Bachelor thesis

Neurally inspired model for Head Direction Cells: Self-Sustain, Learning and Loop Closure

BITAR Ammar

September 2018

University of Fribourg
Department of Informatics

Institute of Neuroinformatics
University of Zurich and ETH Zurich

Supervisor: Prof. Dr. Ulrich Ultes-Nitsche, Université de Fribourg
Supervisor: Dr. Yulia Sandamirskaya, Institute of Neuroinformatics

Acknowledgements

I highly appreciate the support and want to say thank you:
First of all, I would like to thank my supervisor, Dr. Yulia Sandamirskaya, for the patient guidance, encouragement and advice she has provided throughout my time as her student. I felt really lucky to have a supervisor which was so present and always open for meetings and discussions. I would also like to thank my supervisor at the University of Fribourg, Prof. Dr. Ulrich Ultes-Nitsche, who gave me the opportunity and permission to work on this project at INI. A very special thanks goes to Alpha Brenner for his unending help, patience and tips.

To Alpha Brenner for his unending help and patience regarding the technical side of the project.

To Pannin Pieroj who got me acquainted with V-Rep and provided me with helpful tips and tutorials

Contents

List of Figures

List of Tables

Abbreviations

1 Introduction

Throughout human history, biological systems have been used as a source of inspiration for technological advances. The human brain being the most complex and computationally efficient system, its functioning has been extensively studied. Recent findings regarding the development and adaptation of the Head Direction network in rat pups [Bjerknes et al., 2015] could prove to be of interest to improve one of the main computational problems in autonomous robots face, *Simultaneous Localization and Mapping* (SLAM) by helping to solve loop-closure.

This project explores the modelling of a self-sustaining spiking neural network with multiple shifting levels inspired by the biological Head Direction network. The models were augmented with unsupervised learning through an STDP-like synaptic model in order to adapt connections to erroneous shifting layers upon detection of a visual signal as to approach loop-closure.

Loop-closure aims at correcting error accumulated over time in a robots cognitive map through environmental or uncontrolled factors, such as drift, friction, wind and so on. This work tries to tackle loop-closure in a Head Direction network. Short-term by resetting the robots position correctly on the Head Direction network through a visual cue. Long-term by learning to map the motor-commands (angular velocity) to the correct shifting speed of the activity-hill in the neural network. In essence, the learning would permit robots to continuously adapt their believed position (or rotation in this case) even with change in environment or through drifting, as to try to learn the best matching speed for each situation.

Since their discovery near the rat hippocampus in the early ninties [Taube et al., 1990], Head Direction (HD) cells have been extensively studied in a wide range of fields. Indeed, HD cells have interesting properties, they are connected in a circular network and show a preferred firing direction corresponding to the animals believed facing direction with respect to its environment, somewhat like an internal compass[Taube, 2007]. They seem to be driven by proprioceptive feedback, such as integration of the heads rotational velocity. They're also strongly affected by visual signals which take control over the network when available[Taube, 2007; Song and Wang, 2005]. Since their initial discovery, they have been found in multiple species such as in drosophilia, non-human primates and bats[Kim et al., 2017; Robertson et al., 1999; Finkelstein et al., 2014].

Several theoretical models have been proposed, most of them have in common that the network comprises mutual excitatory and/or inhibitory feedback between its neurons[i.a. Song and Wang, 2005; Zhang, 1996].
Recently it has been shown that rat pups already possess such a network as early as three days before opening their eyes, although rather imprecise. In the first 24 hours after opening their eyes the network adapts using the new visual signals to improve its precision and robustness[Bjerknes et al., 2015]. Such discoveries proved to be interesting for a computational problem in *SLAM*, the problem of generating and updating a cognitive map while keeping track of the agent within, by being useful for loop closure as already mentioned. Recent studies have shown the possibility of implementing such networks on neuromorphic hardware using visual reset to relocalize the robot[Kreiser et al., 2018]. This project also aims at implementing a similar model as Kreiser et al. [2018] and extending it by a learning mechanism inspired from the recent studies seen in rat pups[Bjerknes et al., 2015].
The system developed in this work is composed of three main parts; the HD network, a first transformation array - also called relational neural network - which serves to compute the difference between the estimated current direction and the real facing direction and a second transformation array which maps the current active drive neuron to a new relay neuron based on the computed error, and drives through STDP on this new relay neuron. The next paragraphs will go in detail about their origins, functionality and dynamics; their implementation will be explained in chaptersubsection 2.4.
HD network activity has been modeled using attractor networks since the 90s [Skaggs et al., 1995; Zhang, 1996; Song and Wang, 2005]. Taube [2007] describes it as follow:

> “Continuous attractor networks have been the primary approach for computationally modeling HD cell activity. These networks contain interconnected neurons, which involve recurrent excitation onto HD cells of similar preferred directions and inhibition onto cells with different preferred firing directions. Once initiated, the network can sustain activity without outside excitation. The local area of activity (referred to as the activity hill) is moved around the ring to different directional headings following inputs from idiothetic or allothetic sources.“

From Taubes description, the HD network shows three main properties: First, once ini-

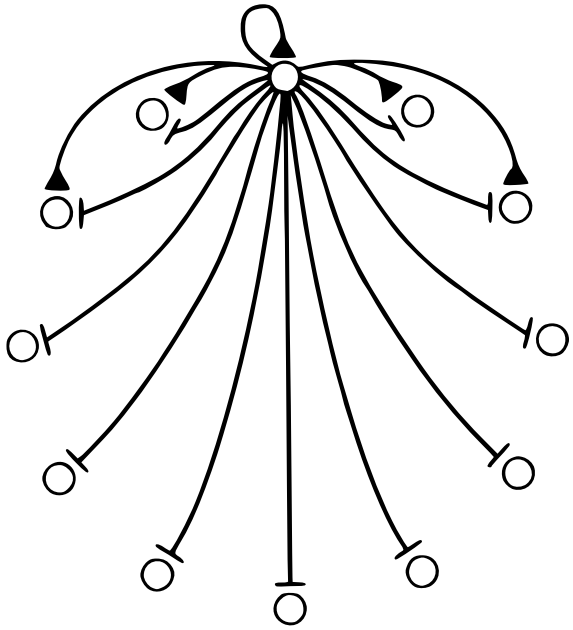

Figure 1.1: Depiction of Head Direction cells connectivity. The active neuron (top-most) locally, laterally excites its neighbours (lines with arrows) while simultaneously exerting a global inhibition on every other neuron (lines ending with a line). Zugaro et al. [2003]

tiated it shows self-sustaining activity, which means even without external input it keeps the activity up. From [Shinder and Taube, 2014] explains how the firing rate decreases when HD neurons are in self-sustain and increases during movement. Secondly, the network shows a preferred firing direction correlated to animals facing direction. This preferred firing direction shows a local *hill of activity*, which follows the heading direction. And thirdly, the HD network takes internal and external inputs to direct the hills movement. These sources being visual cues, such as known landmarks to help orient oneself, or, as mentioned before, proprioceptive feedback, such as the heads angular velocity.

To achieve these properties, a soft Winner-Take-All or ring attroctor network [Zhang, 1996; Taube, 2007; Chen, 2017] model was implemented, in which active neurons latteraly excite their neighbours while having a global inhibitory effect on the network.

A reset group of neurons driven by a visual cue act as external source of hill movement, while a group of *Drive* neurons act as angular velocity (AV) cells[Sharp et al., 2001] and drive internally generated movement.

On top of this, a (hard) Winner-Take-All model was used on the relay neurons as to ensure a single firing relay neuron at a time thus having a single shifting layer active

during and after learning, as seen on figure 1.2.

It has been shown that rats already possess a rudimentary Head Direction network as early as three days before opening their eyes [Bjerknes et al., 2015] and that they quickly adapt and correct it over 24 hours after opening them using the new visual sensory information. This adaptation is believed to be done through synaptic plasticity - an ability where synaptic connections strengthen or weaken depending on their activity [Bannerman et al., 2014].

Synaptic plasticity refers to the strengthening or weakening of synapses, which increases or decreases, respectively, the transmission between the connected neurons. These long-lasting effects are commonly called Long-Term Potentiation (LTP) and Long-Term Depression (LTD) for the strengthening and weakening respectively.

One biological process where plasticity happens is Spike-Time Dependent Plasticity (*STDP*), where the timings between pre- and postsynaptic spikes drive the strengthening or weakening (hereafter *learning* or *unlearning*, respectively) of the connections. This follows the principle of Donald Hebbs famous principle, paraphrase by Carla Shatz as *"Neurons that fire together, wire together"* [Keysers and Gazzola, 2014], and gave rise to one of the oldest learning algorithms *Hebbian Learning*.

The main idea of Hebbian learning being that the closer a presynaptic neuron fires to the postsynaptic one, the bigger the change of the synapses efficieny in transferring the signal, depending on temporal precedence. Figure 1.3, from van Rossum et al. [2000], depicts this relationship, Figure 1.3b shows how the synapses potentiation grows bigger the closer the postsynaptic neuron fires right after the presynaptic one, and vice-versa.

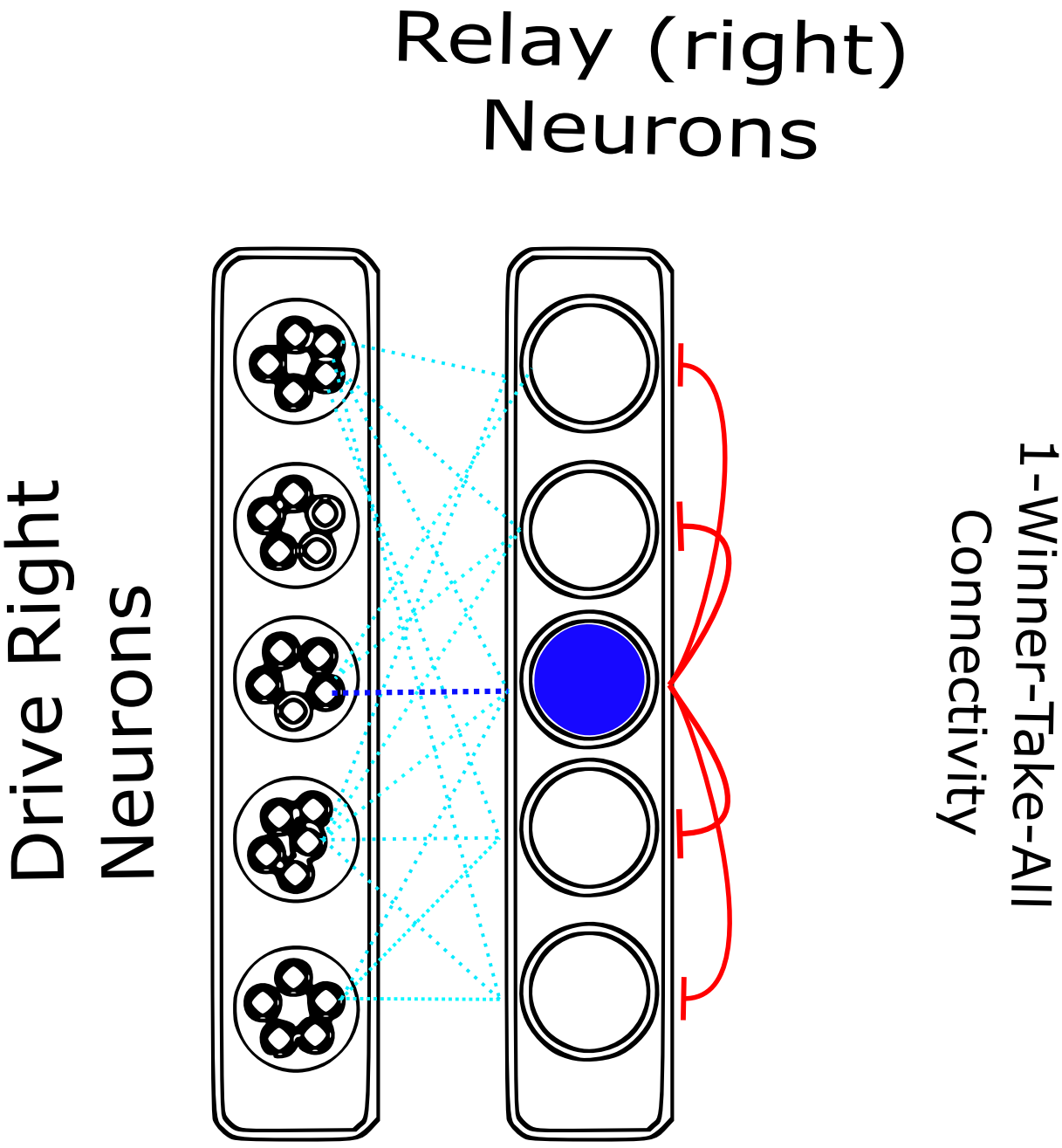

Figure 1.2: Dynamics and design of 1-Winner-Take-All. The global inhibition (sometimes accompanied by self-excitation) generates a 'One-Winner-Take-All' behaviour. Even if other cells get excited, none will spike as long as the current one keeps inhibiting them.

