Developing AntBot:
A navigational system inspired by the insect brain

Robert E. F. Mitchell

Master of Informatics
Informatics
School of Informatics
The University of Edinburgh
April 4, 2019

Supervised by
Dr. Barbara Webb

Acknowledgements

I would like to thank the following people, each of whom as provided support, academic or otherwise throughout this work.

- Prof. Barbara Webb - Thank you for your guidance and insight throughout the project. The support and constant feedback you have provided in your role as supervisor have been invaluable.
- The insect robotics group - I would also like to thank the wider insect robotics group, all of whom have been approachable and open to discussion throughout. In particular, I would like to thank Jan Stankiewicz. Our conversations regarding the operation of the Central Complex and biological approaches to optical flow sensitivity in insects greatly aided my understanding of the topics and helped me develop much of the work in this paper.
- My parents: Tom, and Linda Mitchell - Your unwavering support throughout my education, of my subject choices, and of my goals of further study have allowed and encouraged to push beyond my limits and work to the best of my ability. I can never thank you enough.
- My closest friends: Marko Kerr, Ethan Piddington, and Nik Hogarth - Regardless of how busy or detached I have been through this past year, I have always known I could step back into the circle as if no time had passed at all. Thank you for being there.
- John "Hedge" Hall - It is fair you to say that you have been somewhat of a mentor figure throughout my University career. In the last two years, our conversations on academia, success, education, and whatever philosophy you happened to be reading about at the time, have helped me maintain a healthy perspective on life. You introduced me to the practice of meditation, which has helped me maintain a cool head, where previously I would be absorbed by stress. Thank you for giving me a more robust approach to mental health, and for showing me that getting older does not mean growing up.
- Alana L. White - Your enduring support and love throughout these last years has kept me grounded in an otherwise turbulent time. You were always there for me when I needed you most but best of all, you were there for me when I did not. You were the constant on which I could always rely. Thank you, for everything.

Declaration

I declare that this dissertation was composed by myself, the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Robert Mitchell

Abstract

Ants are incredible navigators. With limited sensory and computational capabilities they are able to perform robust navigation over relatively vast distances, through varying terrains and lighting conditions. How this is done is not quite known, however models for individual aspects of the navigational process have been constructed and demonstrated in recent years in simulation, and on robot platforms. We take two such models, the Mushroom Body circuit for Visual Navigation and the Central Complex model for Path Integration. We combine these with a biologically plausible neural model for collision avoidance to provide a conceptually complete *base model* for insect navigation, the Extended Central Complex. This model has been implemented on a robot platform, AntBot. In an effort to develop the model we also present several advancements to the AntBot platform, experimentation with an alternative optical flow method for collision avoidance, experiments with the Central Complex model to address some isssues with previous tests, and promising (though informal) early results indicating the plausibility of our approach. Finally, we present suggestions for future work on both the component models and our combined model, as well as recommendations for future development of the AntBot platform.

Contents

List of Figures

List of Tables

1 Introduction

Navigation is a complex task. Determining a sequence of actions to reach a known location based on a combination of sensory inputs requires a lot of computational power. Desert ants are capable of performing such a task over comparitively vast distances with limited, low resolution sensory information and remarkable efficiency. While the exact method by which the ants perform this task is still unknown, a reasonably complete navigational model can be constructed from existing physiologically plausible components, which may mimic the insect behaviour.

In this paper we introduce a theoretical combined model, the Extended Central Complex (ECX) model for insect navigation. To be clear, there is no (known) physiological basis for such a model; however, it is biologically feasible and may provide insight into the operation of the real insect brain. The ECX model combines the tasks of Visual Navigation, Path Integration, and Collision Avoidance; using the Mushroom Body Circuit (MB) [1], the Central Complex model (CX) [13], and Optical Flow Collision Avoidance (OFCA) [6, 12] for each task at a low level, then combining their outputs to get a form of higher navigational processing (similar to the weighted "base model" described in [16]). The ECX model is a modified Central Complex model, named simply to ensure distinction between the two models. The individual components are all biologically plausible and two of three (the MB and CX) are known to have a physiological basis. [1, 13, 6, 12, 10].

This is Part 2 of the MInf qualification; the project primarily extends the work done in [6] (Part 1). Therefore we continue using the AntBot platform; a robot constructed for the express purpose of experimenting with the algorithms in the *Ant Navigational Toolkit* [2, 17].

1.1 Motivation

Currently, a full base model for insect navigation does not exist [16]. We here aim to take the abstraction presented by *Webb* and create a biologically plausible implementation using our three-system approach. Both the MB and CX models have been implemented and tested on AntBot previously [9, 6, 2, 19], and a model combining the two has also been constructed by *Zhang* [19]. This combined model is used as a basis for our own and will be discussed further in Section 2.5.

The previous AntBot implementations have demonstrated good performance of the CX and MB models individually [9, 6]. Performance of a combined system has also been shown to be reasonable, however, it is less consistent than we would desire [19]. In the combined model tests from [19] we note two key methodological problems: a fixed outbound route, and fixed component weightings; the fixed outbound route was also present in the CX experiments conducted by *Scimeca*. We address the former by adding the OFCA component to our model; as in [6], the AntBot will follow a non-deterministic outbound route through an arena with obejcts present. The latter brings up the more complicated question of plausible synaptic plasticity (i.e. modifying component weightings dynamically in some fashion) which, though undoubtedly interesting, lies outside the scope of this project (though we will discuss the concept to a limited extent). It is worth noting that there may also have been unknown tecnical issues with the robot which affected the results of [19], making them less consistent than they should be in reality (see [6] and later in this work).

While [6] provides a reasonable collision avoidance system based on optical flow, it does not fit so intuitively into the CX architecture. Therefore we aim to explore an alternative yet still biologically plausible collision avoidance system which will allow us to develop the ECX model.

Our ultimate goal is to provide some insight into the precise biological systems in play during a point-to-point navigational task.

1.2 Practical Goals

The original aim of this project was to build on the experimental scenario from [6]. The robot would be tested by allowing it to navigate through a cluttered environment using a collision avoidance system. The navigational systems would then be tasked with bringing the robot home through the same cluttered environment using a combination of visual information, a path integration vector, and collision avoidance.

In order to achieve this experimental goal, the project was broken down into four stages:

1. The first stage involved solving some choice technical issues picked up by [6]; making any hardware/software adjustments required to provide a solid foundation on which to develop.

2. We then began investigating existing systems. This stage will involved research and review of new topics (the main one being the Central Complex model for Path Integration), and their implementations on the robot (if present). This stage also looked to test the CX model in a non-deterministic navigational task (see Section 4).

3. This stage involved the set up and testing of the individual components of the ECX model. Building the modified optical flow system, using the work from *Zhang* to combine the MB model with the CX, and finally, putting the three pieces together to form a complete implementation.

4. Finally, the collection and compilation of results from the combined system and the individual systems.

1.3 Results

This work is based on work done previously by Leonard Eberding, Luca Scimeca, Zhaoyu Zhang, and Robert Mitchell. [2, 9, 19, 6].

The significant contributions of this project are:

1. Research and installation of a new compass sensor for the AntBot control systems.

2. Refactoring a sizable amount of code to make AntBot more usable for this project, and make it more accessible for future students.

3. Addition of a calibration system to allow the user to auto-detect the position of the camera lens attachment (see Section 3.3).

4. Results gathered for the Central Complex model using a non-deterministic outbound route.
5. Construction of a video recording tool to allow processed image frames to be recorded from the robot for later viewing and offline analysis.
6. Construction of new, larger, more robust tussocks for use in experimentation (as realistic obstacles and visual landmarks), based on the recommendation in [6]
7. Results definitively showing the impracticality of a focus of expansion based system for collision avoidance and justifying continued use of matched filters.
8. Identification and correction of a bug in the visual preprocessing system to raise the framerate from 2fps to approximately 10fps under normal usage.
9. Construction of a neural collision avoidance model based on the matched filter system from [6, 12].
10. Implementation of a biologically plausible "base model" for insect navigation, the Extended Central Complex (ECX) model.
11. Collection of evidence to support the recommendation that the current AntBot platform be retired, and suggestions for a new platform based on lessons learned from the AntBot.

2 Background

This project builds directly upon [6]. We first provide a review of the relevant background topics from that paper, before developing the key ideas further for this project.

2.1 The Focus of Expansion for Collision Avoidance

Optical flow is the 2D approximation of image flow, the 3D motion observed in a scene over consecutive image frames. Optical flow is described by a field of vectors (a *flow field*) which show how tracked points move between frames. It is a large and diverse area of study. Consequently, we will not provide a complete background on the fundamental principles. We will give only a succinct background of the concepts necessary for this paper. A comprehensive introduction to the topic is given by *O'Donovan* in [7].

The main driving point in this paper is the integration of multiple navigational systems into a single model, namely, the Central Complex. Optical flow filtering worked well for a standalone collision avoidance system [12, 6], however it does not fit so intuitively into the CX model. We therefore investigate an alternate approach. The relevant background revisits the concept of the *focus of expansion* (FOE) from [6, 7]. The FOE is defined as the point from which all optical flow vectors originate. The location of the FOE, and its relationship with the rest of the flow field, can give us information about a 3D scene from a 2D image (e.g. image depth)[7, 11].

In [6], the FOE was used only to compute time-to-contact with an obstacle. In this work, we instead attempt to use it directly to determine the potential location of an obstacle. Following [6] we will be using a dense optic flow field (tracking motion for every pixel in the image) as the computation of sparse fields was shown to be unreliable on the AntBot. A common, reliable and intuitive method of computing the FOE is the Least Squared-Error (LSE) method [14, 7, 15]. This method also appears to be in use in [11], a major inspiration for the optical flow methods presented in [6].

