

In a first approach we use the activity of the command population as indication of actual turning. Generally this approach works well since when the command population is active, we are actually turning. In some cases, e.g. if the robots movement is limited by obstacles and it can't turn, this information is false and will suppress events from actual obstacles since the robot mistakenly 'believes' it is turning.

In the current version of the architecture we are therefore using the gyroscope data, which provides reliable information about current angular velocity. In contrast to DVS events, this is an integer number sampled every 50ms, so it can't be directly used to stimulate ROLLS. Similar to the speed we couldn't obtain information about the units of this sensor, but experimentally determined the value of the sensor output at (absolute) maximum angular velocity of the PushBot (≈ 8000) and at maximum angular velocity in our application (≈ 2000).

We use this information to transform the sensor output into a number of ROLLS spikes proportionally.

On ROLLS we define two populations of eight neurons each to represent 'turning to the left/counterclockwise' and 'turning to the right/clockwise'. Every sampling step of the sensor they will receive the calculated number of stimulations.

Finally, we use the defined populations to inhibit all populations that receive input from DVS, i.e. the obstacle and filter populations.

This way we successfully implement neurons in ROLLS that sense if the robot is turning and inhibit the populations receiving DVS input to compensate the additional events.

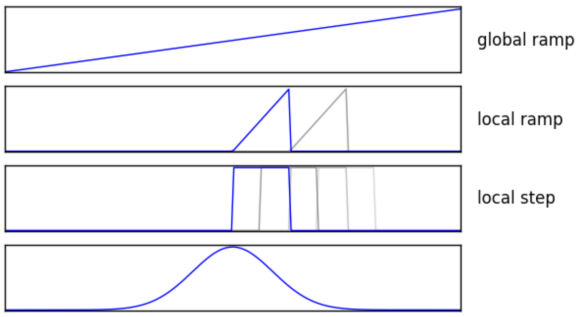

Figure 6: Different shifting inputs. The gradual difference was implemented using different numbers of synaptic connections from the input population. **Bottom**: Activity in target neighborhood.

2.2.5 Extension of Target Memory

As the experiments in 3.7 show, our general target acquisition architecture from section 2.2.3 fails if the target is out of sight. We can tune the target memory in a way that it becomes 'sticky' and keeps the target in memory, but this remains unaware of any turning and therefore might produce wrong behavior: E.g. if the target was on the right when it was lost, but the robot keeps turning to the right while navigating around some obstacles, it will still remember the target on the right even though it might be on the left now.

This wouldn't be a big problem if we were able to detect the target while turning, as the robot would just turn until it finds the target again. However, as described above, turning induces a high rate of DVS events and as our target is marked by a high rate of DVS events, we can't reliably distinguish the target from the background if the angular velocity is too high.

We tried two approaches in order to address this problem: Our first approach was to update the target memory with information from the gyroscope. Our second approach does not memorize the relative target position, but makes use of the compass sensor and memorizes the absolute target position.

Memory Shifting As described in 2.2.3, we remember the target with a 'sticky' WTA Kernel in a population of 64 neurons. We want to shift the position of the actively spiking neighborhood according to the activity in the populations receiving input from the gyroscope, as described in 2.2.4.

Figure 6 shows different inputs we tried to apply to the target memory population in order to shift it. Without loss of generality, we only describe the problem of 'shifting to the right'.

Our first idea was a global ramp input. However, the local difference was too small to overcome the 'sticky' behavior and we either got a non-moving target position, or, for increased influence of the global ramp, we would only shift the target position to the absolute right without any transition (Since the maximum of the ramp becomes stronger than the active neighborhood).

Both the idea of local ramp and local step work in the same principle: There are different inputs shifted by position. This way, there always is one population closely to the right of our current neighborhood providing a sufficiently strong difference to (locally) shift active neurons. All other (interfering) populations are inhibited based on the current position of the active target neighborhood. All shifting populations receive input from the gyro populations, so that the single population that is not inhibited can cause a correct shift. However, both methods could not be realized on ROLLS as we could not use our available weights to create a WTA-kernel that was both 'shiftable' and still had the generally required filter and memory properties as described in 2.2.3.

Global Position Memory The amount of requirements the kernel of the target memory population had to satisfy were to high to effectively implement a 'shifting' method. Therefore, we searched for a completely different approach: Schneegans and Schöner [9] describe a mechanism of coordinate transformation that allows us to combine the general heading direction of the robot, which can be read from the compass of the IMU, to the relative target position, found by processing DVS events, into the absolute angle position of the target with respect to the robot's position.

The global position is updated as long as the target is in the FoV and memorized when it is lost. This memorized position is then transformed back into a relative target position, which is used to excite the turn populations.

Using 108 neurons, we could realize a version of this transformation on ROLLS that distinguishes 6 different heading directions. It was possible to tune this architecture such that the memory was updated as long as the target was in the FoV and the memory was kept if the target was lost. Furthermore, when turning the robot (effectively changing it's heading direction) while covering the DVS lenses, the memory, together with the heading direction, would activate different cells in the transformation matrix, representing different relative positions of the target.

However, our realization was not robust enough to 'survive' the noise received by the DVS while the robot was driving around. In addition, the compass sensor of the PushBot was not reliable enough. A 360 degree turn, if done too quickly, can lead to a compass output difference of up to 180 degrees.

Another critical point is the detection of false targets. The simple target detection we use in our system will occasionally detect a target for other objects that generate high rates of DVS events, e.g. vertical edges (see 3.7). This will immediately update the memory position and we will stop turning back to the original, correct, target.