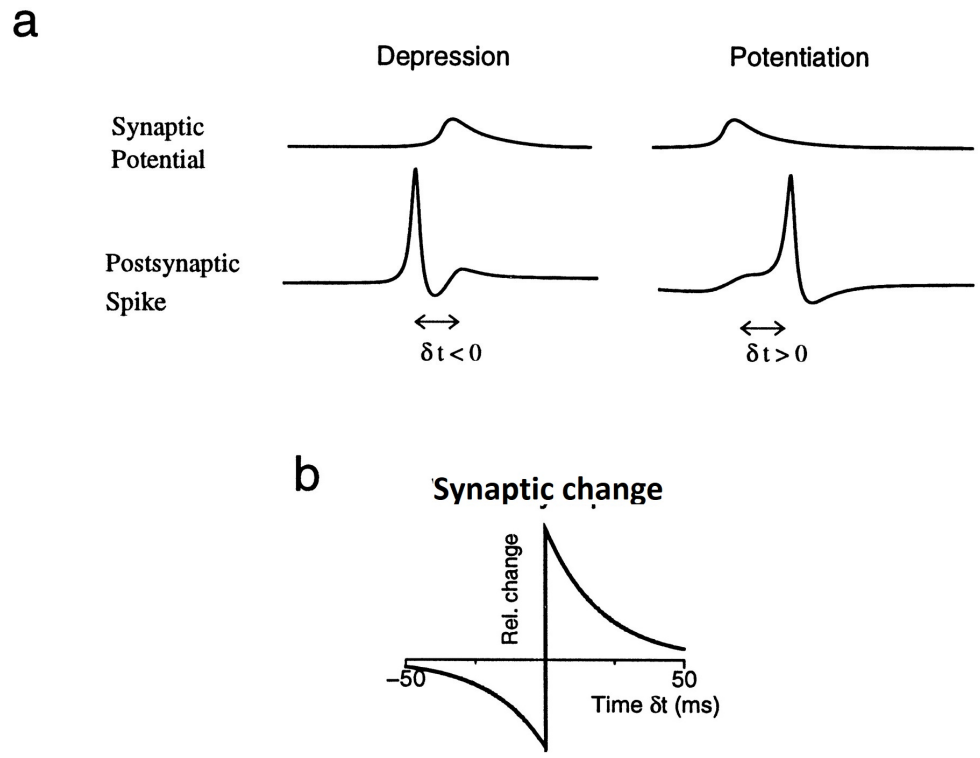

Figure 1.3: Spike timing-dependent plasticity. a, Synapses are potentiated if the synaptic event precedes the postsynaptic spike. Synapses are depressed if the synaptic event follows the postsynaptic spike. b, The time window for synaptic modification. The relative amount of synaptic change is plotted versus the time difference between synaptic event and the postsynaptic spike. The amount of change falls off exponentially as the time difference increases.[van Rossum et al., 2000]

2 Methods

This section will introduce the different software and libraries used to complete the project, as well as explain the mathematical models used to describe the neural models. After presenting the tools, a global overview of the architecture will be presented before delving into the implementation of each underlying part.

2.1 V-Rep Simulator

For this project *V-REP* (Virtual Robot Experimentation Platform [Rohmer et al., 2013]) was used to simulate the robot which would implement the neural network model later described. V-REP is available[1] on Windows, macOS and Linux. V-REP comes with many ready-to-use robots after installation which can be controlled by the users from the V-REP software interface or through a remote controller written using their remote API written in C++, Matlab or Python.
As this project is mainly written in Python, V-REP seemed to be the go-to choice for its simplicity, its price (free for educational purposes) and its pre-installed robot models.

2.1.1 Robot Choice & Setup

This project needed a robot that rotates around itself without moving too far away from its starting point, additionally it had to have the ability to give visual feedback to report if the LEDs (or structures, alternatively) were recognized, the *Pioneer3-DX* was chosen for its stability while rotating (most other ready-to-use robots included in V-REP would strongly sway during rotation which excluded them from consideration). A virtual camera was added and connected to the robots frame to obtain visual feedback. The P3DX, virtual cameras and a preview of the cameras can be seen on figures 2.1 & 2.2.

2.1.2 Light & Structure Detection

For the robot to report that it's seeing the LED (or the pylon of same color), the camera had to detect over 60% of pixels with the same color code as the visual target.

[1] `http://www.coppeliarobotics.com/`

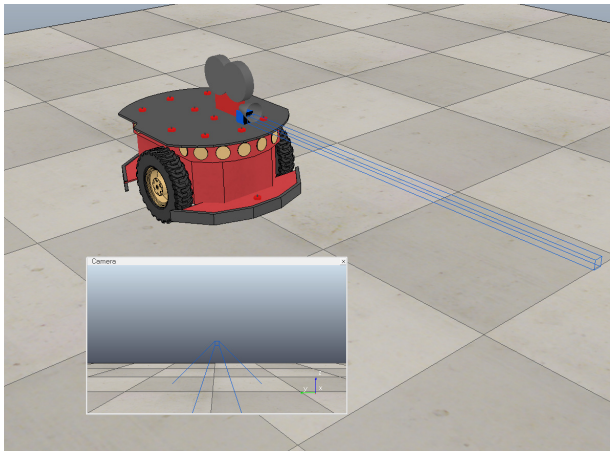

Figure 2.1: P3DX.

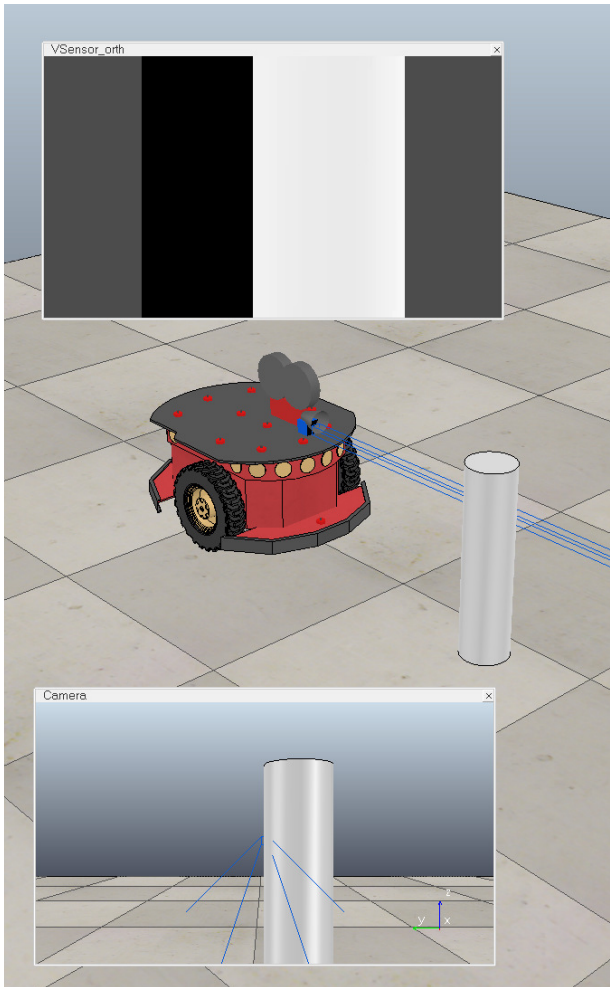

Figure 2.2: Multiple pictures of a run blended together, showing the robots trajectory.

When the condition was true and thus the visual signal detected, a flag was raised by switching a bit and reported to python.

2.1.3 Speeds

Controlling the velocity of the robot requires in principle the knowledge of the the radius, the distance between wheels. Since the robot in this work turns only around itself, the *Instantaneous Center of Curvature* lies exactly in the middle between both wheels. As such the equations the following equations for right- and left-wheel velocities from Dudek and Jenkin [2010]

$$\omega(R + \frac{l}{2}) = V_r$$
$$\omega(R - \frac{l}{2}) = V_l$$

> "Where l is the distance between the centers of the two wheels, V_r, V_l are the right and left wheel velocities along the ground, and R is the signed distance from the ICC to the midpoint between the wheels."[Dudek and Jenkin, 2010]

Since the robot in this project will only turn on itself, we have that at any point in time $V_r = -V_l$ or $V_l = V_r = 0$. Later in the project, it was discovered that for higher velocities, the robots motor would show big differences in actual movement between forward and backwards movement for a same velocity-setting.
Thus, In the end the robot was made to rotate around its right wheel by only changing the right wheels motor speed.
On V-REP, the speed of the robot is set on a per wheel basis, by setting motors velocity. This was easily done by computing $\omega(t) = \frac{v_w * r_w * t}{R_{bot}}$, where $\omega(t)$ is difference in angle (rad) per time (or *angular velocity*), v_w is the velocity of the robots wheel, R_{bot} the robots radius (0.331m for the P3DX), r_w the radius of the robots wheel (0.09751m) and t the time.
Knowing this, we can easily solve for v_w, which gives $v_w = \frac{\omega * R_{Bot}}{r_w * t}$

Figure 2.3: Multiple pictures of a run blended together, showing the robots rotation.

2.1.4 V-REP Scene Setup

The P3DX was setup on a blank scene with four LEDs, one on each cardinal direction. Later the LEDs were replaced by four pylons that could change colors on the fly, which were better and more consistently recognized during higher speed rotations. Finally, the robot was set in the middle with a single pylon somewhere around it as seen on Figure 2.3.

2.1.5 Connecting Python to V-REP

V-REP offers a remote API allowing its control over external applications. After setting up the files as described on their website[2], V-REP can be imported and run in Python as seen in the code snippet below:

```
# Import V-REPs library
import vrep
# Closes all (existing) open connections
vrep.simxFinish(-1)
# Starts communication thread with server (VREP); (try connecting)
clientID=vrep.simxStart('127.0.0.1',19999,True,True,5000,5)
if clientID!=-1:
    print ("Connected to remote API server")
    # ...
else:
    print("Not connected to remote API server")
    sys.exit("Could not connect")
```

[2] http://www.coppeliarobotics.com/helpFiles/en/remoteApiOverview.htm

2.1.6 Gathering Data

V-REP was also used to gather the data of the simulated runs. This data was stored in tuples which included the current time, a flag stating if the robot was currently seeing the LED or not, its current *true* angle (as reported from V-REP) and the nearest neuron corresponding to that angle.
This data was then saved and exported for later use. An example is included in appendix (APPENDIX)

2.2 Brian2 Spiking Neural Network Simulator

The more important part of this study is the simulation of a spiking neural network which was done on Brian2 [Goodman and Brette, 2009].
The Brian2 library is written in Python and provides predefined classes for Neurons, Synapses and much more. While these classes are given, the user is expected to specify the neuron models by giving their differential equations. Brian2 offers a tutorial[3] which was used to start off this project.

2.2.1 Example: Connecting two groups of neurons via a synapse

```
1 from brian2 import *
2
3 eqs = '''dv/dt = (1-v)/tau : 1 (unless refractory)'''
4
5 # Creating two groups of each five neurons
6 n1 = NeuronGroup(5, eqs, threshold='v>-40 *mV', reset='v=-60 *mV',
      refractory='2 *ms')
7 n2 = NeuronGroup(5, eqs, threshold='v>-40 *mV', reset='v=-60 *mV',
      refractory='2 *ms')
8
9 # Setting parameters
10 n1.tau = 10 *ms
11 n1.v = -60 *mV
12
13 # Creating synapses from n1 to n2; Setting connectivity;
```

[3] https://brian2.readthedocs.io/

```
14 synapse = Synapses(n1, n2, model='w : volt', on_pre='v_post+=w')
15 synapse.connect('i==j')
16 synapse.w = 10 *mV
17
18 # Running a simulation for 50 milliseconds
19 run(50*ms)
```

This very short example shows how a Brian2 script is written.
(Line 1) Start off by importing the Brian2 library. *(Line 3)* Set the differential equation ('*eqs*') to model the neurons behaviour. *(Lines 6, 7)* Create two group of each five neurons. *(Lines 10, 11)* set the initial values for the defined variables. *(Line 14. 16)* Creates the synapses which will connect the source neurons (n1) the target neurons (n2) - *on_pre* defines the action of the synapse when a *pre*synaptic (n1) event (spike) happens, here, the post-synaptic (n2) membrane potential will be increased by w volts, which is defined as$10\mathrm{mV}$ in line 16. *(Line 15)* connects the neurons via the created synapses, $\mathrm{i} == \mathrm{j}$ describes the synaptic connectivity. A connection is made when the index of the presynaptic neuron (i) is equal to that of the postsynaptic neuron (j) - which means each neuron from the group n1 is connected 1-to-1 to another neuron with same index on group n2. Lastly, *line 19* runs the simulation for the desired amount of time.

2.3 Mathematical Models

In addition, models from the NCSBrian2Lib - a library built at the Institute of Neuroinformatics, University of Zurich and ETH Zurich - were used to model the *leaky integrate and fire* neuron behaviour and *Fusi Synapses* to model synaptic plasticity for learning.

2.3.1 Neuron model: Leaky Integrate & Fire

$$\frac{dv}{dt} = \frac{v_{rest} - v + Rm \times (I_{in} + xi \times A_{noise} \times \sqrt{second})}{tau}$$

This differential equation defines the evolution of a neurons membrane potential. Breaking down the right-hand part of the equation:

v_{rest} = the resting membrane potential of the neuron in volts,

Rm = the membrane resistance in ohm,

I_{In} = the incoming current flow in amperes,

$xi * second^{0.5} * A_{noise}$ = the white noise equation from Brian2[4]

Figure 2.4 shows a simulation of three neurons using this model. Neuron 1 has a current flowing, while neurons 2 & 3 do not. Neuron 3 on the other hand depicts the effect of leakiness over the membrane potential, whenever it gets depolarized without spiking, the membrane potential tends to go back to its resting potential by slowly leaking.

2.3.2 Synapse model: Voltage-based

Early in the project, a simple voltage-based synaptic model was decided upon. While not being the most biologically plausible, the idea was that it would reduce the number of variables and simplify their tuning.

$$v_{post} = v_{post} + w$$

This synaptic model increases the post-synaptic neurons membrane potential by w each time the pre-synaptic neuron spikes.
This effect is shown in Figure 2.4, where neuron 1 is driven by a constant current

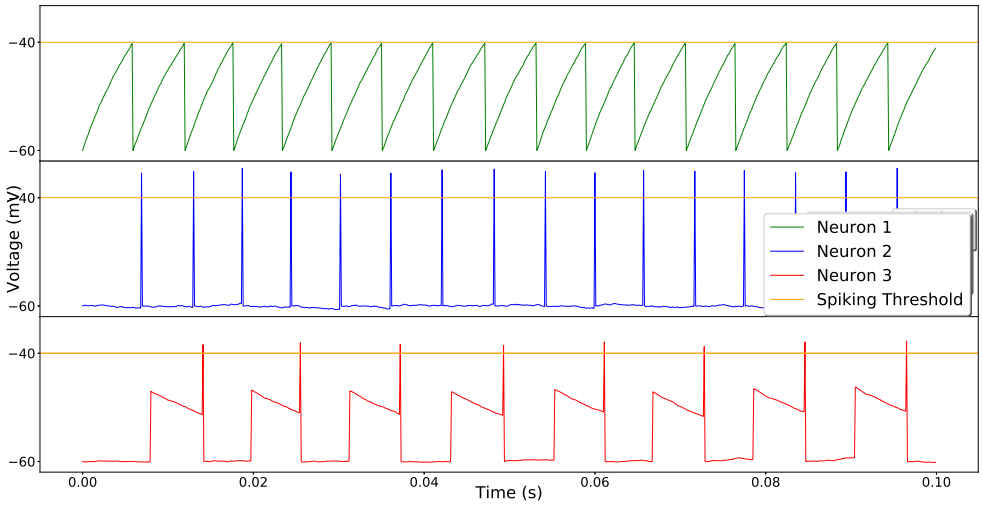

Figure 2.4: Neuron 1 (green, top) driven by a constant current, excitatorily connected to Neuron 2 (blue, middle), enough voltage is passed by the voltage-based synapse to incite a spike from neuron 2, whenever neuron 1 spikes. Neuron 2 excitatorily connects to neuron 3 (red, bottom), transfers just about 10mV which isnt enough by itself to make neuron 3 spike. The slow leakage makes neuron 3 be able to integrate two incoming spikes from neuron 2 which brings neuron 3 to threshold, thus, spiking itself.

and upon spiking excites neuron 2 by depolarizing it sufficiently to generate an action potential. Neuron 2 in turn is connected to neuron 3 but here the weight of w is such that it is not sufficient to generate an action potential on its own, it effectively requires 2 pre-synaptic spikes in a short time frame (defined by the leakiness of neuron 3).