2.1.1 The Least Squared-Error method

This is the method given by *O'Donovan* and discussed in [6]. The FOE is computed simply as:

$$FOE = (A^T A)^{-1} A^T \mathbf{b} \tag{1}$$

$$A = \begin{bmatrix} a_{00} & a_{01} \\ \dots & \dots \\ a_{n0} & a_{n1} \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} b_0 \\ \dots \\ b_n \end{bmatrix}$$

Where, each pixel $p_i = (x, y)$ has associated flow vector $\mathbf{v} = (u, v)$. Finally, set $a_{i0} = u$, $a_{i1} = v$ and $bi = xv - yu$. Note that this computation and explanation have been adapted from [6].

This method for estimating the focus of expansion was originally given by *Tistarelli et al.* in their paper *Dynamic Stereo in Visual Navigation*[14, 7], and it serves as an excellent example for the reason the FOE is so difficult to compute in practice. In theory, we should be able to take any two vectors $\mathbf{u}$, $\mathbf{v}$ from the flow field, and compute the point at which lines running along them intersect. This point of intersection would give us the FOE [7]. While wonderfully simple in theory, this method only works for a perfect flow field. In reality, flow fields are imperfect; for example, visual noise can cause disruptions to areas of the field. Therefore picking two arbitrary vectors could lead to an erroneous FOE. Unravelling the matrix notation of Equation 1, shows this to be a least squares technique; essentially, we compute the best FOE for the flow field given.

2.2 Matched Filters for Collision Avoidance

In light of the results we will present later with regards to the FOE, we will also revisit the matched filter collision avoidance system from [6], though the flow field computation, filter generation and comparision are unchanged.

We compare the observed optical flow field to a precomputed optical flow *filter*. This filter represents the motion we expect to observe in the image frame given some self (camera) motion. This method for collision avoidance has been an active area of research in the biorobotics (and wider robotics) community for some time though there are different hypotheses about how the information is actually used by insects [10].

In [6] (and this work), the filter in use is one implemented by *Scimeca* for the purposes of speed retrieval. This filter represents the flow we would expect to observe when the agent experiences forward translational motion, in a clear environment. A graphical representation of the filter can be seen in Figure 1.

The mathematical breakdown of the filter computation and application process are given by [6] as follows. The filter is given as an $N \times 3$ matrix F such that:

$$F_i = [\quad sin(-\pi + i\frac{2\pi}{N}) \qquad 0 \qquad 0 \quad] \quad for \quad i \in \{0, N-1\} \tag{2}$$

where F_i is the ith row of F and each row corresponds to a pixel value in the x-axis of the 360° image (see Section 3). Let M be the matrix returned by the OpenCV *calcOpticalFlowFarneback()* function when it is given two consecutive image frames. M represents the dense flow field computed by the function and is constructed such that $M_{y,x}$ is a one-dimensional array with two elements, where $M_{y,x}[0]$ is the pixel displacement in the x-axis, and $M_{y,x}[1]$ is the pixel displacement in the y-axis. The filter is then applied by performing the following computation; for each x, y:

Compute pixel's starting location in the shifted frames as:

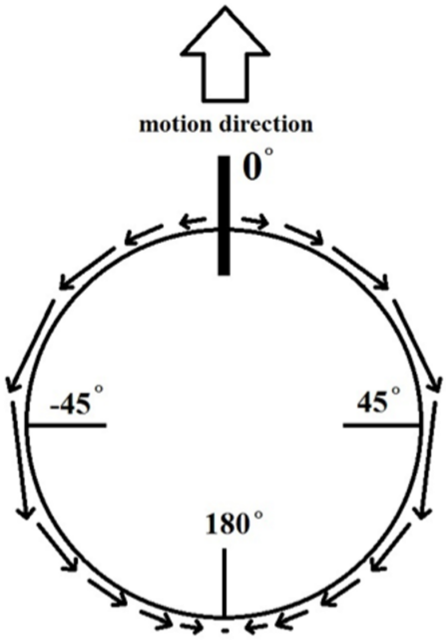
$$PLF = [\quad (x+4) \pmod N \qquad y \quad]$$
$$PRF = [\quad (x-4) \pmod N \qquad y \quad]$$

Compute the flow vector from the start location to where it is now as:

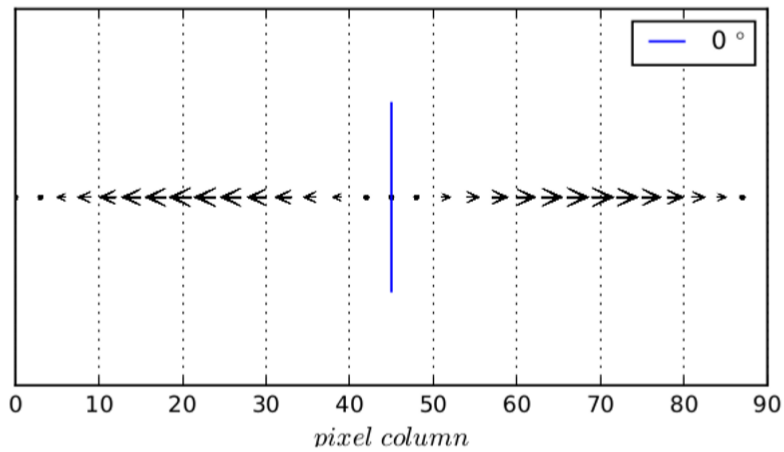
$$CLF = [(M_{y,x}[0] + x + 4) \pmod N \qquad (M_{y,x}[1] + y) \pmod R]$$
$$CRF = [(M_{y,x}[0] + x - 4) \pmod N \qquad (M_{y,x}[1] + y) \pmod R]$$

Extract the correct filter vector from matrix F (Eq. 2):

$$LFilter = F_{PLF[0]}$$
$$RFilter = F_{PRF[0]}$$



(a) Motion pattern



(b) Implemented filter(s)' response

Figure 1: The optical flow filter given by *Scimeca* for speed retrieval, caption by *Scimeca*, Figure 13 (clarifications in square brackets given by *Mitchell* in [6]): Figure (a) shows the idea behind the modication of the lter for speed retrieval. When moving forward, for example, we expect the rectangular 360 [degree] image to have vector ows in dierent length throughout the frame, in the pattern shown in the Figure. Figure (b) shows the lter response of the modied ltering process for this [Scimecas] project. During matching, the found ow vectors will be projected onto the corresponding lter vectors nullifying the information on the y axis and scaling the vectors dierently depending on the area of the image they are found in.

Apply the filter:

$$LFD = LFilter \cdot CLF \tag{3}$$
$$RFD = RFilter \cdot CRF \tag{4}$$

where $N = 90$ is the number of horizontal pixels and $R = 10$ is the number of vertical pixels. LFD and RFD give a numerical measure of how far the observed flow field deviates from the expected flow field on the left and right sides respectively. These measures are then summed over the entire shifted image frame for each side which gives us a measure of overall deviation from the filter:

$$LeftFlowSum = \sum_{i=0}^{K-1} LFD_i \qquad RightFlowSum = \sum_{i=0}^{K-1} RFD_i \tag{5}$$

where $K = N \cdot R$ and LFD_i, RFD_i are the left and right differences for the ith pixel. Finally, the difference between the two flow sums is computed:

$$FlowDifference = LeftFlowSum - RightFlowSum \tag{6}$$

The flow difference tells us if one side is experiencing a dramatic increase in flow disturbance (e.g. if an object is looming, Figure 12). Therefore, this difference can be used to dictate whether an avoidance maneuver is required and in which direction the agent should turn.

A similar method was also used by *Stewart et al.* in their simulations of Drosophila [12]; interestingly they use a matched filter to detect a collision, then use the position of the FOE to determine a steering direction [12]. An example of the flow disturbance caused by a looming object can be seen in Figure 12.

2.3 The Mushroom Body for Visual Navigation

The Mushroom Body model is an artificial neural network which models the mushroom body structures present in the insect brain [1]. It consists of three layers: Projection Neurons (PNs), Kenyon Cells (KCs) and Mushroom Body Output Neurons (MBONs). These MBONs are also referred to as Extrinsic Neurons (ENs) by older works, we use MBON herein. The original MB model proposed by *Ardin et al.* for navigation in [1] contained 360 visual PNs (vPNs), 20,000 KCs, and a single MBON. Every KC connects to the MBON, and each KC also connects to 10 vPNs chosen uniform randomly. The KC-MBON connections all start with weight $w = 1$. The figure and caption from *Ardin et al.* are given here in Figure 2. In their implementation and previous AntBot implementations, captured images were converted to greyscale and downsampled [1, 2, 19, 6]. As such, each pixel can be described by its brightness (greyscale value) and each KC has a brightness threshold.