We were therefore not able to provide a version of the global memory usable in 'real' scenarios, nevertheless this architecture is a little more easy to tune than the shifting memory (although it is still difficult).

2.3 Library

Our starting point is the 'NCSRobotLib' created by INI Labs and at the CapoCaccia[2] workshop, which provides an infrastructure to connect all platforms, i.e. PushBot, eDVS and ROLLS. We made various small improvements to different parts of the code and will list the most relevant additions below.

Furthermore, there exists a javascript framework 'WebAER' to set ROLLS connections and biases.

All our code can be found in our group within the gitlab of INI.[3]. Of particular interest are the repositories `NCSRobotLib` for the library and `pushbot_target_and_obstacle` for the project source code.

[2] `http://capocaccia.iniforum.ch/`

[3] `https://code.ini.uzh.ch/semesterproject_hermann_and_alexander`

2.3.1 Architecture

Setting up the neuronal architecture with javascript while the rest of the code is written in C++ introduces some limitations, e.g. processing of DVS events has to be handled in both parts of the code (in C++ it needs to be specified where the events will be sent, which is determined by the architecture specified in javascript). To simplify this setup we implement utilities to set up the neuronal architecture in C++. Similar to the spiking neural network simulator BRIAN [2] we specify groups of neurons. These groups can then be connected to each other. An important detail is that these groups also contain the respective neurons' indices on ROLLS, so they can later be used to stimulate a specific group of neurons or assign a received spike to a defined group.

Internally the whole architecture is represented by a connectivity matrix and we provide several functions to easily connect neurons or neuron groups in various ways, e.g. every neuron in a group to all neurons in another, every neuron to a selection of neurons in another group, specified by a weighting function, connect with a kernel to close neighbours.

This piece of software makes it easier to define the neural architecture in C++ and connect inputs and outputs to the respective populations.

2.3.2 Plotting

Experimental data needs to be displayed, but to keep the focus on the experiment and not on the processing of data, we automated as many tasks as possible.

First attempts included sending all logging data to a host computer via UDP, but the humongous amount of DVS events (several thousand per millisecond) quickly managed to overwhelm the networking capabilities of parallella, so we went back to saving the log data on parallella. To avoid filling up all available space, we provide an easy script to fetch all current log data from parallella for processing on the host computer.

This processing is automated using 'python', 'jupyter notebook', 'pandas' and 'matplotlib' (common software for these task well known in the scientific community). We provide functions to quickly browse and load existing log data. Furthermore the library has the capability to create plots suited for our architecture, it can:

- accumulate DVS events over a given time period and plot them as one frame
- plot neuron spikes over time and label and color activity according to specified neural populations
- create plots combining above functions easily for several points in time

(All plots in section 3 have been created this way)

Additionally, we recorded our robot from above in most experiments. To turn these videos into a single image we also provide a script using 'ffmpeg' and 'imagemagick' to split the video into frames and overlay them. (Also used for the plots in 3)

These scripts enable fast processing of logs generated by NCSRobotlib for other users as well.

3 Experimental Results

The general robustness of our obstacle avoidance setup as well as it's limitations were tested in a wide range of experiments. We tested it against different types of obstacles and in different surrounding conditions. Concerning the parameters we used, such as ROLLS bias settings and program constants, we did not perform a search to find the global optimal settings as the whole project was a proof of concept to show what is in principle possible with the introduced neuromorphic robotic system. However, the experiments shown here were conducted after a general parameter search for a setting that showed a reasonable balance of robustness and movement speed.

The neural architecture evolved with the analysis of experimental results and it is due to the time limitations of this semester project and the decision together with our supervisors to focus on the development of target memory rather than the characterization of the system that we did not perform all the tests with the final version of the architecture. Therefore, we describe the stage of the architecture with every experiment if it differs from the most recent version.

All experiments were conducted by recording the activity from DVS and ROLLS, if necessary also from the gyroscope, a video of the experiment, and

additionally the movement commands send to the PushBot.

Most of our experiments are set in a controlled 'arena' environment with both white floor and walls. We do not want the walls to interfere with the robot behavior while still allowing the robot to avoid them. We achieve this by attaching a high-contrast tape to the top of the walls. The tape is positioned at a height adjusted for the position of the camera on the PushBot. Only if the robot is close (≈ 5cm) the tape will be seen in the lower half of the DVS image (which is used for obstacle avoidance) – but as soon as it is seen it provides a lot of input due to it's high contrast. This way the robot is not influenced by the walls inside the arena, but is stimulated enough to be able to avoid them.

For many experiments we will show 1 or 2 exemplary results. This is due to the fact that these experiments should not only show *to what extent* our concept is working, but also *how*. All experiments were run for at least 3 times and we mention if there were any inconsistencies between the runs such as failing obstacle avoidance. However, the actual trajectories and neural activities will differ between experiments too much to show a meaningful and useful synthesis of different experimental runs. Of course, all the data and videos of the experiments are collected in the archive server of INI.