2.3.3 Synaptic Plasticity, STDP & Fusi Synapses

Lastly this project also focused on the learning behaviour of such HD systems, as has been seen in nature in rat pups recently[Bjerknes et al., 2015].

It has been shown that rats already possess such a system in a rudimentary state as early as three days before opening their eyes [Bjerknes et al., 2015] and that they quickly adapt and correct it over 24 hours after opening them using the new visual sensory information. This adaptation is believed to be done through synaptic plasticity - an ability where synaptic connections strengthen or weaken depending on their activity [Bannerman et al., 2014].

Synaptic plasticity refers to the strengthening or weakening of synapses, which in-

creases or decreases, respectively, the transmission between the connected neurons. These long-lasting effects are commonly called Long-Term Potentiation (LTP) and Long-Term Depression (LTD) for the strengthening and weakening respectively.
One biological process where plasticity happens is Spike-Time Dependent Plasticity (*STDP*), where the timings between pre- and postsynaptic spikes drive the strengthening or weakening (hereafter *learning* or *unlearning*, respectively) of the connections. This follows the principle of Donald Hebbs famous phrase *"Neurons that fire together, wire together"*, and gave rise to one of the oldest learning algorithms *Hebbian Learning*.
To model these plastic synapses, *Fusi Synapses* were developed at INI displaying a Hebbian behaviour [Brader et al., 2007]. Under this model, LTP happens when the postsynaptic membrane potential is high, and LTD when it is low. On top of this, the postsynaptic Ca levels act as stop-learning thresholds.
Equation 1 shows the weight-update function of said Fusi synapse. This weight-update happens when there is presynaptic activity and is thus defined as the synapses presynaptic equation in *Brian2*.
Equation 2 is the models postsynaptic equation, when the postsynaptic neuron spikes, the Ca level rises. When the neuron is silent the Ca decays and neither of both conditions in equations 1 can be satisfied - there is no learning.

$$\mathrm{w_F} = \begin{cases} \mathrm{w_F} + \mathrm{w^+} & \text{if } (\vartheta_{\mathrm{up}}^{\mathrm{low}} < \mathrm{Ca} < \vartheta_{\mathrm{up}}^{\mathrm{high}}) \;\&\; (\mathrm{V_{post}} > \vartheta_{\mathrm{V}}) \\ \mathrm{w_F} - \mathrm{w^-} & \text{if } (\vartheta_{\mathrm{down}}^{\mathrm{low}} < \mathrm{Ca} < \vartheta_{\mathrm{down}}^{\mathrm{high}}) \;\&\; (\mathrm{V_{post}} < \vartheta_{\mathrm{V}}) \end{cases} \tag{1}$$

$$\mathrm{Ca} = \mathrm{Ca} + \mathrm{w_{ca}} \tag{2}$$

$\mathrm{w_F}$ represents the synaptic weight, $\mathrm{w^+}$ and $\mathrm{w^-}$ the change in weight through LTP or LTD, Ca indicates the calcium level in the postsynaptic neuron. $\vartheta_{\mathrm{up}}^{\mathrm{low}}$, $\vartheta_{\mathrm{up}}^{\mathrm{high}}$, $\vartheta_{\mathrm{down}}^{\mathrm{low}}$, $\vartheta_{\mathrm{down}}^{\mathrm{high}}$ are the thresholds in which Ca has to be in to induce LTP or LTD, *up* meaning the threshold in which the Ca-level promotes LTP, *down* conversely meaning the levels in which Ca promotes LTD. Finally and most importantly, $\mathrm{V_{post}}$ indicates the postsynaptic membrane potential and ϑ_{V} the threshold which will dictate if LTP or LTD happens depending on $\mathrm{V_{post}}$.

2.4 Implementation and intermediate results

The last couple of chapters presented the underlying models and environment that were used to inspire and implement the network. The following sections will focus on the actual implementation of the system. First a global overview of the whole architecture will be presented before delving into each separate part and their intricacies.

2.4.1 System Overview

Figure 2.5 shows the complete final version of the developed neural network. It is composed of three main parts, the main main part being the HD network which will represent the instantaneous believed facing direction by integrating rotational velocity to compute heading.
The HD networks is excitatorily connected to the first transformation array by an copy of its head direction neurons. This first transformation array driven by visual signals computes the difference between the believed heading direction and the true heading direction. The computed error is represented by a layer of neurons representing the neuron-offset between the nearest neuron to the true angle and the internally computed head direction neuron when the light was seen.
The first transformation arrays output serves as input to the second transformation array. That second transformation array is supposed to compute the corrected shift-layer to excite as to minimize the error over time. This is done by driving the corrected relay neuron which then competes against the current active one. This forces the drive neuron to learn / adapt its connection to the new chosen relay neuron.

2.4.2 Head Direction Network

The head direction network is built in layers of group of neurons. The first top-most layer in figures 2.6 and 2.7 is the HD layer which consists of the head direction neurons. Each neuron represents a certain angle depending on the number of neurons in that layer. HD neurons are interconnected in an excitatory and inhibitory fashion as to generate a soft Winner-Take-All behaviour. Each spiking HD neuron excites its nearest neighbours locally (number depending on the defined size of the interaction kernel - in this work the interaction kernel is of size 1), while simultaneously inhibiting every other

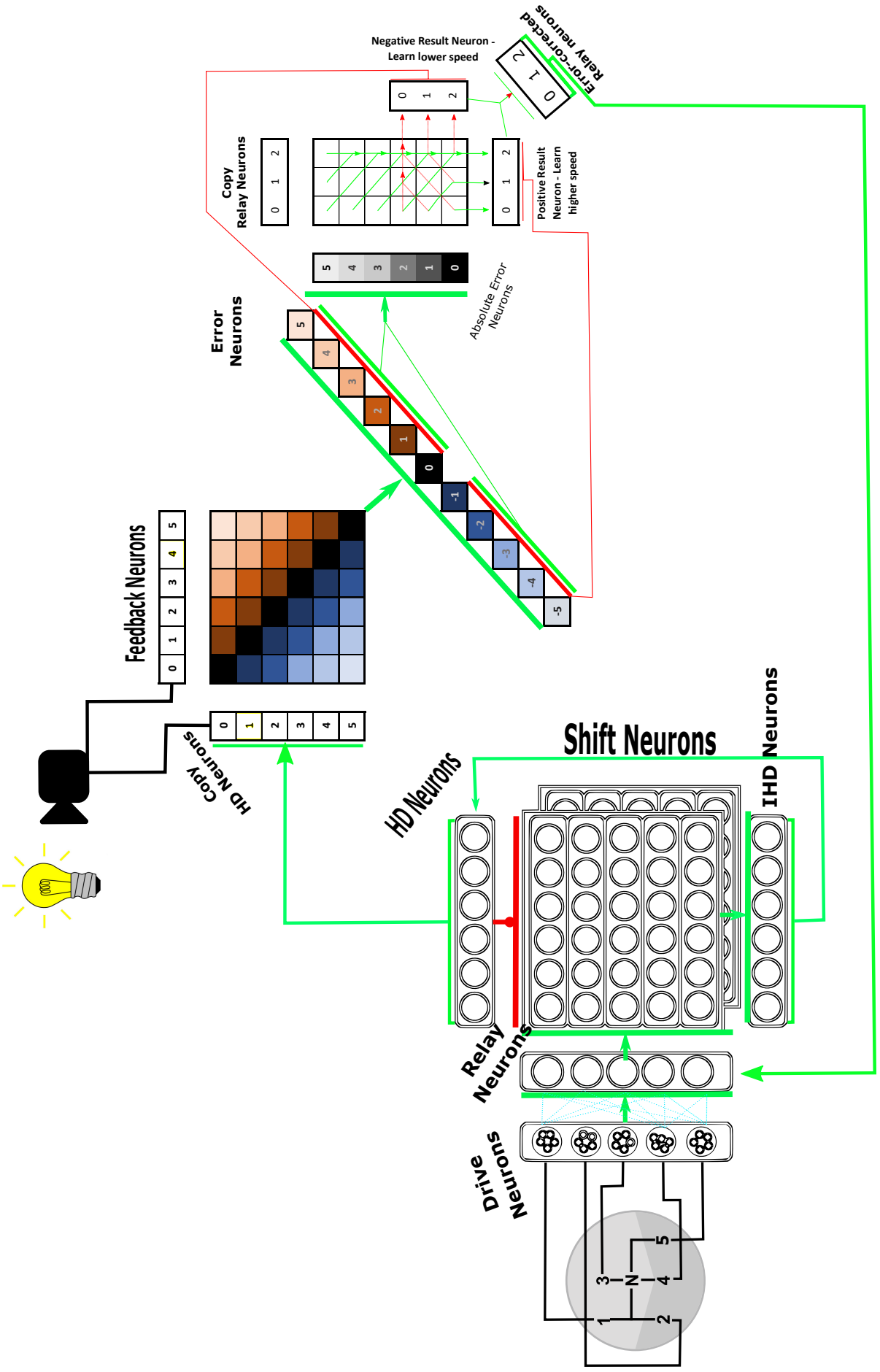

Figure 2.5: Diagram showing the full network. Drive neurons are directly driven by the speed of the robot. When the robot detects the visual signal, the transformation array activates by excitation of the copy HD neurons and feedback neurons. Green lines indicate excitatory and red lines inhibitory connections. Detailed connectivity explained under corresponding sections.

HD neuron globally. This interaction ensures the apparition of a single hill of activity. On top of this, each active HD neuron also inhibits all Shifting neurons except the ones with the same index± 1 as the active HD neuron.

The middle-part of the HD network consists of the aforementioned Shifting layers. These Shifting layers connections will define the speed at which the hill of activity will move. There is a set number of layers each representing a different level of activity-bump moving speed. Since the angular velocity can be clockwise or counter-clockwise, there are two whole sets of Shifting layers *Right Shift* and *Left Shift*. Each Shifting layer consists of the same number of neurons as in the HD layer. All shift neurons are connected *one-to-one* to the Integrated Head Direction (IHD) neurons, which also consist of the same number of neurons as in the HD layer. The connection is shifted with an offset of k-neurons, where k is the layer which was assigned to the shift neuron.

The driving force of these Shift neurons, and by extension the movement of the hill of activity, are the *Drive Neurons* which are represented left-most on the figure 2.6. Their role is to generate the spikes that will drive the whole network. They're driven by the robots speed-command themselves which injects a set current into the Drive neurons. The Drive layer consists of as many neuron sub-populations as there are Shifting layers, thus, they're also differentiated by *Drive Left* and *Drive Right*.

In between the Drive neurons and the Shifting layer are represented the *Relay Neurons*, for which there is a single neuron per Shifting layer. Each Drive neuron is connected to all Relay neurons (and vice-versa) of all sub-populations by plastic synapses. These plastic synapses strengthen or weaken their connections to the relay neurons in a *What fires together, wires together* fashion or, more precisely, following the STDP rules. This plasticity, what we call learning, happens during the window in which the robot receives the visual cue (light or structure). Outside of this window learning does not happen and the plastic synapses act as non-plastic synapses. On top of this, to ensure excitation of a single shift layer, active relay neurons inhibit all other relay neurons in a Winner-Take-All manner.