Learning occurs by showing patterns (images) to the vPNs. Each KC sums the brightness of all connected vPNs and compares that sum to the activation threshold. If the total brightness is greater than the threshold then the KC is *activated*. As the connections between the vPN and KC layers are random, a given image will generate some sparse pattern of KC activation; to learn this pattern, the weight of the

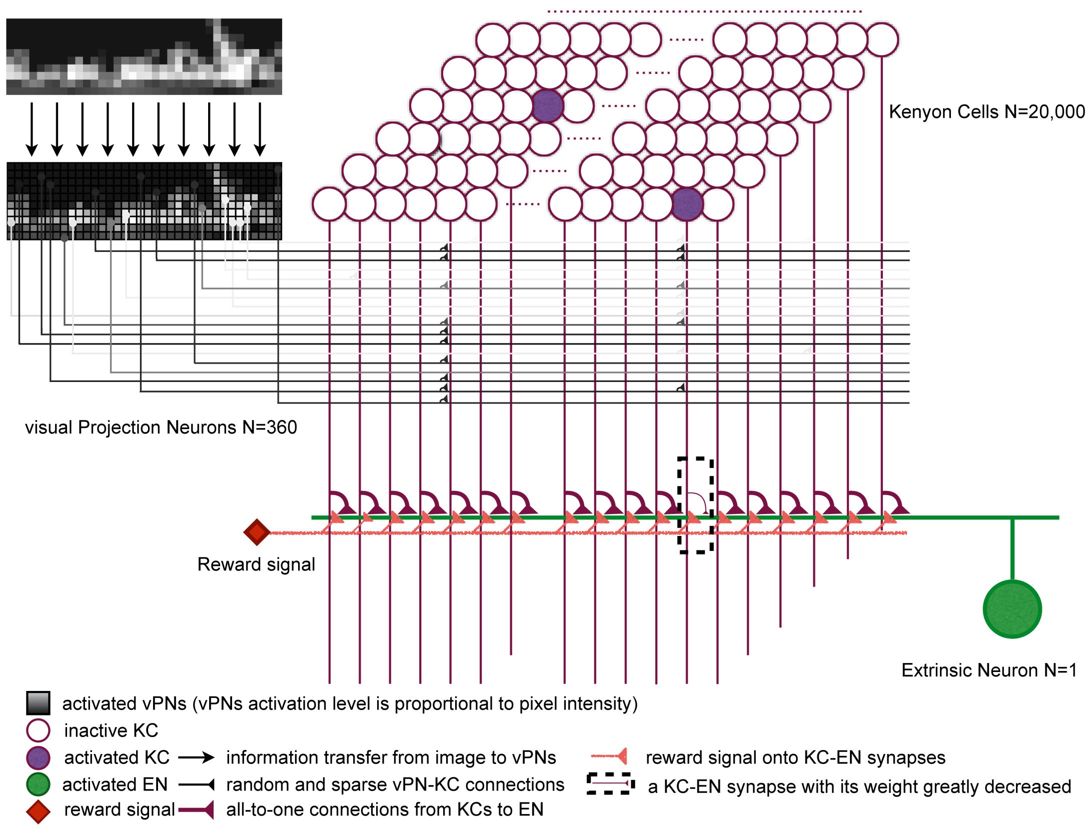


Figure 2: The Mushroom Body circuit: (Caption from *Ardin et al.*, Figure 2; note, their description and figure uses "EN" instead of "MBON"): Images (see Fig 1) activate the visual projection neurons (vPNs). Each Kenyon cell (KC) receives input from 10 (random) vPNs and exceeds firing threshold only for coincident activation from several vPNs, thus images are encoded as a sparse pattern of KC activation. All KCs converge on a single extrinsic neuron (EN) and if activation coincides with a reward signal, the connection strength is decreased. After training the EN output to previously rewarded (familiar) images is few or no spikes.

KC-MBON connection is lowered from 1 to 0 for the active KCs.

To determine image familiarity during the recapitulation process, a pattern is projected onto the vPNs (again giving a sparse pattern of KC activation). The MBON then sums the synaptic (KC-MBON) weights of all active KCs to obtain a *familiarity* measure; recall, these weights are decayed as part of the learning process; thus, the lower the MBON output, the more familiar the image (and vice versa). Different route following strategies have been implemented: scanning [1, 2, 19], Klinokinesis [19], combination with the CX model [19](see below), and visual scanning [6]). Most commonly, some form of scanning is used whereby the agent scans an arc for the most familiar direction.

This explanation has been a brief and is provided solely for context as we do not work too closely with the MB though it is part of the proposed model. Part 1 has more focus on visual navigation and gives slightly deeper insight [6] (particularly in relation to AntBot projects), and of course the work of *Ardin et al.* can give the full details [1].

The MB model described above has been tested on the AntBot in three different works (albeit using different methods and metrics); the network as presented in [2, 19, 6] differs only in the number of PNs present (900 vPNs are used in all three works, as opposed to the 360 present in [1]), and the KC modelling (*Ardin et al.* use a spiking model[1] for the KCs [1]). More relevant to this work is the modification given by *Zhang*, whereby, 8 MBONs are present as opposed to 1 (see Section 2.5).

2.4 The Central Complex for Path Integration

The Central Complex (CX) is a highly conserved structure present in the insect brain [8, 13]. Though the finer structural details and component position may vary, the basic composition is more or less the same across species [8]. Organization and function of the various parts of the CX are set out by *Pfeiffer and Homberg* in [8].

We direct our attention to the CX model presented by *Stone et al.* which is the first neural model for path integration in the insect brain with its structure drawn purely from insect physiology. Interestingly, this model is a more advanced version of an earlier model presented by *Haferlach et al.*, which was *evolved* using a Genetic Algorithm [5]. We present this also, as it provides some historical context for the development of a biologically plausible neural circuit for path integration.

2.4.1 The Evolved Model for Path Integration

Prior to the CX model described below, there were several candidate neural networks proposed for PI in ants [5]. *Haferlach et al.* report the two best-known candidates being hand-designed, with more recent research being directed towards an evolutionary approach to network design [5]. The approach presented in [5] follows this trend and employs a Genetic Algorithm (GA) (see Appendix A for a brief overview of GAs) to evolve a neural model which is suitable for PI with biologically plausible inputs.

[1]The AntBot projects do not explicitly model membrane potentials and realistic neural responses. They use a simple abstraction for the sake of complexity. The network function is the same.

Haferlach et al. encode solutions as lists of integers. In these lists there are *start markers* and *end markers* which delimit the definitions of the actual neurons or sensors present in the model. The complete model (topography, connection weights, sensors, and effectors) is limited to a fixed-size encoding (genome limited to 500 parameters; 50 neurons maximum) [5]. An example encoding from [5] can be seen in Figure 3.

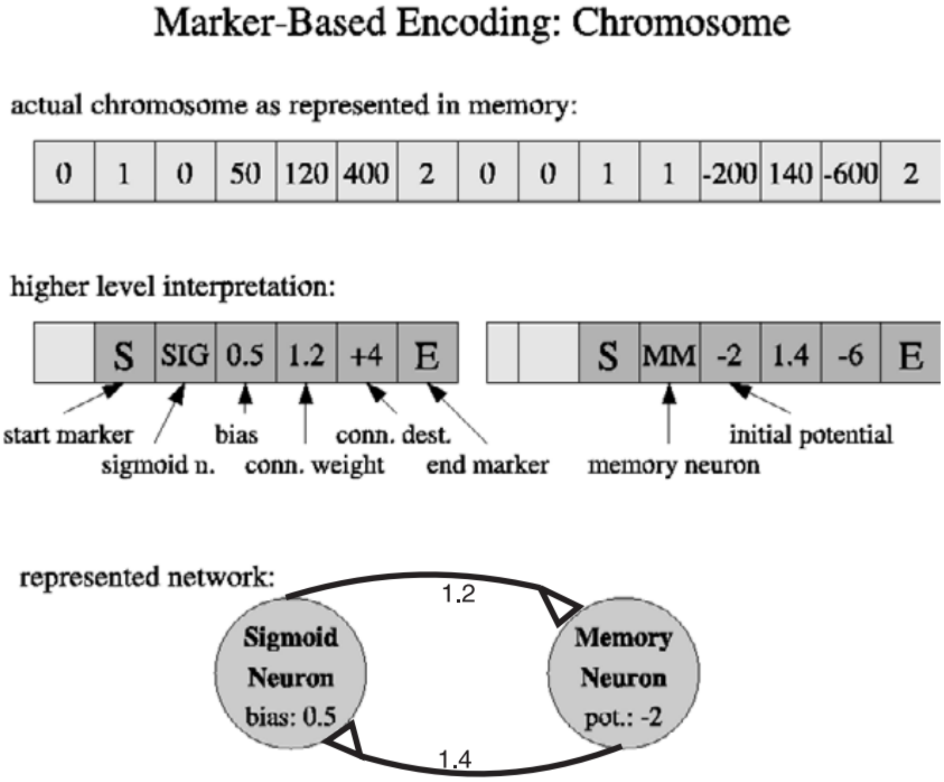


Figure 3: (Figure 2 from [5]) The Marker-Based encoding scheme used by *Haferlach et al.*, demonstrating how a simple neural model would be encoded as a sequence of integers.

Localised *tournament selection* is used to pick solutions for transition into the next generation. Each solution is assigned a location in 1-dimensional space; a solution is picked randomly, then another solution within distance k of the first is picked for the tournament (k usually between 5 and 15) [5]. The solution with higher fitness wins the tournament and progresses to mutation and *crossover*[2]. This localised selection allows the algorithm to find multiple *good* solutions which need not share the same genetic information.