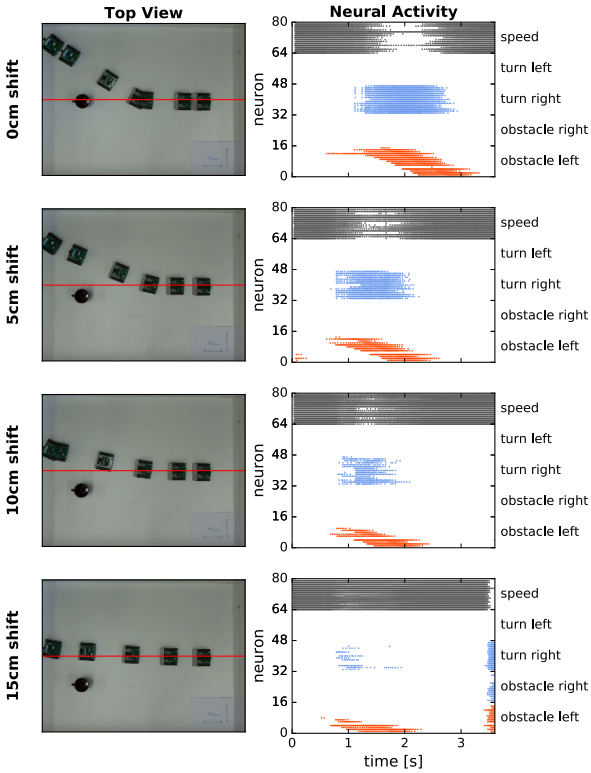

Figure 7: Response of the robot on different obstacle positions. The obstacle is shifted vertically to the robot's initial heading direction. **Left**: Overlays of the overhead camera images with static time intervals to indicate the speed. The red line marks the initial heading direction of the robot. **Right**: Activity on the ROLLS for the labeled neural populations.

3.1 Different Obstacle Positions

This experimental series was conducted with the newest version of the architecture as described in 2.2.2.

Obstacles on in the center of the FoV should have a stronger impact than those on the edges of the FoV.

Figure 7 shows the robot's response to different obstacle positions. The robot always starts with the same initial position and heading. The obstacle in this case is an ordinary cup from INI's kitchen. Initially, it is placed directly in front of the robot, while shifting it vertically to the initial heading direction of the robot by 5cm per experiment.

The experiment qualitatively shows the expected difference in the magnitude of the robot's response. For an obstacle that is less in it's way and therefore closer to the edge of the DVS FoV, both the turn command and the slowdown are weaker. This behavior evolves gradually and monotonically with the shift of the obstacle.

We also observe that not only the horizontal position of the obstacle, but also the distance between robot and obstacle or – more precisely – the relative size of the obstacle on the DVS image (decreasing with distance and increasing with size) are of great importance. The beginning of the experiment with 0cm shift shows that a single neuron in the obstacle population is not enough to excite the turn population. Only as the robot gets closer to the obstacle and therefore the obstacle occupies more columns of the DVS lower half image and excites more obstacle neurons the activity is strong enough to start a turn.

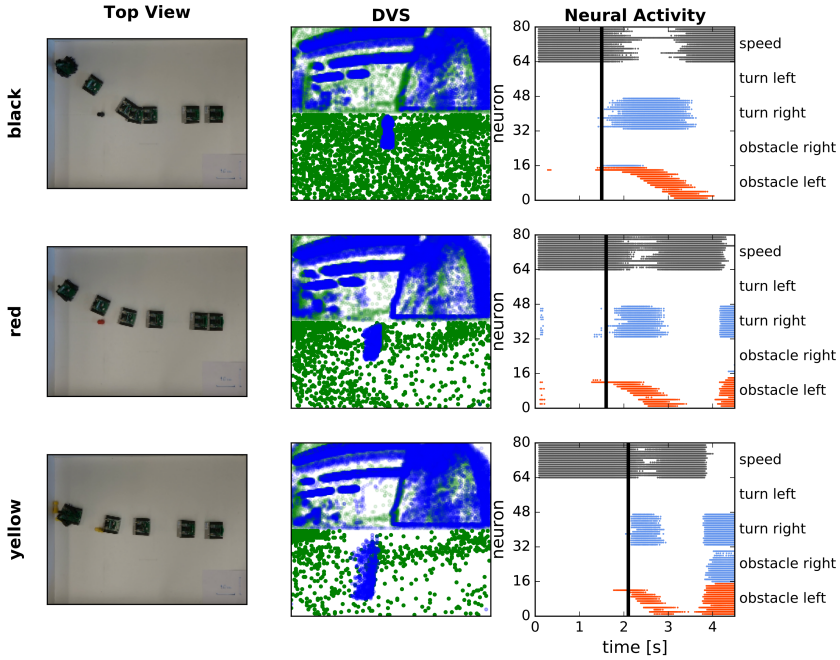

Figure 8: Response of the robot to different obstacle colors. **Left**: Overlays of the overhead camera images with static time intervals to indicate the speed. **Center**: Image of DVS events accumulated over 1.5s up to the start of the turn, indicated by the black vertical line in the neural activity plot. **Right**: Activity on the ROLLS for the labeled neural populations.

This influence of size and distance is a side-effect of the obstacle population setup where each obstacle neuron receives input from a DVS image column, which was implemented to encode horizontal position.

We can conclude that our architecture indeed leads to the intended weaker response to obstacles that are not directly in front of the robot. However, our setup will also avoid bigger and especially wider obstacles with a stronger response than small, narrow obstacles.

3.2 Different Colors

This experimental series was conducted with the newest version of the architecture as described in 2.2.2.

Even though DVS events do not contain any information about color but rather relative contrast, we do find that the color of obstacle relative to the background color of the environment is an important factor that influences the robot's behavior. We show below that this effect is also linked to lighting conditions and the speed of the robot itself.

In this experiment we placed the robot in the same initial position inside the arena for all experiment runs. In front of the robot we placed medium sized obstacles (approx. 5cm height and 3cm diameter) of equal size and shape but different color. Figure 8 shows the behavior of the robot with default bias settings moving towards a black, red and yellow obstacle.