Finally, when a shifting layer is excited, the shift neurons which are *not* inhibited by the current hill of activity reach their spiking threshold and thus generate a spike. That spike is transmitted to the IHD layer in a shifted way (as mentioned ed before). IHD neurons are strongly excitatorily connected to the HD neurons with a one-to-one connectivity

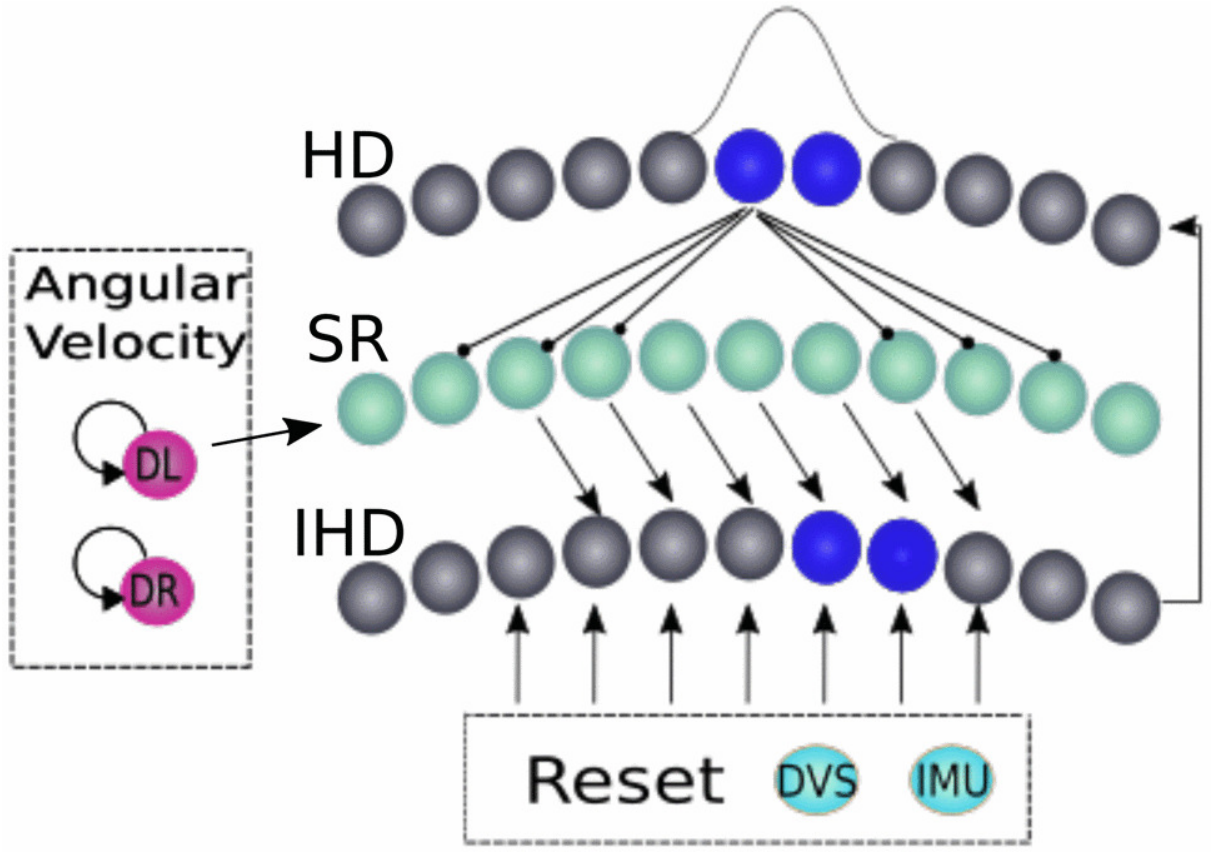

Figure 2.6: From Kreiser et al. [2018], Head Direction network using a single layer of shifting neurons. DR, DL represent 'Drive Left', 'Drive Right' neurons. Reset neurons strongly feed into IHD to incite reset of activity in HD.

to the same indexed neuron. Thus, when the IHD neuron spikes, it elicits a big spike which overcomes the global inhibition in HD layer. This new hill quickly inhibits the older hill thanks to the (soft) Winner-Take-All dynamics.

It is important to note that the different horizontal layers depicted in the figure 2.6 are in fact not connected in a straight line but form a ring, thus the ring attractor dynamics for the soft WTA.

Development & Results:

The HD network was indeed the most tricky and tedious part to implement and correctly tune in this project. The simplified Voltage-based synapse model that was used resulted in a very small range of weights for the desired results. This resulted in three similar implementations, trying to find the best fit for the project.

At the very beginning, I started implemented a very similar model to the one explained above, but where HD neurons had no inherent current. The problem was keeping a self-sustaining activity when there was no movement. Either the hill of activity would

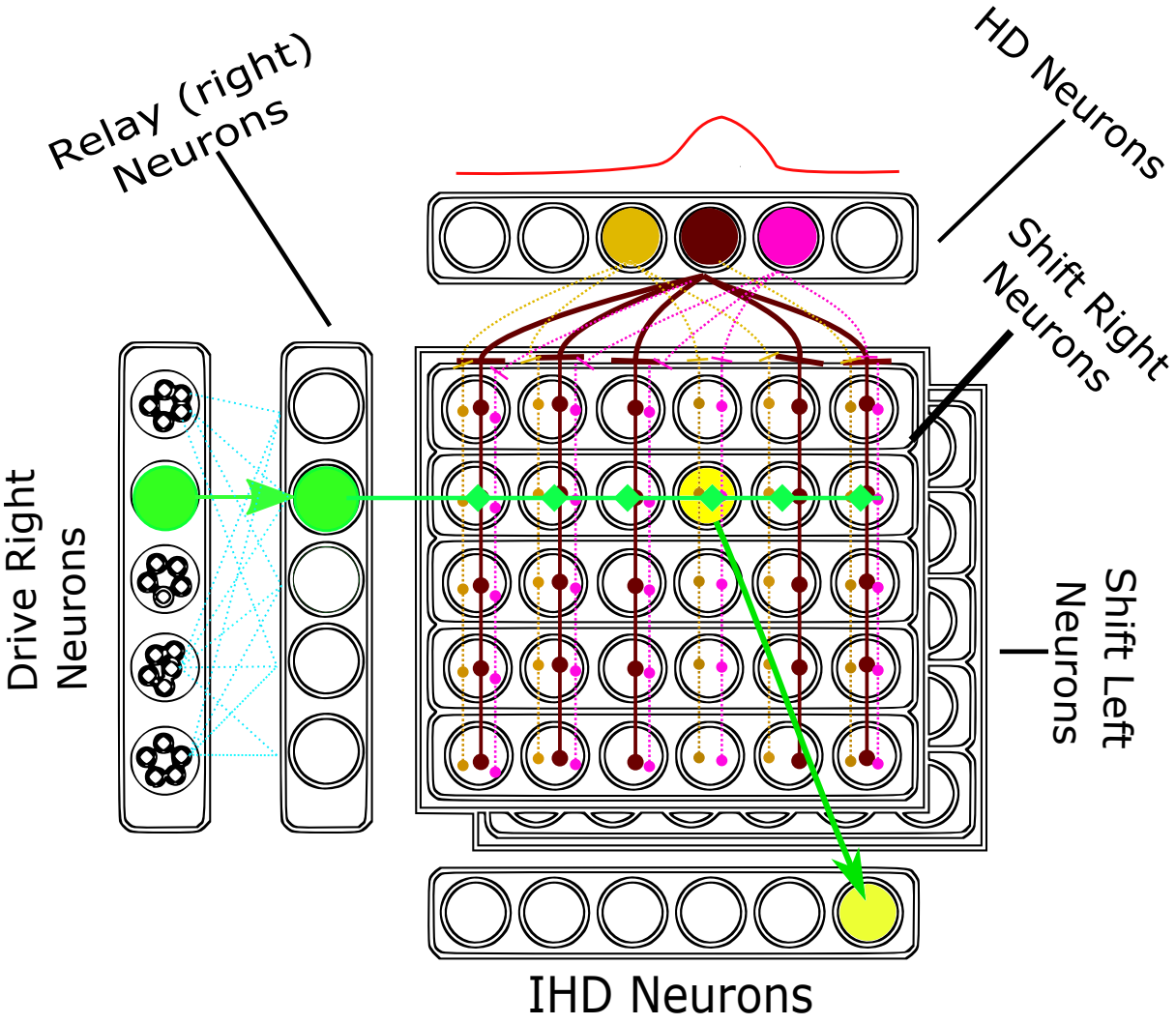

Figure 2.7: Schematic representation of the Head Direction network developed in this project. HD Neurons show a hill of activity, laterally exciting their first neighbour. Active HD neurons also inhibit every shift neuron except those with the same index. Sub-populations of drive neurons wired to specific speeds excitatorily connect through plastic synapses (light blue) to relay neurons. These relay neurons excite a whole layer of shift neurons. The resulting spiking shift-neuron excites an IHD neuron which will strongly excite the correspondent HD neuron. Green arrows represent excitatory connections. Blue dashed lines excitatory plastic connections. Gold, dark red and pink, inhibitory connections.

die and disappear, or I had to basically keep the three active neurons firing at Brian2s clock-speed - which was undesirable, and generated other problems, such as unstable hill-movement, sudden spiking of the whole network, and especially a *very* rapid movement of the hill of activity.

A second implementation was inspired by the transformation arrays which will be presented in the next section. The idea was to sustain hill-activity under no movement by adding an additional Shifting layer which would represent the stationary robot (and thus, not actually *shifting*), basically a "0-shift-layer". This implementation was rather easy to achieve, and worked rather well. But it wasn't a real self-sustaining behaviour, rather it sustained through a non-shifting activity. Also, since it was biologically not plausible, I kept searching for a better solution.

Through addition of a noise-signal, which in essence acts as a constant random input of current, to the first HD network model iteration, it tremendously helped reducing the explosive behaviour of the shifting layers - but it was random. As such, I ended up deciding to add current to drive my HD neurons. This current alone would never reach the spiking threshold. But through increase of membrane-potential by the voltage-based synapses, I could start the network. By carefully tuning the lateral excitation and global inhibition, the start of the network transformed into a hill that wouldn't die down, even without external input. Moreover, as seen in figure 2.9, the relative difference in which firing rates between input-driven (integrating angular velocity or visual cues) and self-sustain corresponded to biological findings[Shinder and Taube, 2014]!

Figures 2.9, 2.10 and 2.11 each depict the activity of 3 HD cells which get excited by IHD following a shift from the Shift^3 layer, before stopping the activity from the corresponding drive neuron. Before the hill reaches the cells, there is no spiking activity. When the shift happens (between 0.1 and 0.2 seconds - red bar), and the active IHD neuron excites its corresponding HD neurons, it spikes instantly as the excitation from IHD is huge. This first couple of spikes in the single HD cell are enough to generate a self-sustaining soft-Winner-Take-All behaviour. After the drive-neuron stops its activity, the hill-of activity stays on the same spot around the ring-network and lowers its firing rate by around 40 to 50%, from the average 130Hz during activity to around 50 to 60Hz. In parallel, the rest of the HD network was implemented. Since the HD network forms a semi-recurrent neural network, each layer influences the next one, and so on. While

trying to find a good model for the HD layer, I started implementing the shift layers.
Each shift layer comprises as many neurons as the HD layer. And one layer per set shifting-speed. Shifting speed, is the speed at which the hill of activity moves in number of neurons offset from the last state. At the beginning I started with 36 neurons per layer, which I would increase to 180 per layer by the end of the project, and 5 shifting-layers, which would increase to 10 by the end. All shift layers can be seen as a grid of N neurons by L layers. This grid is implemented twice, with mirrored connectivity, to generate a right-shift (or clockwise) and a left-shift (counter-clockwise).
Connectivity in the here developed HD network is summarized on figure 2.7. Figure 2.6 from Kreiser et al. [2018] shows the same connectivity but for the HD network which includes a single shift-layer.
As you can see on figure 2.7, the activity hill (red line on top of HD neurons) indicates where HD neurons are active. In our network, the size of the hill is defined by an *interaction kernel* of 1 - which means, HD neurons locally-laterally excite 1 neighbour per side, while inhibiting every HD neuron except itself. On top of this, HD neurons inhibitory connect to the shift neurons by inhibiting every shift neuron except the ones right beneath them. In Brian2 terms, every HD neuron inhibits every shift-neuron except the ones with corresponding index. Now all shift-neurons are inhibited, the ones beneath the hill of activity less so. As such, the membrane potential of the shift-neurons beneath the activity heal require less excitation than the rest to spike.
After implementing the relay neurons, driven by the drive neurons, I connected every relay neuron to its corresponding shift-layer. As such, when a motor-command was sent, corresponding to a drive-neuron we would excite a single row of shift layers. The cross-play of inhibition and excitation gives birth to a single spiking shift-neuron as seen on in yellow on figure 2.7.
This newly spiking shift neuron is excitatorily connected to an IHD neuron. The number of neurons it is offset depending on the layer of the shift neuron. I.e: HD_4 inhibits every shit neurons except Shifts_4 (Shifts of index 3). If the network is generating input through integration of angular velocity through the Drive^2 (drive layer 2) neurons Shift_4^2 (Shift 4 of layer 2) will be spiking. That Shift_4^1 neuron being part of layer '2' will excite an Integrated Head Direction (IHD) offset by '2'. In Figure 2.7 the network is driven by the *right* drive neuron, as such, the hill of activity will move from on top of HD_4 to HD_6 - the last neuron

represented on the same figure.

Important to remember that the whole network is connected in a circular fashion. as such, when shifting from the *last* neuron of the array in Brian2, these last neurons are reconnected to the beginning of the array.

At this step, I had a working Head Direction network. I also had a reset-neuron which would reset the activity to the current true location, through visual feedback. This can be seen as a form of loop-closure, or more precisely re-localization. But for now this network only corrects the error, which with stable conditions would just keep reappearing. The next step of the project was to make the network learn from its errors. For that I needed a way to compute the neural-offset between the true heading direction and the by the system believed facing direction.