The fitness of each solution is evaluated by having the agent navigate an outbound path through two randomly chosen points, then testing its ability to navigate back to the start point. During the outbound phase, information is input into the path integration network via the sensors; the network output is ignored. During the inbound route, the agent is steered by the network output. An individual solution is evaluated until some maximum time limit m is reached, and the distance to the start point $dist_t$ is computed at each time step. The fitness is then the inverse sum of squared distances [5]:

[2]Crossover - A particular method for breeding two solutions whereby their genomes are cut at a particular point and the ends swapped.

$$f = \frac{1}{\sqrt{\int_0^m dist_t^2 dt}} \tag{7}$$

The fitness function is then augmented to penalise homeward routes which spiral (circling the start point, getting closer with each pass), and to reward networks which are simple in structure; shown in Equations 8 and 9 respectively.

$$f = \frac{1}{\sqrt{\int_0^m dist_t^2 |\theta - \omega| dt}} \tag{8}$$

$$f = \frac{1}{(1+k_n)^{c_n}(1+k_c)^{c_c}\sqrt{\int_0^m dist_t^2 |\theta - \omega| dt}} \tag{9}$$

where ω is the nest heading, θ is the agent's heading, k_n is a neuron penalty constant, k_c is a connection penalty constant, c_n is the number of neurons, and c_c is the number of connections (i.e. the network is heavily penalised for the number of connections and neurons it has, forcing simpler networks) [5]. These are applied as a two-stage evolutionary process; evolve a solution first using Equation 8 to compute the fitness, then use an optimum solution from the previous stage to seed a new population and evolve a solution from here using Equation 9. The two-stage evolutionary process alone did not find acceptable solutions. The search space was constrained by providing the following four constraints on network topology:

- each direction cell had to be connected to at least one memory neuron;
- each direction cell had to be connected to at least one sigmoid neuron;
- sigmoid neurons had to be connected to turn effectors;
- a maximum of two turn effectors (left and right) were allowed [5].

Haferlach et al. present the model shown in Figure 4; notice the evolved structure bears a striking resemblance to the CX model presented by *Stone et al.* (Figure 5). Indeed, this network operates on the same principle; encoding a heading vector across multiple neurons with directional preferences. This network provides a compact, elegant, robust model capable of performing path integration with reasonable errors. The network also demonstrated some capability for PI with obstacles present (though errors were generally much greater) [5].

2.4.2 The Central Complex Model

The Central Complex model is a six layer artificial neural network presented by *Stone et al.* which has been shown to provide a plausible neural substrate for Path Integration (PI) both in simulation and on the AntBot platform [9, 13]. The model presented is shown in Figure 5. Splitting the model into its six layers, we can observe a breakdown of functionality:

- Layer 1: Heading preprocessing (TL), Speed (TN)
- Layer 2: Heading preprocessing (CL1)

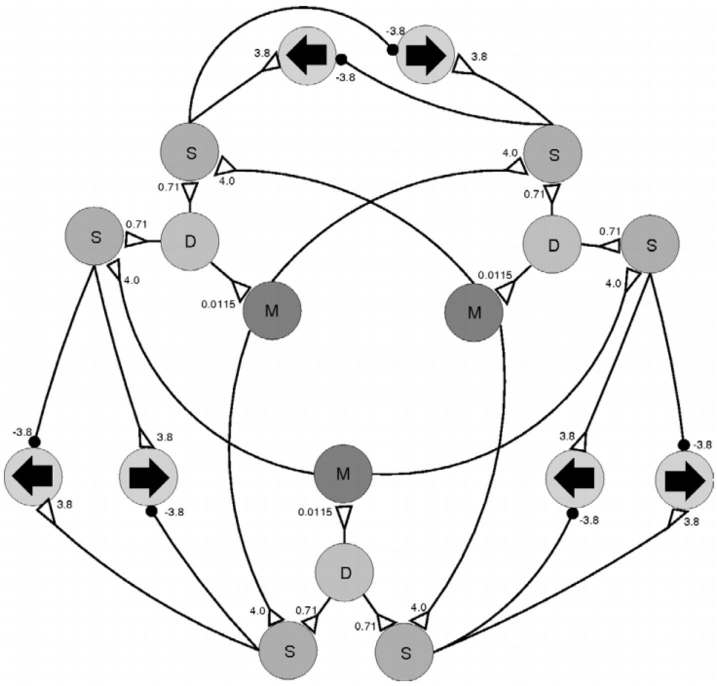


Figure 4: (Figure 3 from [5]) The high-fitness network, consistently evolved using the constrained two-stage evolution process illustrated by *Haferlach et al.*

- Layer 3: Heading (TB1)
- Layer 4: Memory (CPU4)
- Layer 5: Normalisation and Inhibition (Pontine Neurons)
- Layer 6: Steering/output (CPU1)

Figure 5 shows four types of neuron: TN (Tangential Neuron, brown), TB1 (green), CPU4 (yellow and orange), and CPU1 (dark blue, light blue, and purple).

While Figure 5 (Left) distinguishes between CPU1a (blue) and CPU1b (purple) neurons, we will ignore this distinction; for clarity, the physiological mapping between these CPU1 subtypes in the Upper Central Body (CBU) and the Protocerebral Bridge (PB) is different, but the function they serve in the Central Complex is the same [13]; thus, the distinction makes no difference in the model. Similarly the normalisation and inhibition functions of the Pontine Neurons only has an effect when the agent experiences holonomic motion (motion where the view direction does not match the direction of travel); as AntBot is incapable of such motion, we can safely ignore the function of the Pontine Neurons; in our case, the Pontine Neurons would have the same activity patterns as the CPU4 neurons [13]. Figure 5 (Right) shows how the Pontine inhibition is structured.

We now break down the different neuronal types and their proposed functions. Citation is given, but for the avoidance of any doubt, the following descriptions are adapted from *Stone et al.* (see STAR Methods) [13]. We also add implementation information for the AntBot where appropriate; occassionally the AntBot implementations are slightly different.

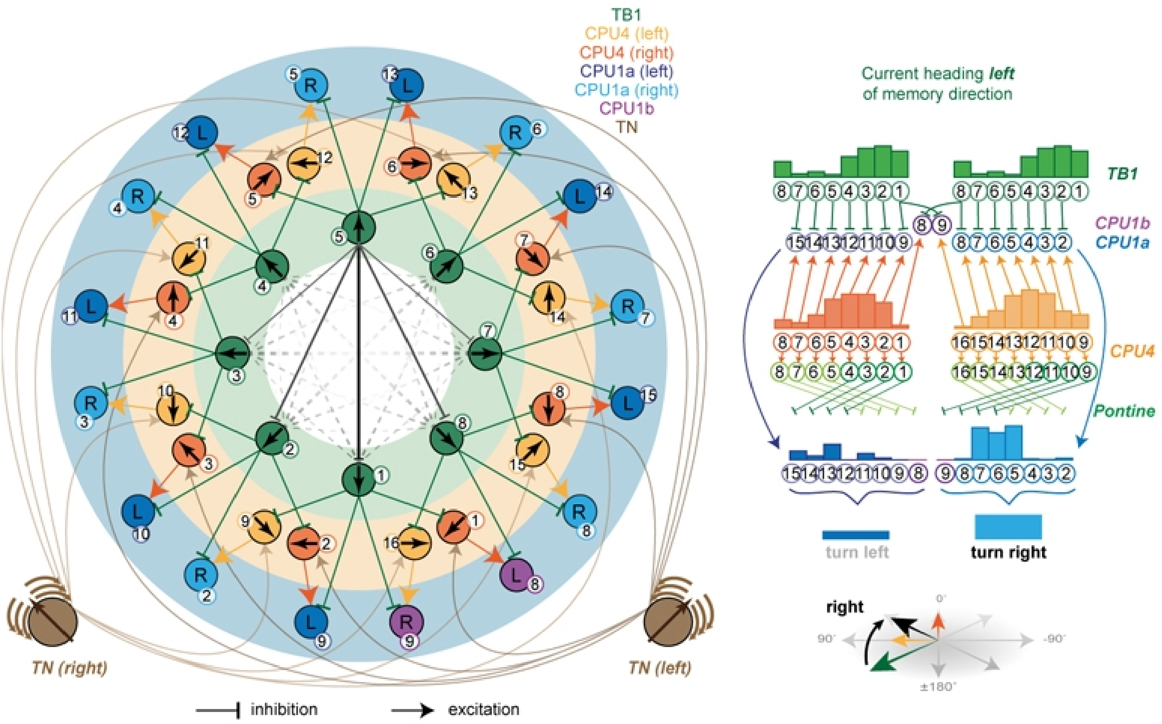


Figure 5: The Central Complex model presented by *Stone et al.* (Left) This graph demonstrates the basic structure of the CX model (Figure 5G from [13]). Pontine neurons have been excluded for clarity. (Right) This graph shows how signals propogate through the network where the current heading lies to the left of the desired heading, i.e. a right turn should be generated (Figure 5I from [13]). The numbers given at each layer on the right correspond to the numbers given for each neuron in the graph on the left.

Simulated Neurons: Each neuron is described by its firing rate, where the firing rate r is a sigmoid function of the input I:

$$r = \frac{1}{1 + e^{-(aI-b)}} \tag{10}$$

where a and b are parameters which control the slope and offset of the sigmoid function [13]. Optionally, Gaussian noise may be added to the output (no noise is applied in AntBot implementations). The input to each neuron is the weighted sum of the activity of each neuron that synapses onto it; say neuron j, the input is:

$$I_j = \sum_i w_{ij} \cdot r_i \tag{11}$$

The weights used by *Stone et al.* can only be 0, 1, or -1 for no-connection, excitatory, or inhibitory respectively [13].