We observe two effects: The robot's behavior depends on the color of the obstacle, and it does not successfully avoid obstacles of every color with the used default bias settings. However, the DVS column of figure 8 clearly shows the obstacle regardless of its color. The ROLLS activity plot also shows spikes in the obstacle populations for all colors. It is the distance to the obstacle at the time of the first spikes that increases from the black to the red and to the yellow obstacle.

As described in 2.2.2: Every event of the DVS is used as an input to one of the obstacle neurons.

However, a certain rate of events is necessary to make the neuron spike. We make use of this effect in general to filter out noise.
While the number of events generated by an obstacle generally increases with the robot approaching the obstacle (because the obstacle will appear bigger on the DVS image), our observation shows that the contrast between an obstacle and the background influences the number of DVS events produced by this obstacle greatly. In this example a yellow or red obstacle in front of a white wall produces less DVS events than a black obstacle and therefore has a worse signal-to-noise ratio. In our default bias setting the filter threshold is too high to avoid yellow obstacles. They provide a sufficient number of DVS events to activate the turn population only when the robot is already too close and their input is too weak to cause a turn strong enough to avoid the obstacle at this point.

In general we could find that the PushBot – with our default bias setting – reliably avoids obstacles of black, red and blue color, while regularly crashes into yellow obstacles. The experimental results show that regardless of the bias setting, our principle of detecting an obstacle by rate of DVS events and only using the pure neural filtering capabilities does require to set an arbitrary threshold that balances the robustness towards noise and low-contrast-obstacle avoidance.
This threshold can be changed by changing the ROLLS bias setting for the stimulating synapses, changing the number of synapses used for one stimulation or changing the number of stimulations per DVS event.
Additionally, we could already show in a previous version of the architecture that reliable avoidance of yellow obstacles is also possible by changing the connecting weights with which the obstacle population excites the turn and inhibits the speed, although this leads to a generally very 'cautious' robot navigating rather slowly (because it decelerates both often and strongly) and turning very strongly for obstacles with high contrast.

3.3 Different Lighting Conditions

This experiment was conducted with an earlier version of the architecture. No gyroscope was used, instead the turn populations themselves had an inhibitory connection to the obstacle population, as mentioned in 2.2.2. The described encoding of the horizontal obstacle position was not used, instead the whole left and right part of the obstacle populations were stimulated for an obstacle in the lower left and right half of the DVS FoV.
The robots default speed (where no obstacles are present) is slightly slower than in other experiments using the newest setup.[4]

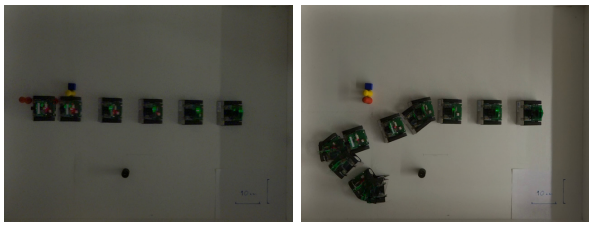

Figure 9: Overlay image of robot trajectory in the arena at different lighting conditions. **Left**: Dark **Right**: Lighter, but still less light than with regular conditions.

The robot is placed in the same initial position for all experiment runs. The experiment was done at night, so there was no sunlight, and we used different office lights to simulate varying lighting conditions.
In the direct heading direction of the robot an obstacle each of blue, yellow and red color was placed. A second, black obstacle was placed off the robot's initial heading direction but in a way that the robot would have to avoid it after turning away from the first obstacle. This was done to test the response to obstacles both after driving straight and while turning.

Figure 9 shows the robot's trajectory for 2 different lighting conditions, both beeing darker than our usual experimental setup. They are relevant examples for the general robot behavior at these lighting conditions as we tested each condition for at least 3 times.

Our experiments show that the obstacle won't get recognized below a certain general brightness level. This result is in our opinion linked to the problem of different colors described in 3.2 as the contrast of obstacles in front of a background is ob-

[4]The default speed is dependent on both a proportional factor as described in 2.2.1 and the ROLLS weight bias settings. The exact speed difference could technically be obtained by analyzing the video files, but is not relevant for this experiment.

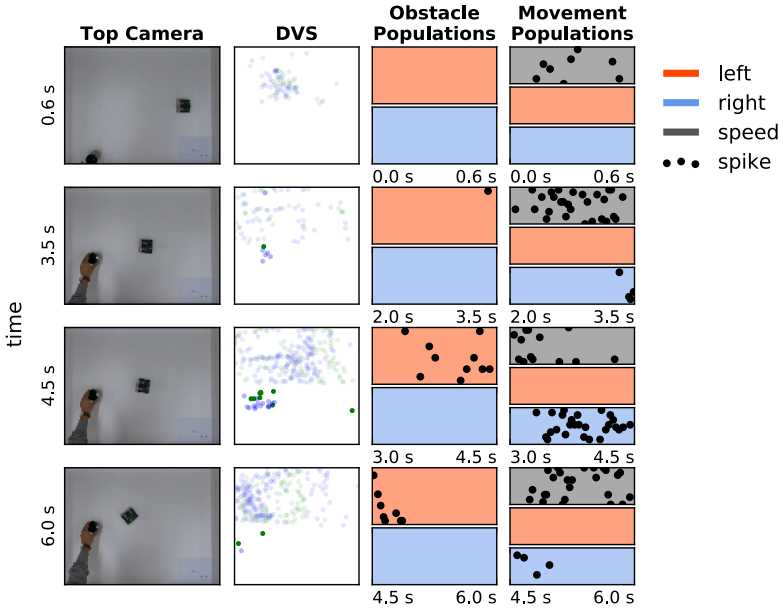

Figure 10: A cup is moved into the robot's FoV while driving forward.
First Column: top view of the scene at the given point in time
Second Column: DVS image integrated over 1.5s up to the given point in time
Third Column: neural activity in the obstacle population
Forth Column: neural activity in the movement populations

viously dependent on lighting conditions. Furthermore, the bottom trajectory shows that obstacle avoidance still works reliably for darker lighting conditions than our regular ones.