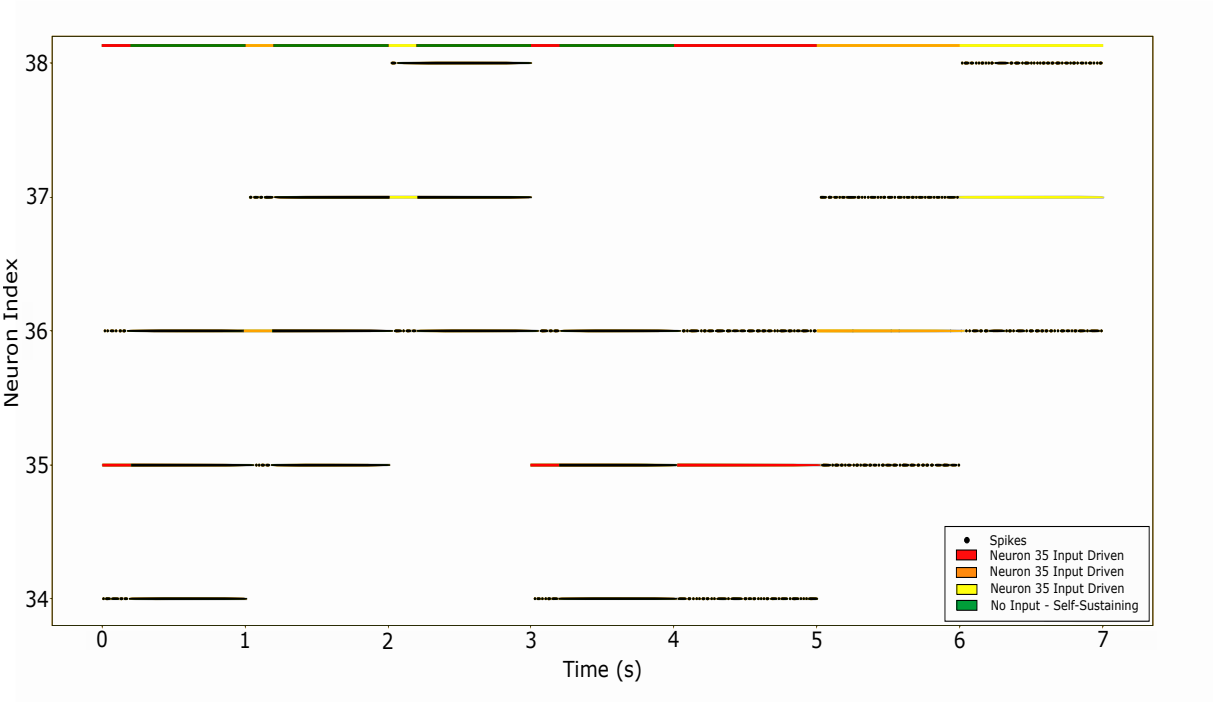

Figure 2.8: Self-sustaining 'Hill' Spiking behaviour of width 3

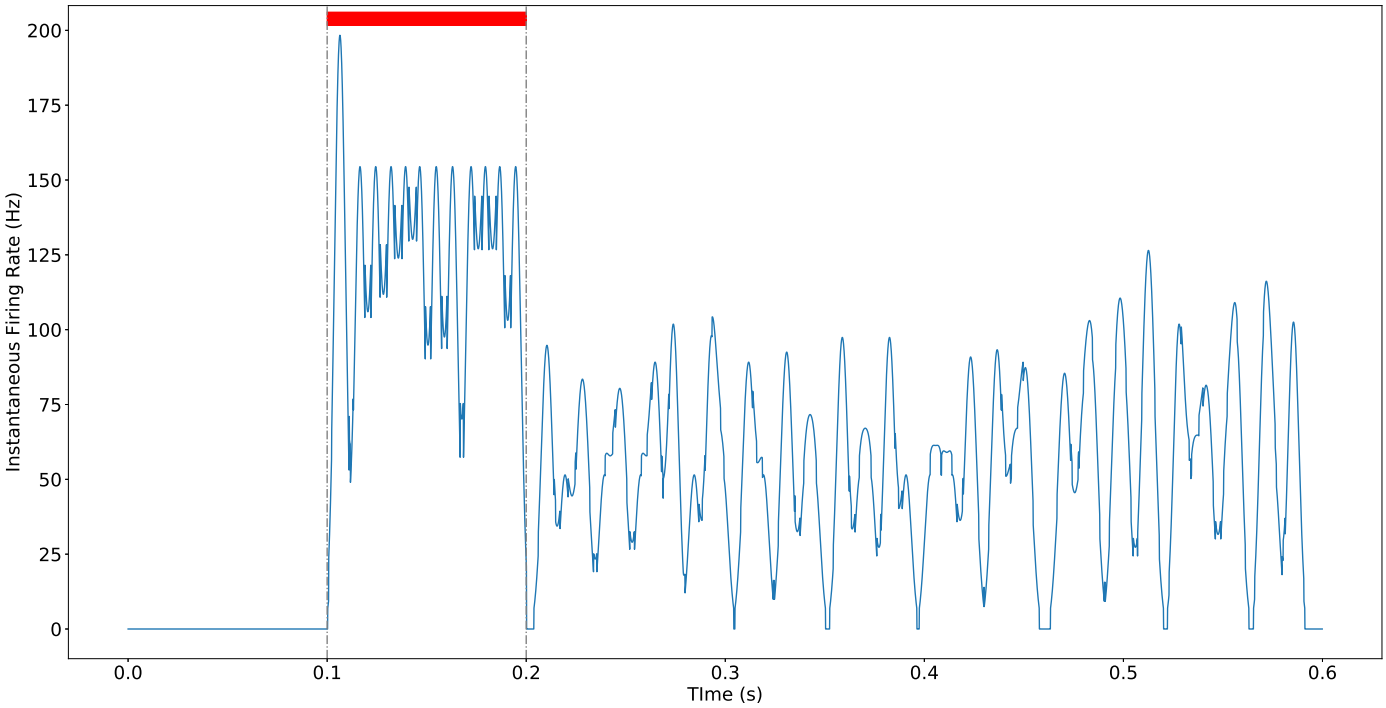

Figure 2.9: Instantaneous Firing Rate of neurons representing an angle between 55 and 61 degrees. From 0s to 0.1s, activity when facing other direction. Between 0.1s and 0.2s (indicated by red bar) during input from drive neurons (0.1s to 0.2s), indicated by red bar). From 0.2s to 0.6s when facing the preferred direction without movement.

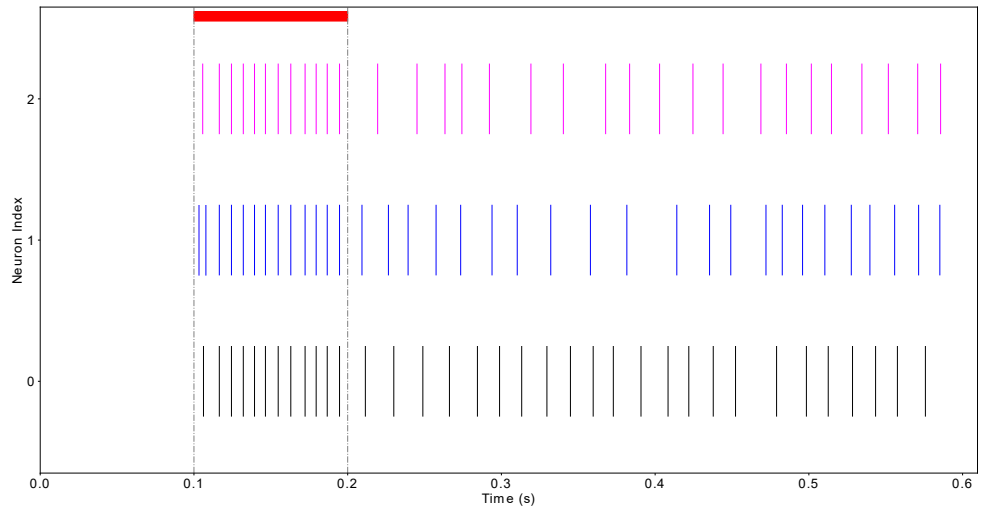

Figure 2.10: Raster plot of chosen activity hill. When facing other direction (0 to 0.1s), during input from drive neurons (0.1s to 0.2s, indicated by red bar), when self-sustaining (0.2s to 0.6s).

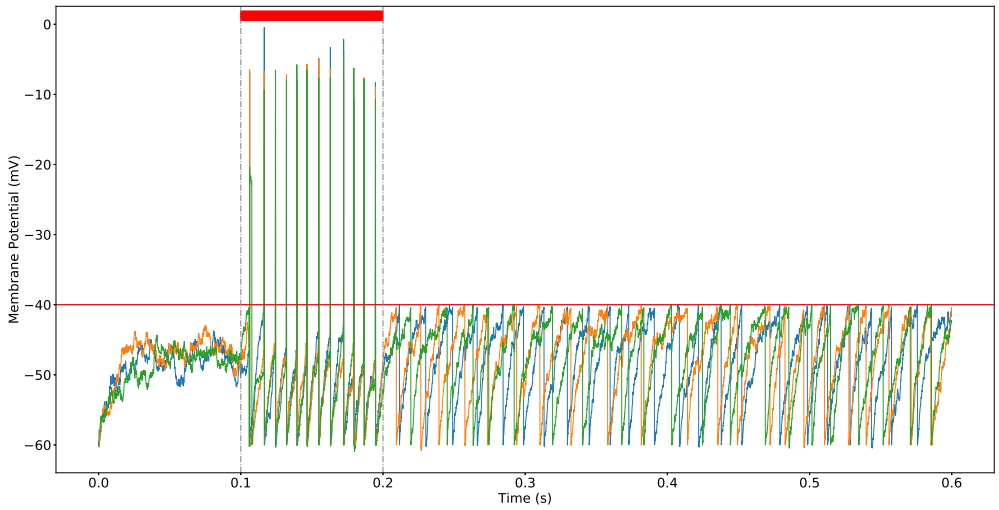

Figure 2.11: Membrane potential. When facing other direction (0 to 0.1s), during input from drive neurons (0.1s to 0.2s, indicated by red bar), when self-sustaining (0.2s to 0.6s).

2.4.3 Transformation Arrays / Relational Neural Network

Having a working HD network with self-sustain and a moving hill for different speeds, the next step was to calculate the error between the true angle and the believed angle of the system - neurally. To that end, I developed a *transformation array* (or relational neural network), which takes two inputs and transforms them into one output, similarly to a distance matrix. As inputs, it takes an efferent copy of the HD neurons (1) and a layer of as many neurons which I called *feedback neurons* (2) representing the real angle when the visual signal is seen. Both groups of input neurons connect to a grid of neurons where the computation happens in form of excitation and inhibition of each grid-neuron.

These input neurons - and by extension both transformation arrays - are driven by the robot when it sees the visual signal. Otherwise they don't show activity, nor influence the HD network.

This (first) transformation array computes the neuron offset between the active HD neuron and feedback neuron. The HD neurons excite the grid-neurons in the X axis while the feedback neurons inhibit all grid-neurons except on the Y axis corresponding to the active grid-neuron.

With this cross-play of slow big spikes of excitation from HD neurons and very quick successive little inhibition, a single neuron in the whole grid will show prolonged activity. By reading out the grid-neurons position diagonally, we can see the offset between HD and feedback neuron as shown on the heat-map on figure 2.12. To translate this neurally, I connected these grid-neurons using their respective X and Y coordinates which were set as seen in Listing 1 to the output group of neurons, which I called *Error Neurons* or *Diagonal Neurons*. These error neurons were assigned a positional value from $-\mathrm{k}$ to k, where k is the number of HD neurons (and also feedback neurons), centered on 0.

Now that I had the amount and direction of the error, I needed to translate this value in a change of shift-layer. If the HD activity hill was travelling faster than the actual robot, the model should adapt to use a lower Shifting layer, and if the hill was slower then use a higher Shifting layer.

To solve this problem, I developed a second transformation array as seen in the lower right part of figure 2.12, which took the absolute value of the error neurons from the first

transformation array and the current active relay neuron as inputs. Contrary to the first transformation array, the second transformation array has two output group of neurons: *Positive Result Neurons* and *Negative Result Neurons*. What this transformation array essentially does, is map the error during a lap at a certain speed, to a new corrected speed.

Sadly, I had to simplify this second transformation array for two reasons. First of all, the activity bump of the HD layer proved to not be consistent enough in between runs or even in a same simulation over longer run times. The other problem appeared from the speed at which the HD activity-hill moved in relation to the number of HD neurons. By moving so fast, the activity hill would sometimes overshoot its target by several whole rounds.

That's where I started to implement a group of helper neurons to keep track of the overshoot in term of laps by the believed internal map. These neurons would be excited by each their own IHD neuron, since IHD spikes are consistent and don't have that hill of activity behaviour. Figure 2.13 depicts the idea of laps being counted by different membrane potential levels. The idea was to read out this membrane potential for all helper neurons and average it to get the number of laps that passed. Sadly since IHD neurons also skip some neurons on higher speeds, this wasn't reliable... The idea was dropped, for time reasons, but by knowing the number of IHD that are shifted (which should correspond to the active relay) I'm positive we could be using this to count laps.

```
eq_IF = """
    dv/dt = (v_rest - v + Rm* (I_in + xi*Anoise*(second**0.5)))/ tau : volt
    Anoise : amp
    v_rest : volt
    Rm : ohm
    tau : second
    I_in = I_max * (vrepInp) : amp
    I_max : amp
    vrepInp : 1
    x : 1 # Position X in space
    y : 1 # Position Y in space"""

grid_Neurons = NeuronGroup((N_Neu*N_Neu), eq_IF, threshold='v>-40*mV',
    reset='v=v_rest', method='euler', name="grid_Neurons", refractory="2*ms"
    )
```

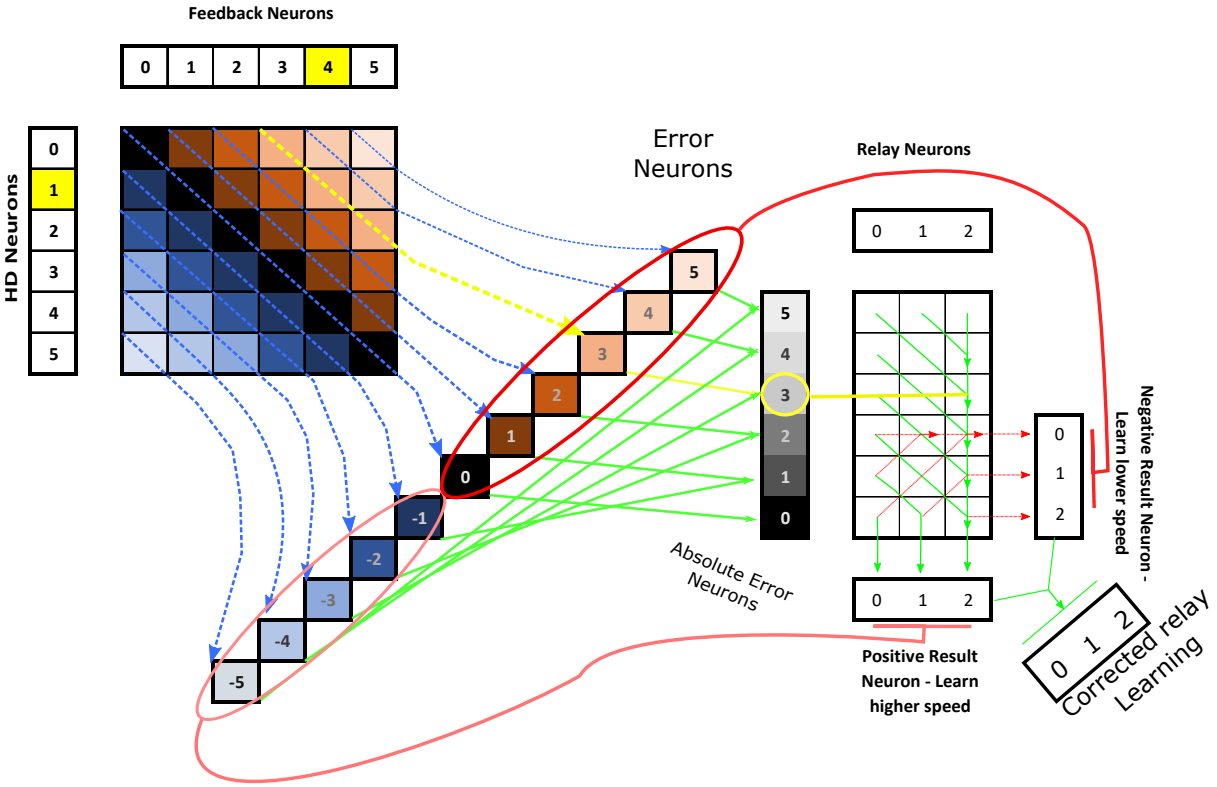

Figure 2.12: Schematic design and connectivity of the two Transformation Arrays. Not shown, The first transformation array, on the top-left, is driven by visual feedback, without it, this transformation array doesn't activate. Blue and green lines indicate excitatory connections, red inhibitory connections. Yellow shows the path taken by a signal with inputs from HD_1 neuron and $\mathrm{feedback}_4$ neuron.