TL & CL1: The TL neurons are the input point for heading information. In the ant brain this heading information comes from a *sky-compass*; the ant eye contains cells sensitive to polarized light which allows the ant to infer an accurate allothetic heading from vision alone. Interestingly, there is also evidence that ants have the capability to infer a direction without a view of celestial cues, suggesting they may have access to some other signal, the candidate signal being the geomagnetic field [3, 4]. The TL

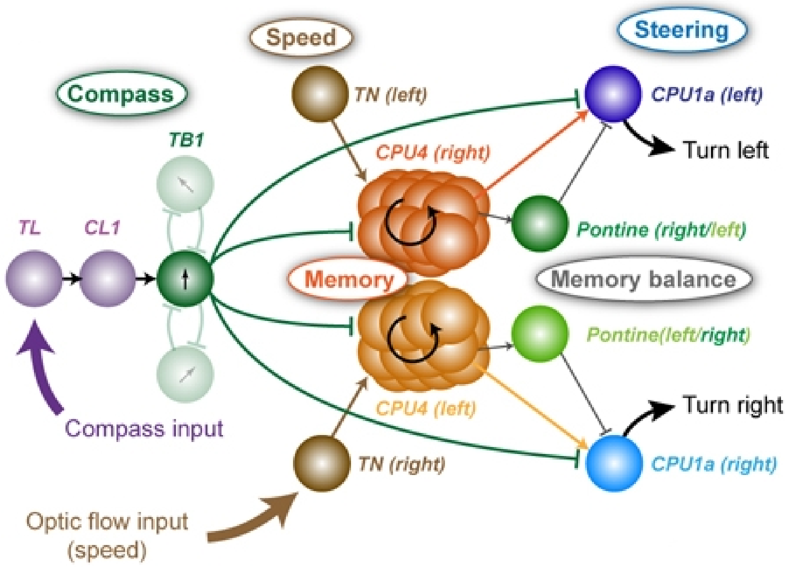


Figure 6: Here we can see the layers of the CX model and how they fit together. A heading signal is input to the TL neurons, propogating through the CL layer to TB1 (heading ring-attractor) and CPU4 (memory). TN neurons (speed sensitivity) input directly to CPU4. So, the combination of heading and speed inputs to CPU4 gives a measure of distance travelled in a particular direction; this facilitates generation of a steering command in CPU1 providing a mechanism for path integration.

neurons have been shown to be polarisation sensitive across multiple insect species [13] (see STAR Methods). Each TL neuron has a preferred direction (i.e. a specific direction of poliarisation sensitivity) θ_{TL}, and there are 16 such neurons representing the 8 directions around the agent (i.e. $\theta_{TL} \in \{0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°\}$) [13]. Together, the 16 TL neurons encode the heading of the agent in a single timestep; each neuron receives input activation as:

$$I_{TL} = \cos(\theta_{TL} - \theta_h) \tag{12}$$

where θ_{TL} is the preferred heading of the neuron as above, and θ_h is the current heading of the agent. In the next heading layer, there are 16 CL1 neurons which use inhibition to invert the polarisation response [13]. *Stone et al.* comment that these neurons effectively make no difference to the model and are included for completeness. They are also included in previous AntBot projects which make use of the CX model [19, 9]. On AntBot, the heading is derived from the onboard compass on the mobile phone (see Section 3).

TN: There are 4 TN neurons which act as an input for speed information. The TN neurons are sensitive to optical flow and can be split into two subtypes: TN1, and TN2. *Stone et al.* showed that TN1 neurons are inhibited by simulated forward flight and excited by simulated backward flight, while TN2 neurons are inhibited by simulated backward flight and excited by simulated forward flight [13]. Each of the four TN neurons has a tuning preference; these tuning preferences were measured in bees as approximately $+45°/-45°$ for TN2, and $+135°/-135°$ for TN1 (where 0° is straight ahead). In summary, we have TN1_{left}, TN1_{right}, TN2_{left}, and TN2_{right}. It is thought that these neurons provide a (or part of a) mechanism for odometry by allowing the model to integrate speed with respect to time giving a distance measure [13]. *Stone et al.* give the speed calculation from the TN neurons as:

$$I_{TN_L} = [\sin(\theta_h + \phi_{TN}) \quad \cos(\theta_h + \phi_{TN})]\mathbf{v} \tag{13}$$

$$I_{TN_R} = [\sin(\theta_h - \phi_{TN}) \quad \cos(\theta_h - \phi_{TN})]\mathbf{v} \tag{14}$$

where $\mathbf{v}$ is the velocity of the agent in Cartesian coordinates, $\theta_h \in [0°, 360°)$ is the current heading of the agent, and ϕ_{TN} is the preferred angle of that TN neuron [13].

The TN neurons are not modelled in the *basic* CX model on AntBot. There is, however, a *holonomic* implementation which models both the TN1 and TN2 neurons present on the robot. The formulae above are implemented on the robot but they are never used. Instead, the raw left and right speeds computed from the optical flow are passed in (see *Scimeca* [9] for the details of speed computation from optic flow). The TN2 neurons will simply clip these speeds to make sure they lie in $[0, 1]$; the TN1 neurons will perform $(1 - s)/2$ for both the left and right speeds s, then clip the output to lie in $[0, 1]$. The output is returned directly, rather than being returned as a sigmoid (as for all other neuron types).

TB1: There are eight TB1 neurons, each with a directional preference θ_{TB1}, which correspond to the eight cardinal directions in the model. Each TB1 neuron receives excitatory input from the pair of CL1 neurons that have the same directional preference. The TB1 layer contains inhibitory connections between peer neurons where each TB1 neuron strongly inhibits other TB1 neurons with opposite directional preferences (see Figure 5) forming a *ring attractor*[13]. The weighting for an arbitrary inhibitory connection from neuron i to neuron j is given by:

$$w_{ij} = \frac{\cos(\theta_{TB1,i} - \theta_{TB1,j}) - 1}{2} \tag{15}$$

where $\theta_{TB1,i}$ is the direction preference of TB1 neuron i (similarly for $\theta_{TB1,j}$). The total input for each TB1 neuron from the CL1 layer at timestep t is:

$$I_{TB1_j^{(t)}} = (1 - c) \cdot r_{CL1_j}^{(t)} + c \cdot \sum_{i=1}^{8} w_{ij} \cdot r_{CL1_j}^{(t-1)} \tag{16}$$

where $c = 0.33$ is a scaling factor which determines the relative strength of excitation from the CL1 layer and inhibition from other TB1 neurons; the AntBot implementation of the TB1 neurons is the same. This network layer produces a stable heading encoding which provides accurate input for the CPU4 layer, underpinning accurate path integration.

CPU4: The 16 CPU4 neurons receive input in the form of an accumulation of heading $\theta_h^{(t)}$ of the agent, along with a modulated speed response from the TN2 neurons [13]. The CPU4 neurons accumulate distance with direction. The input the CPU4 neurons is given by:

$$I_{CPU4}^{(t)} = I_{CPU4}^{(t-1)} + h \cdot (r_{TN2}^{(t)} - r_{TB1}^{(t)} - k) \tag{17}$$

where $h = 0.0025$ determines the rate of memory accumulation, and $k = 0.1$ is a uniform rate of memory decay [13]. All memory cells are initialised to $I_{CPU4}^{(0)} = 0.5$ and are clipped at each timestep to fall between 0 and 1 [13]. As shown in Figure 5, each TB1 provides input to two CPU4 neurons, each of which receives input from the TN2 cell in the opposite hemisphere. As these neurons accumulate distance with respect to a direction, they provide a population encoding of the *home vector* (the integrated path back to the nest) [13]. Interestingly, *Zhang* showed that the network can be initialised to an arbitrary state, allowing the agent to navigate along arbitrary vectors [19]. While this seems intuitive, the experimental evidence is valuable and demonstrates that the CPU4 layer could form a basis for the *vector memory* discussed by *Webb* in [16] (see Section 2.5).

The basic CX model on the AntBot does not model TN neurons, so the input is computed differently; furthermore, the gain and loss factors are different. The input can be expressed as:

$$I_{CPU4}^{(t)} = I_{CPU4}^{(t-1)} + s \cdot ((1 - r_{TB1}^{(t)}) \cdot g - l) \tag{18}$$

where $l = 0.0026$ is the uniform rate of memory loss, $g = 0.005$ is the uniform rate of memory gain, and s is the current speed. This value is then clipped to lie in $[0, 1]$. As an aside, in the holonomic model mentioned previously, the input is computed as in Equation 17 as the TN neurons are present in the model.

Pontine: The pontine neurons project contralaterally connecting opposite CBU columns (shown in Figure 5 (Right)). The 16 pontine neurons each receive input from one CPU4 column [13]; pontine input can be given simply as:

$$I_{Pontine}^{(t)} = r_{CPU4}^{(t)} \tag{19}$$

The pontine neurons are not implemented on AntBot for simplicity. AntBot is incapable of holonomic motion, so they would have no functional effect.

CPU1: There are 16 CPU1 neurons present in the model. Each of which receives inhibitory input (weight $= -1$) from a TB1 neuron; where, each TB1 neuron provides inhibitory input to two CPU1 neurons (in the same pattern as the TB1-CPU4 connections - see Figure 5). Each CPU4 neuron also provides input to a CPU1 neuron (so we get input from vector memory and current heading). The CPU1 input can be expressed as:

$$I_{CPU1}^{(t)} = r_{CPU4}^{(t)} - r_{Pontine}^{(t)} - r_{TB1}^{(t)} \tag{20}$$

As can be seen the CPU1 neurons also receive input from the pontine neurons. The CPU1 neurons form two sets, those connecting to left motor units and those connecting to the right. On AntBot the pontine term is omitted (i.e. the input becomes $I_{CPU1}^{(t)} = r_{CPU4}^{(t)} - r_{TB1}^{(t)}$).