As 'night vision' was not part of our project, we did not further follow the problem of obstacle avoidance in dark environments and consider the robustness of the obstacle avoidance in darker and lighter environments sufficient for our tasks.

3.4 Moving Obstacles

This experiment was conducted with an earlier version as decribed in 3.3.

Moving obstacles are of special interest for implementations of obstacle avoidance as they are very common in real world navigation problems and require the ability to react on changing environments.

The robot is placed in the same initial position for all experiment runs. Initially, there is no obstacle present in it's FoV. After the robot starts moving forward, an obstacle is moved in it's way. This procedure is repeated with different distances between robot and the obstacle and different speeds of the obstacle.

The robot is successfully avoiding the moving obstacle without difficulties.

3.5 Cluttered Environment

This experiment was also conducted with the newest version of the architecture. We show that our architecture enables the robot to navigate in a cluttered environment.

The robot is placed in the 'arena', which is populated with black cylinders, roughly 5cm high and 3cm in diameter, as obstacles. The cylinders are placed randomly.

We find that the robot is able to avoid most obstacles 'on-the-go', i.e. without the need to stop completely, and is also able to drive through relatively narrow gaps (1.5 times as wide as the robot), as seen in *Position 2* in figure 11.

On the other hand it also shows the weak depth-perception of our robot. Objects further away occupy less columns in the DVS image and therefore provide less input to the obstacle populations, but this is not always sufficient to suppress overly cautious avoidance maneuvers: In *Position 3* in figure 11 the robot recognizes the cylinders on the right as obstacles and turns away from them, although they are far away and no action would be necessary. Still, since the current architecture can't differentiate between a close, narrow and a far-away, wide obstacle, it makes the correct decision to ensure it will not collide with the potentially close obstacle.

In different experiment runs we encountered

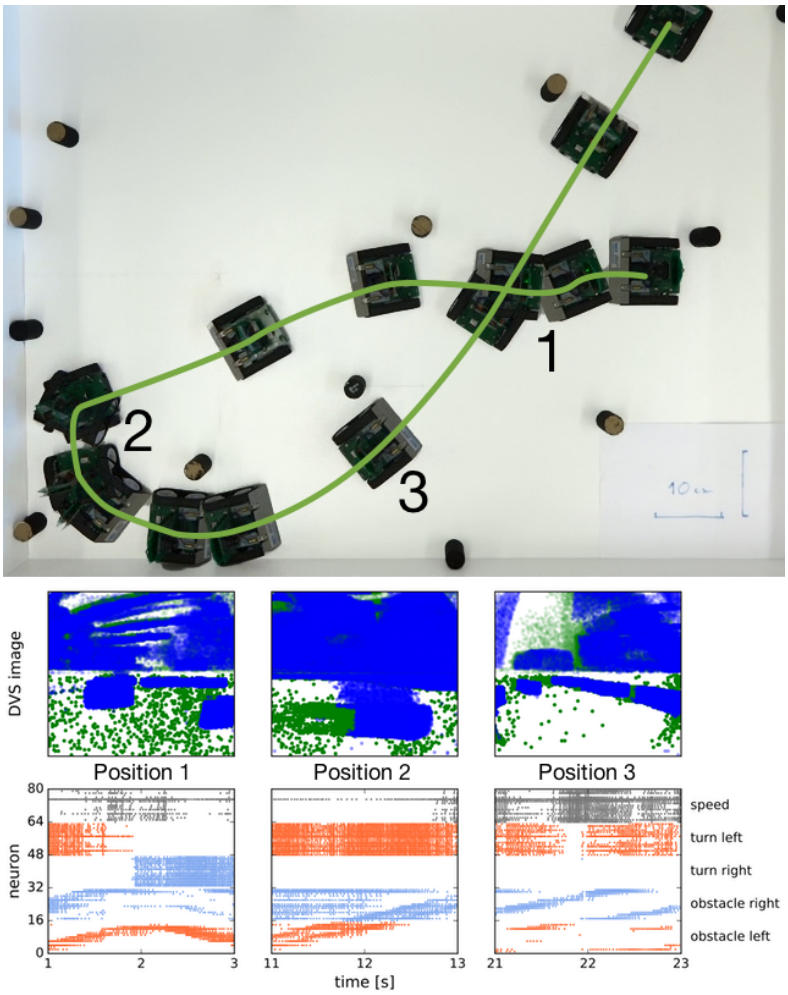

Figure 11: Robot navigationg a cluttered environment.
Top: Overlay-ed overhead camera images with static time intervals to indicate the speed. The marked points correspond to the columns below.
Center: Image of DVS events accumulated over 1.5s up to the start of the turn, indicated by the black vertical line in the activity plot.
Bottom: Activity on the ROLLS for the labeled neural populations.

problems in the combination of our taped walls and regular obstacles. Walls were not always recognized and therefore the robot's movement was different to the intended movement as a wall was in it's way. As we do not implement motor encoder feedback or GPS position feedback, the robot could not adapt to this situation. Occassionally we did observe failing obstacle avoidance connected to this problem. As the walls themselves were 'unnatural' and the tape an engineered way to make them visible, we did not want to develop a solution for this self-generated and irregular problem.