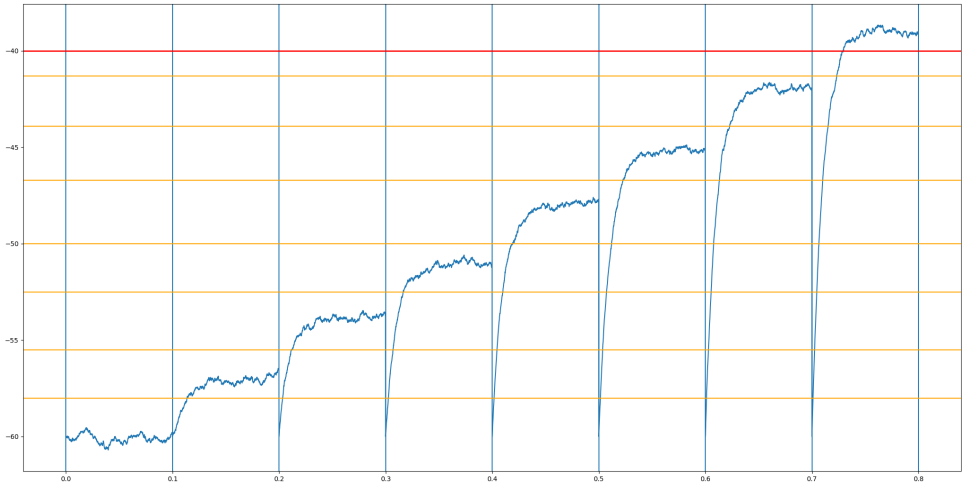

Figure 2.13: Functionality of lap counter helper neurons. Each presynaptic spike from the corresponding IHD neuron would excite the helper neuron to a certain new level which it is kept at by its current inflow. Yellow lines indicate a new lap, red is the spiking threshold.

```
14 # ...
15 # Setting grid neurons X and Y coordinates
16 grid_Neurons.x = 'i%N_Neu'              # Defining X's position relative to
       index
17 grid_Neurons.y = 'floor(i/N_Neu)'       # Defining Y's position relative to
       index
18
19 diag_Neurons = NeuronGroup(((N_Neu*2)-1), eq_IF, threshold='v>-40*mV',
       reset='v=v_rest', method='euler', name="Diag_Neurons", refractory="2*ms"
       )
20 # ...
21 # Setting diag neurons X coordinates such that it's centered on 0
22 var = (len(diag_Neurons.i)-1)/2
23 diag_Neurons.x = 'i-var'
24
25 # Connecting synapses using X and Y variables
26 synapses_cHD_Grid.connect('y_post==i')
27 synapses_Feedback_Grid.connect('x_post!=i')
28 synapses_Grid_Diag.connect(condition='(y_pre-x_pre==x_post)')
```

Listing 1: Connecting Grid Neurons to Error Neurons

Now having a way to compute the error and quantify at least the direction of the correction, I needed a way to adapt the connectivity such that the network minimizes the error over time.

2.4.4 Plastic Synapses

Lastly, with a working HD network and transformation arrays, plastic synapses were implemented between the drive neurons and relay neurons. Each drive neuron connects to *all* relay neurons of the same direction, which for 10 shifting layers makes 500 synapses per direction.

Fusi synapses show another interesting property, they show 2 states of activity, either active, or not. The code snippet below (Listing 2) shows the presynaptic equation used in Brian2 for the Fusi synapses. In *line 5* the amplitude of the weight-update is clamped between w_{min} and w_{max}, which are set to 0 and 1 respectively. In *line 6*, the post-membrane potential (v_{post}) is increased by either the full weight or nothing at all, thus the synapse behaving in a binary way.

```
preEq = '''
            up = 1. * (Ca>theta_upl) * (Ca<theta_uph) * (v_post > theta_V)
            down = 1. * (Ca>theta_downl) * (Ca<theta_downh) * (v_post <
    theta_V)
            w += (wplus * up - wminus * down) * gate
            w = clip(w,w_min,w_max)
            v_post += floor(w+0.5) * weight *mV
            '''
```

Listing 2: Fusi presynaptic equation - Binary model

Development & Results:

While implementing the model was rather trivial, understanding the underlying mechanisms and thus tuning the Fusi synapses required a bit more thoughts.

As is shown on Listing 2, the weight-update rule for the plastic synapses internal weight (w) depends on w^+, w^-, 'up' and 'down'. w^+ and w^- are the values by which the internal weight will be updated. 'up' and 'down' are binary variables which chose to either increase or decrease the weight by w^+ or w^- respectively. 'up' and 'down' are set to 1 (True) if the Calcium levels of the postsynaptic neuron is in a defined threshold and if the post-membrane potential is under or over a set threshold. The calcium simply increases by a calcium weight-variable each time the postsynaptic neuron fires.

The idea of this model is that, if a postsynaptic neuron (relay neuron in my case) fires a lot, the Calcium level will rise and reach Ca required for 'up' drift of the plastic synapses internal weight. But, also, since the postsynaptic neuron fires a lot, the postsynaptic membrane-potential will more often be above the required threshold. thus enabling this 'up' drift. On the contrary, if a postsynaptic neuron is not firing, post-membrane potential will be low, also Calcium levels will decay and both this means that the conditions for a 'down' drift of the internal weight variable are met.

Now, a new learned connection doesn't automatically *unlearn* an old one. To weaken a previous active connection, I made use of the 1-Winner-Take-All model, which means, an active neuron inhibits every other neuron. To achieve the expected result, the new postsynaptic neuron had to fire in a quicker firing rate than the previous one. I achieved this by inhibiting the drive neurons activity during seeing of the visual signal, which simultaneously acts as the visual cue to compute the error, and thus to learn new connections. Since the drive neurons are inhibited, the new relay neurons excited through

the second transformation array have a higher spiking rate - thus winning this Winner-Take-All competition and inducing strengthening of its own synapses while weakening the other ones.
Figure FIG shows the evolution of Ca levels and W levels when two relay neurons are competing. Figure FIG shows how a drive neuron drives a single different relay neuron, trying to approach the right one for the current speed.
Finally, for a stable network and a stable environment, learning won't be necessarily better than hard-wired connections to specific speeds tuned to the environment. The idea behind learning is more so that for any initial speed, the robot will be able to learn to map the best matching speed to its internal belief. The beauty behind the idea, is that flexible speeds and enough time, the robot should be able to learn the perfect matching speed for any situation.

2.4.5 Weights

Following are listed all the synaptic weights, Table 2.2 lists the different weights and thresholds for the plastic synapses, Table 2.1 lists all non-synaptic weights.
Weights & Biases:

Table 2.1: List of non-plastic synaptic weights

Description	Name	Value
Relay to Shift, Exc	syn_Relay_Shift	15mV
Left Relay to Left Shift, Exc	syn_LRelay_LShift	15mV
Shift to Integrated Head Direction, Exc	syn_Shift_IHD	40mV
LShift to Integrated Head Direction, Exc	syn_LShift_IHD	40mV
IHD to HD, Exc	syn_IHD_HD	120mV
HD Local Excitation	syn_HD_Self_Excitation	19mV
HD Global Inhibition	syn_HD_Self_Inhibition	–14.2mV
HD to Shift, Inh	syn_HD_Shift_Inh	–53.6mV
HD to LShift, Inh	syn_HD_LShift_Inh	–53.6mV
IHD to Lap Counter Neuron, Exc	syn_IHD_Round	0.3mA
HD to Efferent HD Copy, Exc	syn_HD_cHD	40mV
Error to abssolute error, Exc	syn_Diag_AbsErr	40mV
Error to PosInhib, Inh	syn_Diag_PosInhib	–100mV
Error to NegInhib, Inh	syn_Diag_NegInhib	–100mV
Pos change to Relay, Exc	syn_Pos_Relay	00mV
Pos change to Relay, Inh	syn_Pos_Relay_G_Inhib	–200mV
Pos change to LRelay, Exc	syn_Pos_LRelay	50mV
Pos change to LRelay, Inh	syn_Pos_LRelay_G_Inhib	–200mV
Neg change to Relay, Exc	syn_Neg_Relay	50mV
Neg change to Relay, Inh	syn_Neg_Relay_G_Inhib	–200mV
Neg change to LRelay, Exc	syn_Neg_LRelay	50mV
Neg change to LRelay, Inh	syn_Neg_LRelay_G_Inhib	–200mV
HD Copy to 1st Transform Grid, Exc	syn_cHD_Grid	18mV
Visual Feedback to 1st Transform Grid, Inh.	syn_Feedback_Grid	–18mV
1st Transform Grid to error, Exc.	syn_Grid_Diag	50mV
Abs. Error to 2nd Transform Grid, Exc.	syn_AbsErr_Grid	25mV
Drive Copy to 2nd Transform Grid, Inh	syn_cDrive_Grid	–20mV
2nd Transform Grid to Pos change, Exc.	syn_Grid_Pos	40mV
2nd Transform Grid to Neg change, Exc.	syn_Grid_Neg	40mV
Relay global inhibition, Inh.	wta_Relay	–15mV

Table 2.2: List of plastic synapses weights and values

Description	Name	Value
Positive weight-update variable	w^+	0.2
Negative weight-update variable	w^-	0.2
Low calcium threshold for strengthening	ϑ_{up}^{low}	0
High calcium threshold for strengthening	ϑ_{up}^{high}	10
Low calcium threshold for weakening	ϑ_{down}^{low}	0
High calcium threshold for weakening	ϑ_{down}^{high}	2
Post-membrane potential threshold	ϑ_V	-55mV
Internal weight-variable controlling slow updrift	alpha	0.0000001
Internal weight-variable controlling slow downdrift	beta	0.0000001
Calcium decay-rate	τ_{Ca}	8ms
Calciums weight variable	w_{Ca}	1
Minimum value of the synapses internal weight	w_{min}	0
Maximum value of the synapses internal weight	w_{max}	1
Threshold which stabilizes internal weight at w_{max} or w_{min}	ϑ_w	0.5
The synapses output weight, when it's active..	weight	8

3 Results

In this section I will summarize the results of the project as well include any significant results that shaped the development of this work that have not been covered yet.

3.1 Head Direction with fixed synapses and visual reset

The Head Direction Network which runs on fixed non-plastic synapses runs stable on 10 different speeds. These speeds are defined by the shifting through the shifting layers.

In the final version of the HD network, I used 180 HD neurons and 10 shift layers as a base for the whole network. This meant: 10 sub-populations of each 5 Drive neurons, 10 relay neurons, $10*180$ Shift neurons, 180 IHD neurons and 180 reset neurons. The firing rate of active neurons (except self-sustaining) was tuned to run around 130Hz.

Using this configuration in conjunction with the listed weights in Table 2.1, and a interaction kernel (lateral HD interaction) of 1, I ran several runs of 10 seconds per speed and calculated the average angular velocity as generated from the HD network, as well as the standard deviation. Figure 3.1 plots the angular speed of the 10 shift layers in angles per second and figure 3.2 plots the average speeds of the different runs adding standard deviation, we clearly can see that the 2 first shift layers behave differently. The dotted lines in Figure 3.1 indicate the two first shifting speeds which I highlighted because their values varied significantly compared to the rest. This variation comes very likely from the fact that the shifts produced by these two shifting layers (shifting by 1 and 2, respectively) acts on HD neurons right near the hill of activity. The hill of activity is always composed of 3 active neurons, and since each active neuron also laterally (by 1) excites their neighbour, the inhibition doesn't reach its global minimum until the 2nd neuron further from the main active one. To circumvent this oddity, one could increase the inhibition to act inversely to the excitation by the activity hill. This can be done by using *Mexican Hat Style* function[5] to scale the weights as to achieve a uniform activity [Schneegans, 2015], but this was considered out scope for this study.

Having the average angular velocities per shift-layer, I converted the values to use them in the robot. I then ran the robot multiple simulations of different lengths and compared

[5] https://en.wikipedia.org/wiki/Mexican_hat_wavelet

the angular error over time - a summary of the data can be found on table 3.1. Next, I reran the simulations using the visual reset. Every time the robot saw the visual signal, the reset neurons would reset the activity hill to the correct angle. Again, I plotted the networks believed direction to the true angle.