We choose a direction simply by summing the CPU1 outputs on both sides and the difference indicates the direction and strength of the required heading correction:

$$\theta_h^{(t)} = \theta_h^{(t-1)} + m \cdot (\sum_{i=1}^{8} r_{CPU1R_i} - \sum_{i=1}^{8} r_{CPU1L_i}) \tag{21}$$

where $m = 0.5$ is a constant [13]. The angle computation is the same on the AntBot.

There are some details which have been omitted for clarity. The stack presented here should effectively describe how the network generates a turning signal from its current state and inputs to perform effective, accurate path integration. For further details, please consult [13] (the STAR Methods section contains the full technical details of the model).

2.5 The Eight MBON Model (CXMB)

In [19], the MB network is modified by adding 7 MBONs. Each MBON has its own KC-MBON connection array, each with their own unique weights. Each connection array corresponds to one of the eight cardinal directions represented by the TB1 layer of the CX model (namely, 0°, 45°, 90°, 135°, 180° ,225°, 270°, 315°) (see Section 2.4).

The additional MBONs give image memory an associated direction, so training is performed with respect to orientation. For example, if the agent has a heading of 45° when a image is stored, then only the corresponding connection array has its weights updated during learning. Practically, this is done by querying the TB1 layer of the CX model to find the current direction according to the model (rather than directly querying the robot's onboard compass directly). This process is visualised in Figure 7.

While, in theory, this eight MBON model could function independently, *Zhang* uses it to (rather neatly) augment the CX model (creating the combined CXMB model); this allows navigation to be performed using a combination of visual memory and path integration information. The navigation process can be described by a sequence of equations. We modify the notation slightly for clarity, but the equations are the same as presented in [19].

During recall, the MB circuit is shown an image in the usual way (see Section 2.3); however, we now get eight responses, giving us a response distribution. This distribution can be interpreted as giving us the most likely direction of travel, when the image presented was first observed. This distribution requires some modification to integrate it into the CX response. Let M_i be the familiarity response of the ith MBON; the responses are normalised as:

$$\bar{M_i} = \frac{M_i}{\Sigma_{k=0}^{8} M_k} \tag{22}$$

which gives us the normalised response $\bar{M_i}$ between 0 and 1. $\bar{M_i}$ must be inverted so that the most likely direction has the greatest response (in the MB model, the most familiar direction would give the lowest response):

$$\bar{M_i}^{-1} = 1 - \bar{M_i} \tag{23}$$

This visual response is then combined with the memory response from the CPU4 layer of the CX model to give output at the CPU1 layer:

$$CPU1_{output} = k \cdot W_{CPU4} \cdot CPU4 + (1 - k) \cdot W_{MBON} \cdot \bar{M}^{-1} \tag{24}$$

where k is a weighting factor that determines the relative stregths of the CX response and the MB response in the output ($k = 0.8$ in [19]), $\bar{M}^{-1}$ is the collection of all inverse

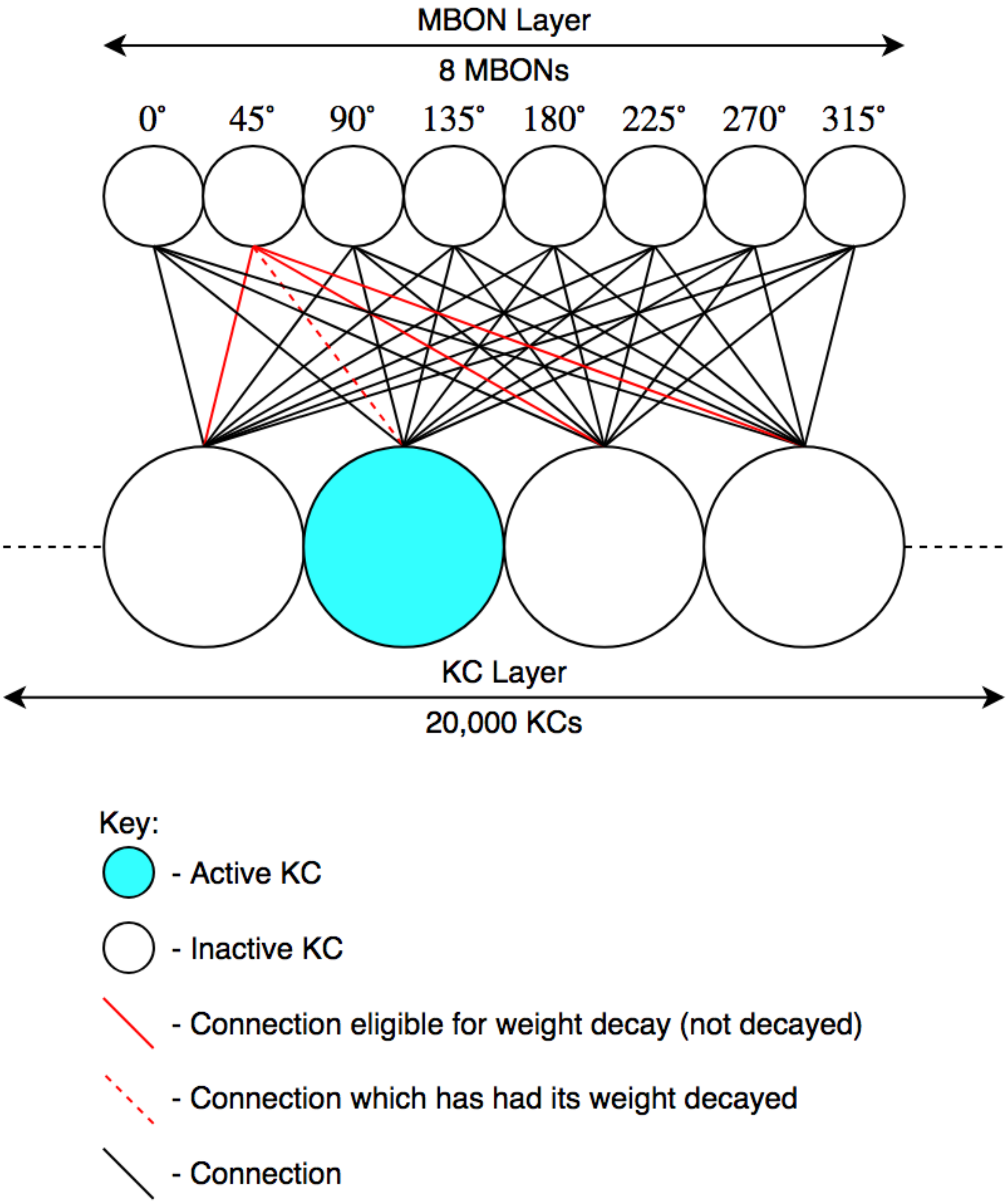


Figure 7: Our interpretation of the eight MBON model proposed by *Zhang*. Every KC connects to every MBON. All connection weights start out at $w = 1$. Following the example presented in the text, if an image being learned corresponds to facing a direction of 45°, then only the conncetions to that MBON (highlighted in red) are eligible to have their weights modified. Recall, however, that these weights will only be modified if the KC was activated (not shown in the figure).

normalised MBON responses, W_{CPU4} is a custom matrix[3], W_{MBON} is an identity matrix that will expand the MBON response array from 8x1 to 16x1 [19].

In order to test the CXMB model, *Zhang* first demonstrated the functionality of a *copied memory model*. The test of the copied memory model aimed to prove that the CPU4 state of the agent could be copied and stored, the CPU4 state modified, and then

[3]This is the term used by *Zhang* to describe the W_{CPU4} matrix. More specifically, this matrix describes the connections between the CPU4 and CPU1 layers of the CX model.

restored from the copy to allow the agent to navigate home. The copied memory model was tested by sending the agent on a pre-determined oubound route to a feeder (chosen at random, but consistent between trials), copying the CPU4 state, and allowing the robot to navigate home using the CX model; the CPU4 state will be modified by this homeward navigation. The agent is then replaced at the feeder, its CPU4 state restored from the copy, and tasked with navigating home a second time. The copied memory model was tested as a pre-requisite for testing the CXMB model, however, it demonstrates an important capability of the CX model; namely, the capability to directly load a state into its memory in order to navigate; a concept referred to as *vector memory* in [16]. Indeed, this concept of vector memory is shown to be quite a useful tool in insect navigation [16]. At time of writing, the precise biological mechanism employed by insects to perform this task is unknown.

The CXMB model is then tested in the same way. The agent follows its oubound route to the feeder, navigates back once using the CX model (the MB model is trained on this first inbound trip), is replaced at the feeder, and finally, tasked with navigating back again, this time using the CXMB model. To be clear, the CPU4 state of the CX model is stored after the outbound route, and loaded back into the network before the second inbound trip. *Zhang* reports that the second inbound trip (using both CXMB) showed more heading adjustments during its traversal and, on average, performed slightly better than a pure CX implementation[19]. It should also be noted that the average performance of both models in *Zhang's* work was good [19].

We note two methodological problems with *Zhang's* work: The outbound route, while randomly chosen, was the same across all trials, and, for the copied memory and CXMB experiments, the agent was placed facing the nest for the test run (second inbound). To add to the first problem, the outbound route ended such that the robot was facing back towards the nest. Both of these flaws could lead to apparent path integration behaviour where, in fact, the agent could have made it sufficiently close to the nest by simply following a straight line. While the agent was certainly employing the CX to perform path integration, we do not feel that the tests chosen provide strong evidence of functionality.