All in all navigation in a cluttered environment works smoothly and reliably, although the robot sometimes unnecessarily avoids obstacles that are quite far away.

3.6 Cluttered Environment without Gyroscope

In 2.2.4 we describe that turning increases the number of DVS events making obstacles seem larger as they are and we can give our robot some proprioception (by using its gyroscope) to compensate this effect. In this experiment we examine robot behavior without the gyroscope data.

The robot is placed in the same environment as in 3.5, the only difference in setup being the disabled gyroscope.

Comparing *Position 1* in both figure 11 and 12, the much greater activity in the obstacle population without gyroscope directly shows the missing inhibition. This leads to keeping the robot turning while it actually could pass between two objects,

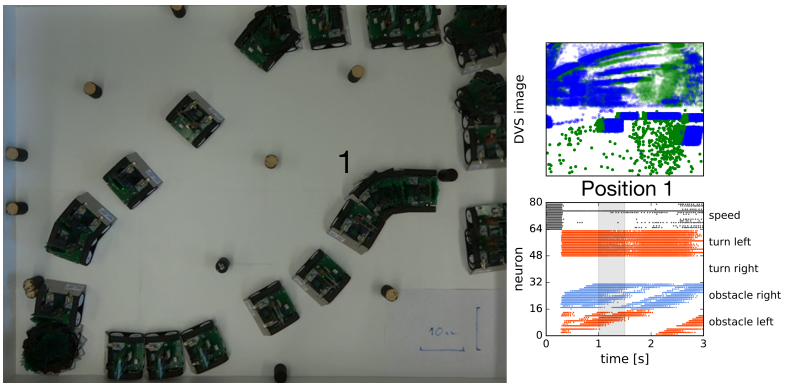

Figure 12: Robot navigation a cluttered environment without proprioception (i.e. no gyroscope).
Left: Overlay-ed overhead camera images with static time intervals to indicate the speed. The marked point corresponds to the plots on the right.
Right, Top: Image of DVS events accumulated over 0.5s at the indicated robot position.
Right, Bottom: Activity on the ROLLS for the labeled neural populations. The gray area marks the time interval used for the DVS plot.

without the gyroscope the avoidance maneuver is much longer and the gap between the two cylinders in front of the robot (although big enough) is not used, since the obstacles are too strong. In addition the robot is driving more slowly (indicated by the overlays, which are takes in the same time intervalls. In figure 11 the distance between two still images of the robot is greater than in figure 12, indicating higher speed.

Nevertheless, the robot is still able to navigate the cluttered environment without crashes, but we conclude that by using the gyroscope the robot is able to drive faster and go through narrower gaps while turning more smoothly.

3.7 Target Acquisition

This experiment was conducted with the most recent version of the obstacle avoidance architecture as described in 2.2.2, but with some differences in the target acquisition. Different to the architecture described in 2.2.3, the first layer does not have a WTA-kernel, only the natural noise-filtering capability of the neurons is used. The filtered spike train is fed into the target layer, where we filter out the global maximum using a mexican-hat kernel that features a global inhibition and an even stronger local inhibition, while it is also tuned to have self-excitatory behavior so the last target position stays in memory even if DVS input decreases.

The experiment was conducted with a static target PushBot that has it's LED blinking at 4 kHz with 75% on-time. The active PushBot is placed in the same static position for all experiment runs finding the target left of it's initial heading direction. On the line between the two robots we placed a small black obstacle, the same one we used in 3.5.

The driving speed of the robot and the neural settings differ a lot from the experiments only showing obstacle avoidance. This is mostly due to the huge amount of background activity in the upper half DVS image generated by the robot's movement. If the robot is too fast, this activity will be higher than the activity of the blinking LED and target acquisition is not possible anymore with our simple target recognition method.

Figure 13 shows 1 exemplary run of the experiment. The trajectory shows that the robot successfully approaches the target while avoiding the obstacle.
The ROLLS activity shows that the single target is successfully found and followed by the WTA 'target' population, but due to the 3 objectives (find global maximum, update in real time, stay in memory if target not present), the representation of the target position in the 'target' layer does not move as smooth as the input 'edvs layer' population. This is part of the reason we later changed the setup of the two layers to the architecture described in 2.2.3. The path of the robot is 's-shaped' and this alternating behavior can also be found in the activity plots of the 'turn left' and 'turn right' populations on the ROLLS. This is the result of an attractor-repeller dynamic between the obstacle avoidance and target acquisition. As the connection from target or obstacle representing layers to the turn