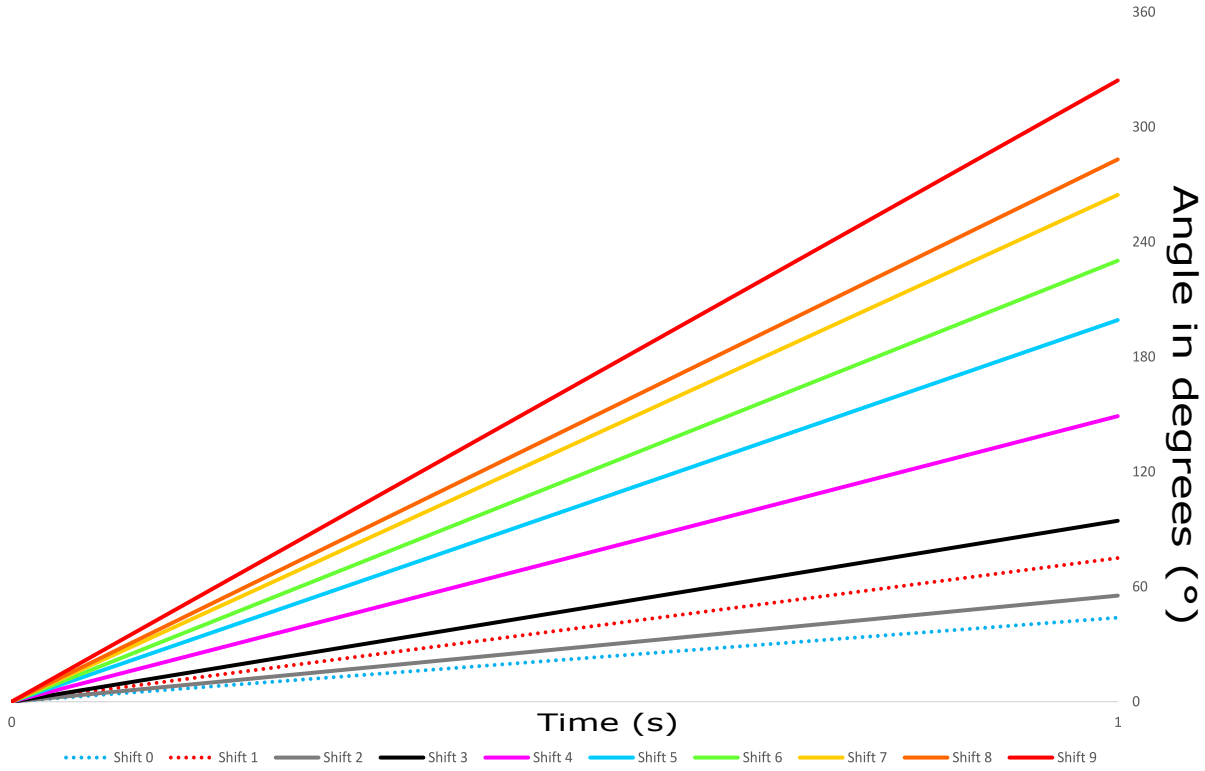

Figure 3.1: Average change in angle over a duration of 1 second, which is the angular velocity ω. Dashed lines indicate Shift layers 0 and 1 which behave differently due to local excitation from HD neurons.

3.2 Transformation Arrays

The transformation arrays are running only a small fraction of time during each run, as such, their plots show rather little insightful information. But it's important to remember that its these transformation arrays that initiate learning which is presented in the next section.

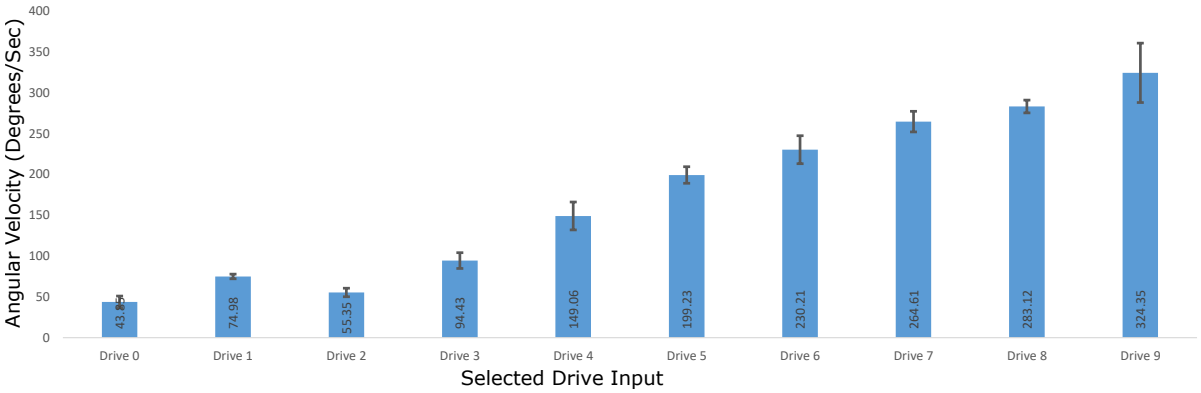

Figure 3.2: Average velocity for each drive neuron, shift-layer. We can see that the first two shift-layers shift faster in a non-linear fashion relatively to the others. This is due to the relatively low inhibition acting on the HD neurons they shift on. Bars indicate standard deviation. All data can be found on table 3.1

3.3 Learning - STDP

Unfortunately, my PC kept crashing trying to plot the weights evolution from a run, even by trying to extract only half a second of data. As such, I'll be presenting the behaviour and evolution of calcium3.3 levels and the internal weight3.4 in a small controlled simulation of 1 neuron learning to connect to another neuron and weakening its connection to a third neuron. From the *real* run, we can visualize the effect of learning by watching evolution of the relay neurons. Figure 3.5 shows exactly that. The different relay neurons are activated a single one at a time from the WTA connectivity. All these are driven by a single drive neuron which strengthens and weakens its connections to the different relay neurons at each time it sees the light. Another simulation shows a limitation of the current second transformation array and learning-behaviour 3.6; as we can see, we could think that after the first learning-pass at around 5 seconds, the network would keep that speed which seems to correspond rather well to the actual robot-speed. But as seen on figure3.7, another relay is learned and the correct one unlearned, at least until the next learning-iteration. The network continuously tries to refine its actual speed, and as such, will never end up learning a long-term connection. To correct this, the next step would be to add some error margins, since as we can see, even when the current active relay neuron incites a rather good response, the network currently forces it to learn another connection. This is due to the way that the second transform array wasn't properly tuned and requires more work. Currently it only forces the drive neuron to connect to a lower or higher relay neuron at each incoming visual

	5+ Runs of each				Average			
	2 Laps	4 Laps	10 Laps	15 Laps	Degree/Sec	Rad/Sec	STDev	
Drive 0	49.21	33.38	46.9	45.9	43.85	0.77	7.12	Diff
Drive 1	78.54	74.89	74.97	71.53	74.98	1.31	2.86	31.13
Drive 2	57.50	48.00	55.7	60.19	55.35	0.97	5.23	-19.63
Drive 3	90.14	108.13	85.91	93.52	94.43	1.65	9.65	39.08
Drive 4	136.46	134.96	153.46	171.38	149.06	2.6	17.08	54.63
Drive 5	201.38	211.93	195.81	187.79	199.23	3.48	10.14	50.17
Drive 6	244.89	222.39	243.83	209.75	230.21	4.02	17.13	30.98
Drive 7	254.56	253.06	273.12	277.68	264.61	4.62	12.62	34.4
Drive 8	272.18	288.28	282.78	289.23	283.12	4.94	7.83	18.51
Drive 9	330.95	370.35	283.43	312.67	324.35	5.66	36.38	41.23
								38.43

Table 3.1: Table showing the the average angular velocities over multiple runs of different lengths.

cue.

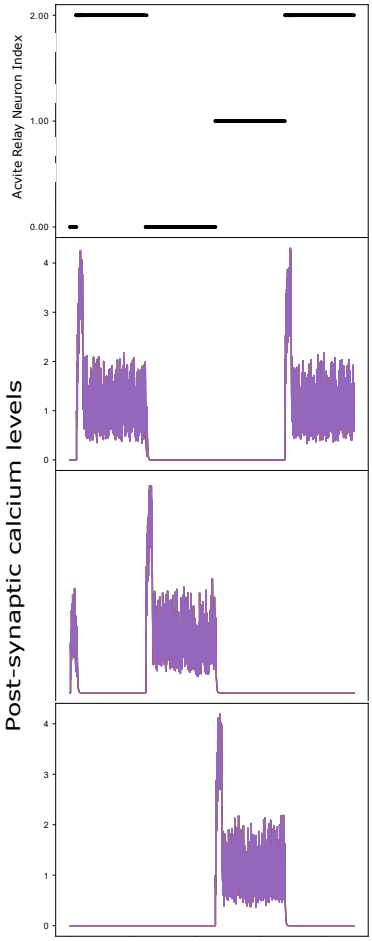

Figure 3.3: Calcium evolution during learning. On quick post-synaptic spikes-successions, the Calcium concentration goes up. The threshold to strengthen the connection being between 0 and 4 and weaken 0 and 2, going above 2 ensures learning.

Figure 3.5 shows the evolution of an active relay neuron during a run with a single speed. The selected drive neuron is initially connected to the first "out-of-the-hill" relay neuron (basically, excluding shifts 1 and 2, which I studied apart because of the difference seen before). At each visual cue, the network learns to connect to a new relay neuron, the WTA mechanism ensures that at all times a single relay neuron is active - thus stabilizing the network. – Plot: Error diminution; Fusi-weights evolution; Adaptation/Learning for multiple speeds; Error over time with/without learning

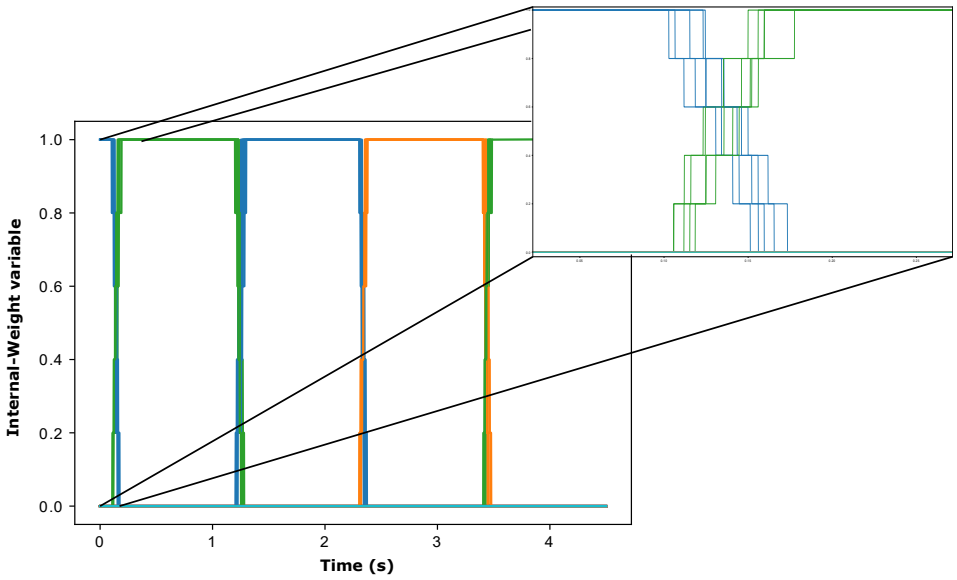

Figure 3.4: Internal weight evolution during learning. After reaching the internal-weight threshold (at around 0.5) it automatically stabilizes to 0 or 1 depending from where it reached that threshold. Zoomed-in part shows the evolution on a per-event (spike) basis.

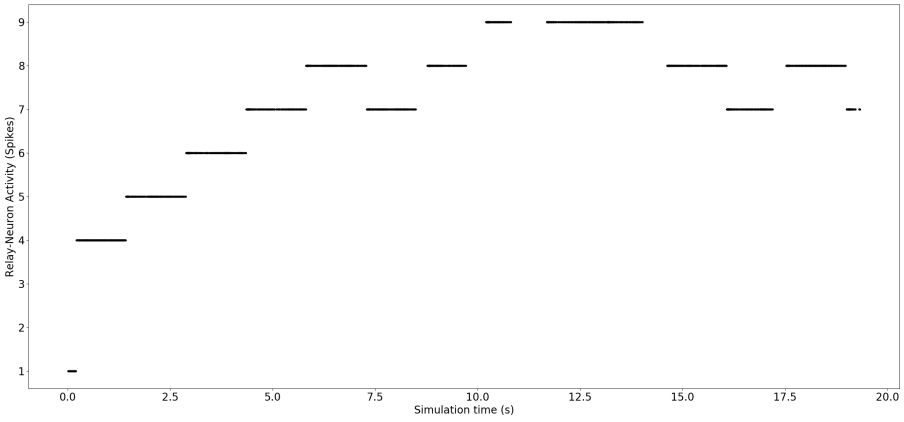

Figure 3.5: Relay neurons activity depending on current learned connections by the active drive sub-populations.

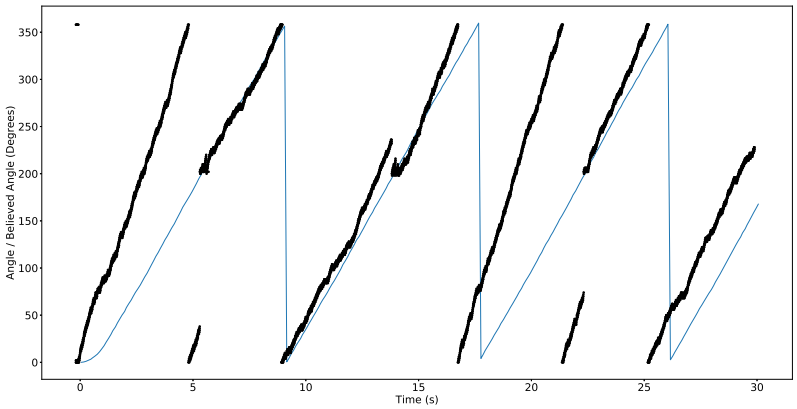

Figure 3.6: Drive Neuron continuously tries to learn new connections, even when HDs pace is very close to actual angular velocity.