2.6 Review of Part 1

The work in [6] specifically covers the Mushroom Body and Optical Flow Collision Avoidance. The Mushroom Body model used in [6] is the same as that used by *Ardin et al.* in [1], except for the small differences noted in Section 2.3. The model tested in [6] used a single MBON with a scan-based route following strategy. The scanning differed from previous works as it was implemented as an image manipulation algorithm (termed *visual scanning*) instead of a physical turn performed by the AntBot.

Multiple OFCA systems were tested, however in final experiments an optical flow filtering system is used (see Section 2.2). This strategy proved simple, but effective. While we initially move away from it for this project, it was used for initial tests of the CX model, as it provided a simple out-of-the-box method to generate a non-deterministic path for the model to integrate.

Both systems functioned well and provided a solid baseline from which we will work in this project. The MB model proved very capable at following routes learned in a non-deterministic fashion through a cluttered environment.

3 Platform

The AntBot is an autonomous robot built to allow easy testing of the various navigational algorithms in the *Ant Navigational Toolkit* on a physical agent. AntBot is assembled using off-the-shelf parts with an Android phone at its heart. We give a high level description here; additional detail can be found in [6].

3.1 Hardware

The robot is composed of three main parts: a Dangu Rover 5 chassis, an Arduino microcontroller, and a Google Nexus 5 Android smartphone. The smartphone provides a (theoretically) solid base for an autonomous agent, with sufficient processing power and memory to run the models, a convenient touchscreen interface to display information and an adaptable control interface, a built-in camera, and extensive software libraries. As such, the smartphone works as the computational centre, performing all higher sensory processing and acting on this analysis. Once the algorithm running on the phone has decided on a course of action, it sends a command via a serial connection to the Arduino board; the board, then parses the command and translates it into a sequence of motor commands which are sent to the built-in motor board in the chassis.

Figure 8: The AntBot, connected to the charging station. This figure also shows the position of the mobile phone, camera attachment, and retroflective motion capture markers on the top of the robot.

Additionally, the robot possesses a 360° camera attachment, giving the robot a panoramic view to match that of the desert ant[4]. The robot was also modified as part of [6] to allow it to be charged with an external charger without dismantling it.

3.2 Software

The robot requires two levels of software; higher level Java (Android) code to perform the high level functions and lower level code written in C/C++ (Arduino dialect). The Arduino code can be thought of as the firmware.

[4]In reality, the desert ant has a visual field of only 280° (approx.)[1], however an approxmiation of 360° was considered appropriate when the robot was built.

Figure 9: The Kogeto Dot 360° panoramic lens attachment.

Figure 10: A sample view of the lens from the front facing camera, before any processing.

3.2.1 Android

Android provides a reasonable conceptual basis for working on robotics projects. The Operating System permits multiple applications to run asynchronously and pass information between them; for example, the user could develop an application to perform all visual processing then broadcast a processed image for other applications which require it. The original AntBot software was split into separate five applications: Ant Eye, Visual Navigation, Path Integration, Combiner, and Serial Communications, neatly compartmentalising each task. In recent years, this structure appears to have been abandoned and almost all code is contained within the *AntEye* application[5]. Only the Serial Communications application remains true-to-design.

Typical flow within AntEye will see an image captured from the front camera, processed such that we get a downsampled 90x10 image which represents the 360° view around the robot. This image is then used for optical flow analysis. The image and flow analysis are then made available to any thread which requires it. A thread will then use this data to run an experiment.

3.2.2 Arduino

The Arduino code acts as a bridge between the Android platform and the Dangu motor board, as there are no libraries which will directly connect the two. The Android code contains a Command class which allows the user to insert robot commands into the code; these are then transformed into serial messages by a broadcast library [2] and sent to the Arduino. The Arduino code contains a parser which will decode the serial messages and then call the correct method to execute the desired motor commands.

3.3 Modifications and Development

The modifications or additions listed below represent a small part of this dissertation, however for reasons discussed later these required a large amount of project time.

[5]This was the case on our inheretance of the platform for [6].

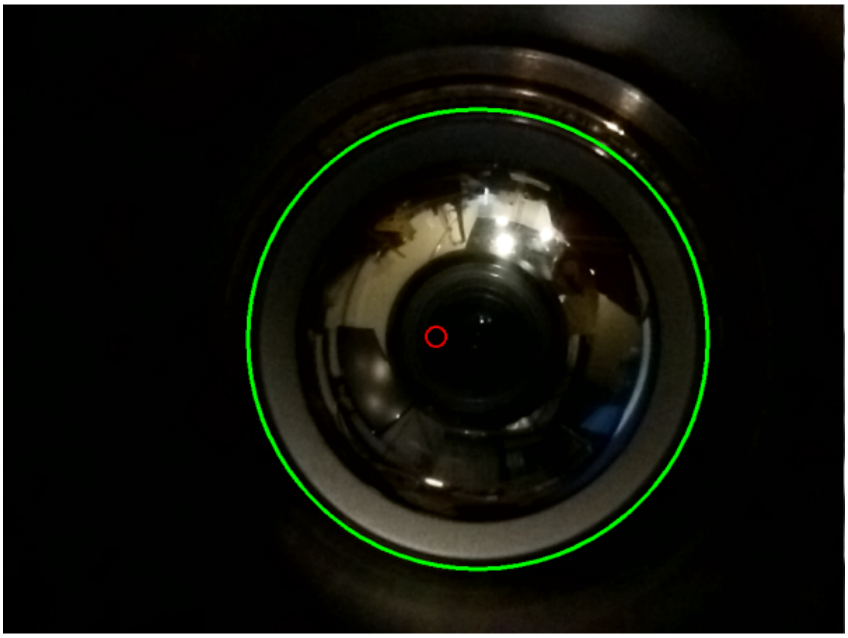


Figure 11: The hard-coded centre used for the polar transform (red), against the circle detected using the Hough transform (green).

Compass Sensor: While the Android phone has a built-in compass (used for other parts of this project) this is impractical to use as part of a control system. The communications delay is too great to allow accurate feedback control. As a step to introducing proportional control to the robot, we undertook an investigation into available compass sensors. Multiple sensors and libraries were tested and ultimately we settled on a Grove 6-Axis Compass & Accelerometer. This sensor remained reasonably accurate when tested alongside motors (a source of magnetic interference) and the library code was straightforward to include. The sensor was installed by the Informatics Workshop. While we have not used it for this project, we would encourage a future student to make use of this.

Ocular Calibration: The 360° camera attachment is held in place by a pressure sensitive adhesive which allows the position to shift if the attachment is knocked or removed. The lens was removed for use in another project and on re-attaching, it was noticed that the image was warped, indicating that the pre-processing was not working as required. The image pre-processing algorithm used a hard-coded pixel value for the centre of the camera attachment which was no longer correct (see Figure 11). We implemented a simple detection algorithm using the Hough transform available in OpenCV to allow the user to detect the new position of the camera attachment and set the centre accordingly. The calibration is not performed automatically as the Hough transform can be slow, and is highly dependent on lighting; instead, it need only be done if the attachment is known to have moved.

Code Refactor: As mentioned above, all behavioural code for the robot has moved into the AntEye application. In fact most code had been integrated into a single Android Activity (Java Class); this file contained multiple long threads dictating behaviour as well as a number of utility methods and robot commands. There were also a large number of global variables. The resulting file was over 4000 lines long and almost unusable. Approximately half of the code is useful for archival purposes only.

Consequently, it was decided that it was worth refactoring the codebase in order to make it easier to work with. All utility functions were split into their own static Util class, similarly the robot control commands were moved to the Command class. Runnable threads were moved into an archive package and split into classes according to their use: Mushroom Body, Central Complex, Optical Flow, or Old Navigation; that final group consists of the oldest code which is not used explicitly but is useful for reference. A single thread was left in the MainActivity file to act as a test thread. An amount of unused code was removed entirely.

Video Recording: In order to analyse the visual processing in more detail, a video recording utility was added to AntBot. Platform constraints mean that the utility is not implemented as originally desiged. A straightforward method using FFMPEG had to be abandoned due to library conflicts; this method would have allowed a video file to be recorded directly rather than the frame-stitching solution described below. The reason this proved difficult was that the recording had to take place after preprocessing (image unwrapping and downsampling to the 90x10 360° image). In order to achieve this, frames are passed off to a recording buffer after processing, the buffer is then processed asynchronously by a separate thread which saves the frames to a set directory. This does not impact the frame rate. The video frames are then retrieved from the robot and stitched together using a custom python utility (a simple wrapper around an FFMPEG terminal command). The recorded video can then be analysed offline. This utility was created to allow offline analysis of optical flow, however it also highlighted a major problem with the robot which was not previously known; we discuss this next.

Video Pre-processing Pipeline: When stitching the video frames from the recorder, we identified that the framerate was not quite right. The framerate on the robot was found to be an unacceptably low 2fps which greatly reduced the amount of information available from the visual system. In an effort to improve the framerate we isolated different parts of the processing pipeline. We found the best possible framerate to be approximately 14fps with no processing (this is computed simply as the number of times the *onCameraFrame()* callback function is called per-second). Adding processing steps back to the pipeline, two steps were found which caused the framerate to degrade. The first step was an image convolution algorithm; the unwrapped image does not line up with the direction of travel of the agent, so the resulting frame must be shifted (azimuth) so that the centre of the frame is in line with the forward direction. This step was reducing the framerate by approximately 5fps. Replacing the implemented algorithm with an OpenCV library implementation reduced the penalty to approximately 1fps. Similarly, an image downsampling algorithm had been manually implemented which cost around 5fps; replacing this with an OpenCV library implementation reduced the degradation to approximately 1fps. The full pipeline can now run at 10-12fps with the recording utility running on top. When simulating a model like the ECX, the framerate drops marginally to 8-10fps. We also attempted to move the frame processing into its own high-priority thread, however this resulted in numerous bugs and did not improve the framerate in any noticable way.