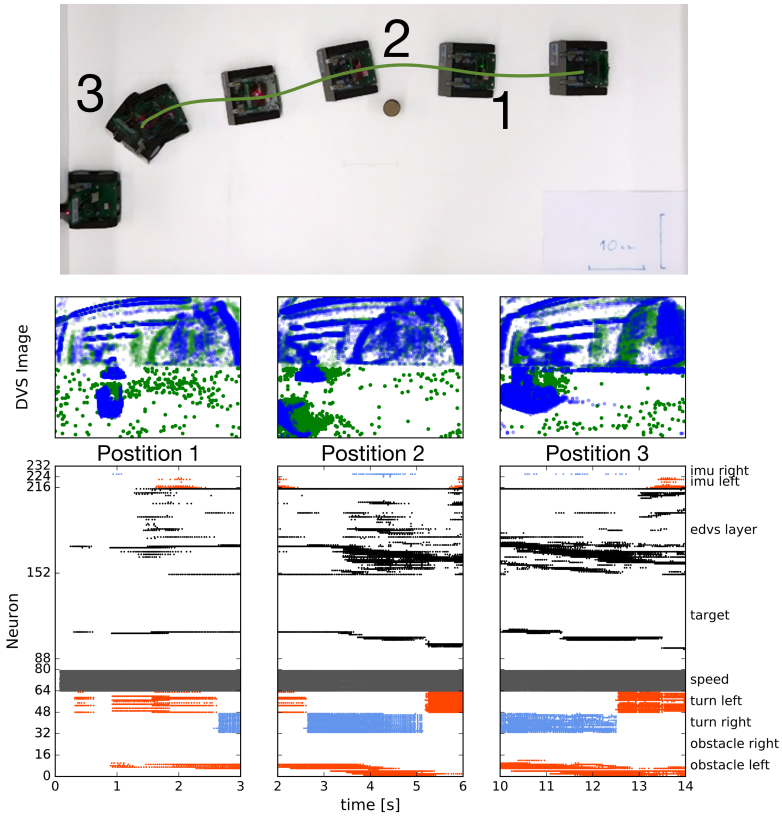

Figure 13: Robot approaching a target robot while avoiding an obstacle on the way.
Top: Overlay-ed overhead camera images with static time intervals to indicate the speed. The marked points correspond to the columns below.
Center: Image of DVS events accumulated over 1.5s at the indicated robot position.
Bottom: Activity on the ROLLS for the labeled neural populations.

populations depend on the relative position of the target or obstacle to the robot, the 'strength' of the target attractor as of the obstacle repeller increase of decrease within every turn. Consider for example the activity up to position 2. From second 3 to 5 the robot will turn right to avoid the obstacle. While turning, the obstacle moves to the left of the FoV, making the repeller effect weaker and weaker. At the same time, the target moves towards the left edge of the FoV, but the target attractor becomes stronger for a target at the edge of the FoV. Therefore, from second 5 to 6, the robot will change it's turn from right to left even though the obstacle is still in the FoV, as the DVS image at position 2 shows.

The main limitation we could found in our experiments is the position of the obstacle. If this position is only slightly changed, as shown in figure 14, the robot will simply loose the target because he has to turn away from the obstacle. Here we also find the limitations of our current memory setup. Even thought the target representation on ROLLS has a sticky behavior, the robot will simply find a new target as the target is only defined as the maximum global DVS input column. We will discuss this topic further in 4.

In addition to the presented experiments we did successfully test target acquisition in the office environment. However, this was only successful with less clutttered background scenes.
Furthermore, we did conduct tests where the target was not stable but moved around, remote controlled by the experimenters. We did in general find that moving targets were followed as long as they did not move much faster than the following robot and that they did not move outside of the FoV.

We could show here a working combination of target acquisition and obstacle avoidance where the ultimate decision of which to follow is taken by the mutual WTA dynamic between the 'turn-left' and 'turn-right' populations, which get input from an attractor-repeller-system.
Limitations of this scheme are the general robot speed, which has to be slow enough to detect the

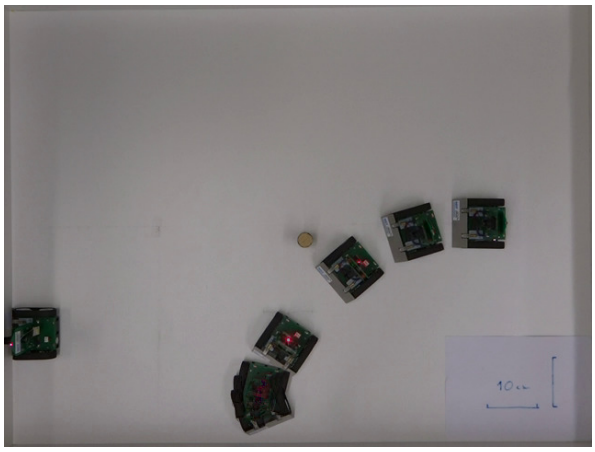

Figure 14: The robot, trying to approach the target, avoids an obstacle and loses the target out of sight.

target, and the awareness of a shift in the relative target position when turning away from an obstacle.

4 Discussion

In this semester project we have demonstrated that the neuromorphic hardware can be used to implement both obstacle avoidance and target acquisition with only 256 neurons. The robot is able to navigate cluttered environments, avoid moving obstacles and follow a target at the same time – and all decisions are made on the neuromorphic hardware.

However, it needs to be noted that the combination of obstacle avoidance and target acquisition is difficult. Combining these two, the limited number of weights becomes a great problem since it is nearly impossible to tune both parts to the optimum. It was found unavoidable to use the same weights in different parts of the architecture, leading to complex interferences in the tuning process.

There are more limitations to the system: In the latest stage of our architecture, we make use of all available neurons, making it impossible to extend our work with the current hardware.

While our experiments show that the processing of raw DVS events is an efficient and robust method for obstacle avoidance, it is a very basic method and could be extended. The experiments for target acquisition also clearly showed limits of this low-level processing.