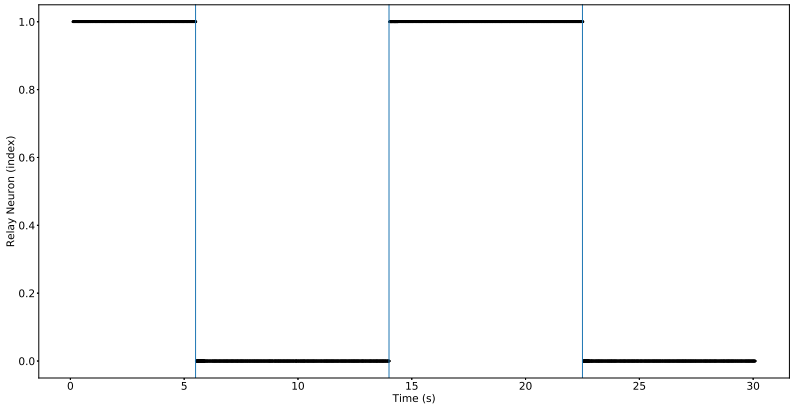

Figure 3.7: Relay neurons activity fluctuating near the correct speed, due to lack of long-term learning by considering error margins or a slower learning time (which isn't feasible with the speeds of this network. Blue lines indicate visual cue, inciting learning.)

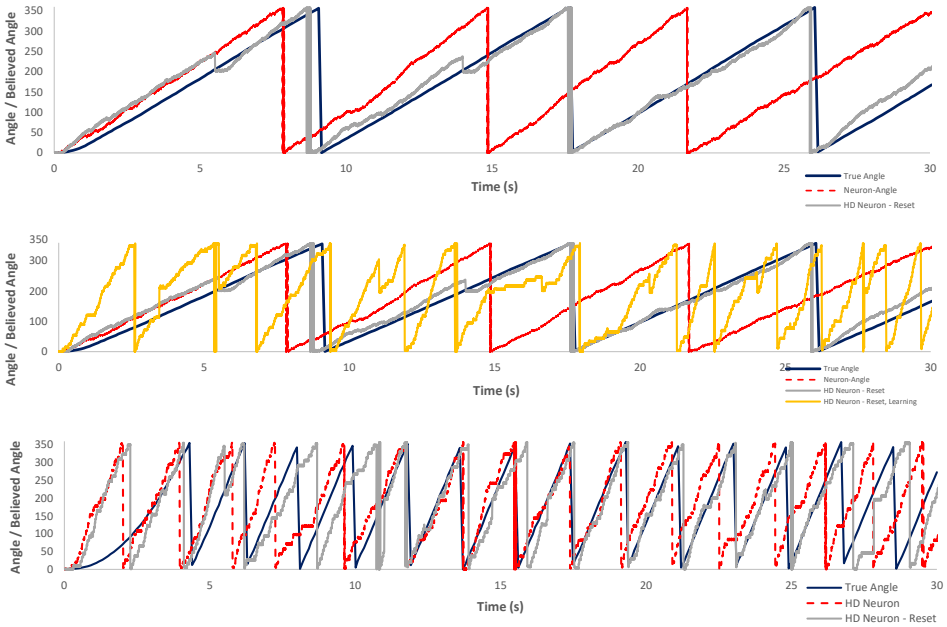

Figure 3.8: A summary to show the effect of a run without help, one using visual reset and a last one using learning. In dark blue is represented the true angle of the robot. The red line represents the run on HD without any re-localization or learning. Grey represents the run aided by reset/re-localization at each turn it sees the light. Finally, yellow represents a run with a random initial speed.

3.4 Project Results

Figure 3.8 summarizes and visualizes two runs which I found interesting. The first two plots plot the same run, the top one omits the learning run. I just wanted to highlight how a run with little noise could be very well managed with HD network only using reset. Comparing to the red line which never resets, there's a very clear difference.

The second plot was an interesting contrast to figure 3.6. While learning could sometimes work perfectly like in 3.6, sometimes with bad luck, it would keep overshooting or undershooting its target like in the second plot of the current example in yellow. But it kept trying to approach the right speed.

Finally, the last plot was something related to the robot, which I didn't think about until the very end. For shorter runs, the acceleration required by the robot to reach the defined speed can take a very long time.

4 Discussion

During this thesis major findings of the study - Achieved HD Implementation, developed 3 models for self-sustain, SCALABLE!!! – 1) voltage-based self-sustain (+: easy to implement, -: Spiking rate not under control (spikes at each clock-step), dies with refraction, heavily influenced by noise (noise can either kill the peak, or create a second peak leading to explosion of spikes in the whole network, since it ticks so fast, global inhibition and local excitation are hard to control) – 2) Transformation-array inspired, using "0th" speed-layer (+: rather easy to implement, -: not biologically true, no "true" self-sustain – 3) Semi current-driven: - Developed and Implemented 2 neural operations through transformation arrays – Difference, +/- - Learning works, rudimentarily - More Neurons = More precision = More stable = More speeds (for voltage-based model)

- Could help solve learning/adapting of speeds (i.e. different environmental situations affecting roboter speed on-terrain) for autonomous robots

- While scalable, shifting-speed is less easily changed

– Means, with little number of neurons, little precision

- Transformation arrays: Practical way to compare similar info stored by neurons

4.1 Issues & Limitations

While working on this project, several issues and limitations appeared, these will be listed in this following paragraph, with ideas or advises for future works that didn't make it into this thesis.

The very first big issue was the synchronization between V-REP and Brian2. Brian2 requires time computing the neural network. We tried synchronizing V-REP to Brian2, but everything seemed to fail. We thought about using ROS (*Roboter Operating System*[6]), but I ended up gathering V-REPs run-data and using it as input for Brian2.

Another big limitation, I felt, was the use of purely voltage-based synapses and neurons. While it simplified the implementation. It severely limited the tuning. Most neurons presenting no current inflow severely complicated the control of spiking behaviour and spiking rate.

[6] www.ros.org

As such, the only way to control the speed of HD, was increasing the number of neurons. Which with all connectivity depending on this size, exponentially grew the size of the network - especially the number of synapses - resulting in memory errors and overflow when trying to record variables using StateMonitor.
The resulting speed in relation to the size of the network that was used was still very fast. In learning, for higher shifting-layers, the HD network would overshoot the rotation by several full-rotations if the initial wrong relay was too far from the true one. I tried fixing this by adding a neuron which would count the laps by increasing its current inflow. But for higher speeds, so many HD neurons were shifted that counting their activity became unreliable.

4.2 Further Work

As mentioned earlier, while my current implementation of learning isn't complete nor 100% reliable, I really do think it's an achievable goal and something worth studying and working on. As next steps for further research, I would try solving the problem of lap-counting, for when there are huge miss-matches between actual angular velocity and believed angular velocity. Further, I would explore the options to have flexible speeds instead of set speeds, this would allow for a much smoother adaptation. The second transformation array which maps error to the new relay-neuron also seems promising, and I'll be personally exploring this in the near future. Finally, adapting the local excitation-inhibition ratio as to have a smooth shifting curve with different speeds, by using a function such as Difference of Gaussian or 'Mexican Hat'.

5 Conclusion

The aim of this study was to implement a neurally inspired Head Direction network, including a self-sustaining hill of activity, and shifting behaviour through angular velocity integration. Further, a visual reset to re-localize the HD activity correctly was implemented. Error-computation and error-correction are achieved during the reset. Lastly, learning through synaptic plasticity was studied and implemented.

As has been shown throughout this thesis, it is possible to implement the neurally inspired Head Direction network using the Brian2 spiking neural network simulator. Additionally, three three different ways to induce self-sustain behaviour to create the typical hill of activity on the ring-attractor network have been developed. Each with their own advantages and limitations. For the rest of this study, a single model was chosen.

Modulation of the travel-speed of the activity-hill on the ring attractor network was possible through addition of shifting layers. Each new shift bringing a bigger neural shift corresponding to a set increase in speed. This method of shifting layers to generate different hill-movement speeds may result in simpler implementations but come with the drawback of having a non-continuous speed-curve but instead set speeds. One limitation to consider in this study being that the local excitation causes the shifting layers that act inside the local excitation in the HD layer to behave differently than the rest.

Additionally, a neural architecture has been developed allowing error computation between 2 groups of neurons representing similar information. This architecture also allows the reclassification of information and thus approaches loop closure through error-correction and learning / adaptation.

Learning was the last studied part of this project, it has shown successful and promising results on single-trials, although there are problems with higher speeds because of the short time available to learn. This has been corrected for higher speeds through a non-neural solution, by simply artificially increasing the time available for learning.

To improve the system, the speed of the hill movement should be able to change in smaller iterations. This could be achieved by either increasing the number of neurons - which would cost more computation power, or add a current or conductance based synapse model which would permit a better fine-tuning of the system. Further, the

second transformation array should be tuned, or set to learn, the error-margins to adapt connectivity to a new relay neuron. As mentioned before, the local HD inhibition should be adapted to counteract on the lateral excitation as to smooth-out the neural response from incoming shifts through the IHD neurons. With these improvements in mind, the network could enhance and help solving a fully functioning loop-closure system for SLAM, allowing for fully autonomous robots.

References

Michaël B. Zugaro, Angelo Arleo, Alain Berthoz, and Sidney I. Wiener. Rapid spatial reorientation and head direction cells. *The Journal of Neuroscience*, 23(8):3478–3482, 2003. doi: 10.1523/jneurosci.23-08-03478.2003.

M. C. W. van Rossum, G. Q. Bi, and G. G. Turrigiano. Stable hebbian learning from spike timing-dependent plasticity. *The Journal of Neuroscience*, 20(23):8812–8821, dec 2000. doi: 10.1523/jneurosci.20-23-08812.2000. URL `https://doi.org/10.1523/jneurosci.20-23-08812.2000`.

Raphaela Kreiser, Matteo Cartiglia, Julien N.P. Martel, Jorg Conradt, and Yulia Sandamirskaya. A neuromorphic approach to path integration: A head-direction spiking neural network with vision-driven reset. *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018. doi: 10.1109/iscas.2018.8351509.

Tale L. Bjerknes, Rosamund F. Langston, Ingvild U. Kruge, Edvard I. Moser, and May-Britt Moser. Coherence among head direction cells before eye opening in rat pups. *Current Biology*, 25(1):103–108, 2015. doi: 10.1016/j.cub.2014.11.009.

JS Taube, RU Muller, and JB Ranck. Head-direction cells recorded from the postsubiculum in freely moving rats. i. description and quantitative analysis. *The Journal of Neuroscience*, 10(2):420–435, 1990. doi: 10.1523/jneurosci.10-02-00420.1990.

Jeffrey S. Taube. The head direction signal: Origins and sensory-motor integration. *Annual Review of Neuroscience*, 30(1):181–207, 2007. doi: 10.1146/annurev.neuro.29.051605.112854.

Pengcheng Song and Xiao-Jing Wang. Angular path integration by moving "hill of activity": A spiking neuron model without recurrent excitation of the head-direction system. *Journal of Neuroscience*, 25(4):1002–1014, 2005. doi: 10.1523/jneurosci.4172-04.2005.

Sung Soo Kim, Hervé Rouault, Shaul Druckmann, and Vivek Jayaraman. Ring attractor dynamics in the drosophilacentral brain. *Science*, 356(6340):849–853, 2017. doi: 10.1126/science.aal4835.

Robert G. Robertson, Edmund T. Rolls, Pierre Georges-François, and Stefano Panzeri. Head direction cells in the primate pre-subiculum. *Hippocampus*, 9(3):206–219, 1999. doi: 10.1002/(sici)1098-1063(1999)9:3<206::aid-hipo2>3.3.co;2-8.

Arseny Finkelstein, Dori Derdikman, Alon Rubin, Jakob N. Foerster, Liora Las, and Nachum Ulanovsky. Three-dimensional head-direction coding in the bat brain. *Nature*, 517(7533):159–164, 2014. doi: 10.1038/nature14031.

Kechen Zhang. Representation of spatial orientation by the intrinsic dynamics of the head-direction cell ensemble: a theory. *The Journal of Neuroscience*, 16(6):2112–2126, 1996. doi: 10.1523/jneurosci.16-06-02112.1996.

William E. Skaggs, James J. Knierim, Hemant S. Kudrimoti, and Bruce L. McNaughton. A model of the neural basis of the rats sense of direction. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 173–180. MIT Press, 1995.

Michael E. Shinder and Jeffrey S. Taube. Self-motion improves head direction cell tuning. *Journal of Neurophysiology*, 111(12):2479–2492, 2014. doi: 10.1152/jn.00512.2013.

Yanqing Chen. Mechanisms of winner-take-all and group selection in neuronal spiking networks. *Frontiers in Computational Neuroscience*, 11, apr 2017. doi: 10.3389/fncom.2017.00020. URL https://doi.org/10.3389/fncom.2017.00020.

Patricia E Sharp, Hugh T Blair, and Jeiwon Cho. The anatomical and computational basis of the rat head-direction cell signal. *Trends in Neurosciences*, 24(5):289–294, may 2001. doi: 10.1016/s0166-2236(00)01797-5. URL https://doi.org/10.1016/s0166-2236(00)01797-5.

David M. Bannerman, Rolf Sprengel, David J. Sanderson, Stephen B. McHugh, J. Nicholas P. Rawlins, Hannah Monyer, and Peter H. Seeburg. Hippocampal synaptic plasticity, spatial memory and anxiety. *Nature Reviews Neuroscience*, 15(3):181–192, mar 2014. doi: 10.1038/nrn3677. URL https://doi.org/10.1038/nrn3677.

C. Keysers and V. Gazzola. Hebbian learning and predictive mirror neurons for actions, sensations and emotions. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 369(1644):20130175–20130175, apr 2014. doi: 10.1098/rstb.2013.0175. URL https://doi.org/10.1098/rstb.2013.0175.

E. Rohmer, S. P. N. Singh, and M. Freese. V-rep: a versatile and scalable robot simulation framework. In *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

Gregory Dudek and Michael Jenkin. *Computational principles of mobile robotics*. Cambridge University Press, 2010.

Dan Goodman and Romain Brette. The brian simulator. *Frontiers in Neuroscience*, 3: 26, 2009. ISSN 1662-453X. doi: 10.3389/neuro.01.026.2009. URL https://www.frontiersin.org/article/10.3389/neuro.01.026.2009.

Joseph M. Brader, Walter Senn, and Stefano Fusi. Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural Computation*, 19(11): 2881–2912, nov 2007. doi: 10.1162/neco.2007.19.11.2881. URL https://doi.org/10.1162/neco.2007.19.11.2881.

Sebastian Schneegans. *Dynamic Field Theory of Visuospatial Cognition*. PhD thesis, 03 2015.