For obstacle avoidance, we did show that different colors of the same shaped object produce differently strong reactions, sometimes even reactions that are not strong enough, as seen in 3.2. Other problems are caused by very big, low contrast objects, of which the robot could detect the borders but mistakenly tries to pass in between the borders, through the object.[5] This is especially a problem with walls.

While we are filtering in the target layer, we still only take the most salient input as our target, which can lead to false identification if the real target is out of sight and the next strongest input is mistaken as the target.
This was also found to be one of the main problems in our approaches to improve the target acquisition (see 2.2.5). As a results, none of the ideas could be fully realized on ROLLS.

Possible solutions and extensions to the visual processing or memory would require more neurons (and/or weights). With more neurons and weights available, arbitrary many processing steps could easily be added to the architecture to refine its behavior, e.g. not connecting the obstacle population directly to the command populations, but to a series of processing layers, which are then finally connected to the command populations.

This visual processing could also use patterns like optical flow to extract more data (i.e. distance, motion, direction of objects) from the DVS events to further improve the avoidance decisions, e.g. if an object is moving out of the way, less avoidance is necessary, and most importantly making it possible to filter our far-distance background to detect the target.

Working with a neuromorphic processor has several limitations compared to a simulation. Not only are far less neurons available, we had to work everywhere with populations of neurons as single neurons vary too much from each other to encode or process information reliably. Since we introduced a way to compensate the limited number of weights in the ROLLS by varying the number of synaptic connections between populations of multiple neurons, we

[5] Low texture objects are a well known problem in computer vision, leading for example to the crash of a TESLA car into a white truck that was not detected in front of a cloudy sky.

consider the number of neurons a harder limitation than the number of weights.

However, the number of neurons can not only be increased by building bigger chips, but by connecting multiple chips together. With a look at the architecture described here it is actually possible to pipeline information between different stages. The neural populations for obstacle position and target position do not influence each other at all, they only have inputs from the IMU and the DVS and output to the command populations. Therefore, future work could have a look at stacking multiple ROLLS chips together in order to make space for extensions of the architecture. The cxQuad architecture by INI [3] might be interesting as well, since it features a much greater number of neurons and allows exactly this kind of scalability.

The PushBot platform has shown to be easily usable and well suited for our task, but lacks the possibility to be directly connected to the neuromorphic processor. We have bridged this gap in software on parallella (as described in 2.2.1), but for future implementations it might be interesting to have a hardware implementation that could be driven by spikes, ideally with continuous decay and not sampled every n ms to provide an even more direct influence of the command populations on the robot.
Also, 'real-world' tests would benefit from a platform that is able to carry the whole setup, making the Wi-Fi network handling obsolete.

Finally, we did not make any use of the learning capabilities of ROLLS in this project. All weights of the network are set by hand in an often time consuming and error prone process. We are able to show that the architecture works, but if the network could learn, the process of tuning the network might be considerably simplified and the performance improved.

All in all our proof of concept is especially interesting since we present a simple yet flexible architecture that can easily be extended with additional functionality. Together with our improvements to the library (2.3) we hope to enable future work in this direction.

5 Publications

A first stage of this work was submitted to 'Frontiers in Neuroscience' [5] and is currently beeing reviewed.
Another paper on basis of this project was submitted to the IEEE International Symposium on Circuits & Systems (ISCAS) [6] and is also currently under review.

6 Acknowledgments

Our gratitude and appreciation go to Yulia Sandamirskaya and Moritz Milde who have supported our project with tremendous amounts of patience, creativity, overtime and candy bars.

Special thanks go to Tobi Delbruck for making this work possible at all.

We thank Giacomo Indiveri for many useful ideas and literature suggestions.

For the *NCSRobotLib* we would like to thank Julien Martell and Aleksandar Kodzhabashev and all the other people involved.

We thank Jörg Conradt and his team for the creation of the PushBot platform and Michel Frising for first experiments combining the PushBot and ROLLS as these works were the basis for our project.

References

[1] Valentino Braitenberg. *Vehicles: Experiments in synthetic psychology*. MIT press, 1986.

[2] Dan FM Goodman and Romain Brette. The brian simulator. *Frontiers in neuroscience*, 3:26, 2009.

[3] Giacomo Indiveri, Federico Corradi, and Ning Qiao. Neuromorphic architectures for spiking deep neural networks. In *2015 IEEE International Electron Devices Meeting (IEDM)*, pages 4–2. IEEE, 2015.

[4] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128× 128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE journal of solid-state circuits*, 43(2):566–576, 2008.

[5] Moritz B. Milde, Hermann Blum, Alexander Dietmüller, Dora Sumislawska, Jörg Conradt, Giacomo Indiveri, and Yulia Sandamirskaya. Obstacle avoidance and target acquisition for robot navigation using a mixed signal analog/digital neuromorphic processing system. *Frontiers in neuroscience*, 2017.

[6] Moritz B. Milde, Alexander Dietmüller, Hermann Blum, Giacomo Indiveri, and Yulia Sandamirskaya. Obstacle avoidance and target acquisition in mobile robots equipped with neuromorphic sensory-processing systems. In *IEEE International Symposium on Circuits & Systems*, 2017.

[7] Pushbot. Pushbot robotic platform. `http://inilabs.com/products/pushbot/`, 2017. Accessed: 2017-01-09.

[8] Ning Qiao, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumislawska, and Giacomo Indiveri. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Frontiers in neuroscience*, 9:141, 2015.

[9] Sebastian Schneegans and Gregor Schöner. A neural mechanism for coordinate transformation predicts pre-saccadic remapping. *Biological cybernetics*, 106(2):89–109, 2012.