We do not know when the problematic code was added. In any project following its insertion, the framerate surely affected the performance of the agent. We can say with certainty that the algorithms used and tested in [6] were affected. Indeed, in [6] it is noted that few images were stored during Visual Navigation experiments. The low storage rate was thought to be a thread synchronisation issue, but it is now clear that the frames were not finished processing in time to be stored. In testing the matched filter collision avoidance system from [6] we note a significant change in behaviour after the framerate improvement. It is suspected that this is due to an increase in flow information, and therefore an increase in noise which can cause erroneous and erratic behaviour, not witnessed in [6]. This is one of the main issues that resulted in our lack of results for the ECX.

4 Methods

4.1 Collision Avoidance

Fitting AntBot with sensors which feedback to the higher neural models is prohibitively difficult (see Discussion). We therefore continue to look for a method of collision avoidance which can be integrated into the Central Complex which uses the available visual information. The matched filter model from [6] performed reasonably consistently but was initially considered too coarse-grained for integration into a modified CX. Our first approach therefore looked for expansion patterns in certain areas of the flow field.

4.1.1 Shifting Expansion Fields

We noted in research that the Focus of Expansion is used in many visual collision avoidance systems. Usually it is used as a stepping stone to compute the time-to-contact with an obstacle. Some papers describe using the FOE to determine a turn direction in the event a collision is detected [12, 15]. We also note that the often the FOE seems to shift to the deepest part of the image [15, 11] (though we should also note that in [12] an avoidance saccade is triggered *away from* the FOE); this is not an empirical observation, however it was enough for us to seek to test if the position of the FOE was predictable given an approaching scene.

If the FOE *is* drawn to the deepest part of the image then its location could be reliably used to choose a steering direction in a collision avoidance system. Furthermore, this has an intuitive integration into the CX. If we split the 360° image seen by AntBot into eight equal sections reflecting the eight cardinal directions of the CX model, we can use the horizontal position of the FOE as an input signal. The position would mark a “bump” of activity in a set of eight neurons (similar to the output layer of the eight MBON MB model). Ultimately we wish to see if the FOE will behave predictably in the presence of a looming obstacle, which relies on being able to compute the FOE reliably. When working with the FOE, [6] notes difficulty in computing it consistently. Often nonsensical values would be output by the computation. This was suspected to be due to a technical detail not known at the time of implementation and thus warranted further investigation[6]. Here, we managed to compute reasonably sensible values, though they were not consistent or predictable (indicating that the technical problems from [6] did indeed affect the results). To compute the FOE we use the OpenCV *calcOpticalFlowFarneback()* function to produce a dense optical flow field then use the computation from [7] (see Section 2.1).

While fixing the problems raised by [6] made the computed FOE more sensible in terms of general location, the lack of consistency warranted further investigation. For this purpose, the video recording tool described in Section 3.3 was developed with the goal of performing offline video analysis of the agent’s point of view. Thus, we can visualise the computed optical flow field observed by the robot under controlled circumstances (see Figure 12).

This analysis produced the following observations:

1. The optical flow field is very noisy: The majority of the flow produced is erratic and unpredictable.

[6] See [6], discussion of general purpose matrix multiplication in the OpenCV implementation for Java.

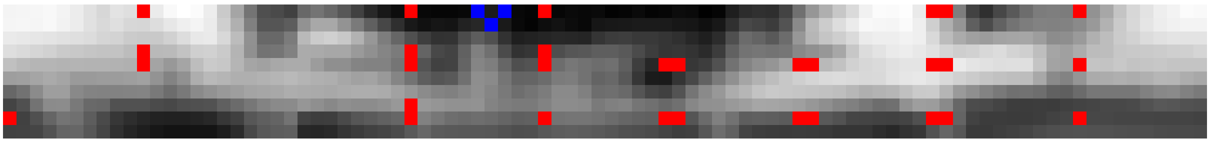


Figure 12: Frame 3 from Figure 21. A subset of points from the dense optical flow field observed by the agent. The agent is experiencing forward translational motion. These frames were captured after the framerate improvement from Section 3.3. The FOE is also shown in blue. There are no obstacles present in the arena.

2. The amount of perceived motion is exceptionally small: In simply looking at the video it is incredibly difficult to determine how the motion observed relates to the movement experienced by the camera. In most frames the perceived flow is as shown in Figure 12, where each vector is only about 1 pixel in length.

3. The FOE jumps randomly between frames: The FOE is not consistent between frames though it can appear to stay in the same approximate region (we do not believe this indicates anything significant). We cannot rely on its location when presented with a looming object in the frame.

4. Looming objects can be identified from the flow field: This observation explains the good performance seen by [6] when examining a matched filter system. Looming objects can appear as a large (potentially single frame) disturbance of the low-level noise produced by the flow field (Figure 20).

We can therefore conclude that optical flow cannot be used to compute any fine-grain properties. The flow field at large is just noise, and only drastic image changes result in a large disturbance in a specific region of the flow field. While this effectively precludes any further investigation into the FOE/depth methods which utilise optical flow, it does indicate one semi-predictable property of the flow field which could potentially be used.

4.1.2 Matched Filters

The matched filter model of collision avoidance as seen in [6] was used without modification for the experimental results in this paper. However, in an effort to establish the ECX model, we present here an adaptation of matched filter collision avoidance which can have its outputs integrated with the CX and MB models. We will use the language of artificial neural networks, but the principle is essentially the same as [6]. The neural approaches presented are all integrated directly into the CX so that we can make use of the existing architecture and information available (e.g. the heading information from TB1 and steering output from CPU1).

The neural model for collision avoidance features two types of neuron, ACC (ACCumulation) and MRSP (Movement ReSPonse). ACC neurons are key in a working system, and their representation can affect the behaviour of the model.

ACC Neurons: ACC neurons take on the role of the leaky accumulators from [6, 12]. The system contains two ACC neurons, ACC_{left} and ACC_{right} for the left and right sides respectively. We investigate three different representations of these neurons: Rate, Leaky Integrate & Fire (LI&F), and Reset Integrate & Fire (RI&F). The two I&F variants are only subtly different and behaved largely the same but we include them for completeness. We will consider them in chronological order.

The matched filter system produces a sum for the left and right hand sides of the robot. This sum denotes the difference between the expected motion and the observed motion in the image frame. If we take the difference between the two sums we can see if one side has experienced a significant deviation from the expected motion (i.e. a looming obstacle on that side). This difference acts as the input to the ACC neurons.

ACC Neurons (LI&F): For both I&F models, we need to set a lower threshold on what we consider an input. Each input must be greater than (absolute value) a lower threshold in an attempt to reduce noisy inputs to the ACC neurons. For more details as to the output from filter matching, please see [6]. Once the input has been thresholded, an ACC neuron at time t can be described as:

$$V(t) = \gamma \cdot V(t-1) + I_{ext} \tag{25}$$

where $V(t)$ is the membrane potential, t is the current timestep, γ is the leak current and I_{ext} is the external current flowing in - the thresholded difference of flow sums. We use terms like membrane potential and external current as these best describe the function in the language of simulated neurons. It should be noted by the reader that we do not explicitly model voltages and currents in the model though this is merely a case of semantics; these properties could be modelled more explicitly.

We also define V_{fire}, the value at which the ACC neurons fire. If $V(t) > V_{fire}$ for some ACC neuron, then the neuron that exceeded V_{fire} produces an output signal and both have their membrane potential reset to zero (the resting potential). The neuron that fires then governs the signal produced by the MRSP layer.

ACC Neurons (RI&F): The RI&F representation is almost identical. The difference arises from the leak model. In RI&F we reset the neuron periodically (e.g. every third timestep). While this is functionally an I&F neuron, it does not function in any biologically plausible manner. It is essentially the same as the leaky accumulators from [6]. We can describe the RI&F neuron at time t by:

$$V(t) = \begin{cases} V(t-1) + I_{ext} & \text{if } t \not\equiv 0 \bmod p \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

where $V(t)$, I_{ext}, and t are as above and p is the reset period (for example, if $p = 3$ the membrane potential is reset every 3 timesteps). Firing behaviour is the same as the LI&F model.

ACC Neurons (Rate): While I&F neurons provide an intuitive neural metaphor for the leaky accumulators, the CX model (into which this CA system is being built) uses a rate model for all other neurons (including our own MRSP neurons). As such we felt it appropriate to investigate such a representation for the ACC layer. In fact, using a rate representation requires only a single ACC neuron and allows for the generation of a steering response which is proportional to the size of the optical flow disturbance (in theory, the bigger the obstacle or loom effect, the bigger the response); neither of which are possible with the I&F representation. Recall, a firing rate given some input I is represented as a sigmoid (Eq. 10):

$$r = \frac{1}{1 + e^{-(aI-b)}}$$

where a and b determine the *slope* and *bias* of the function respectively. If we set $a = 5$, $b = 0$ we obtain the rate function shown in Figure 13. Importantly, we must scale down the input (difference of flow sums) such that it lies in (-1,1).

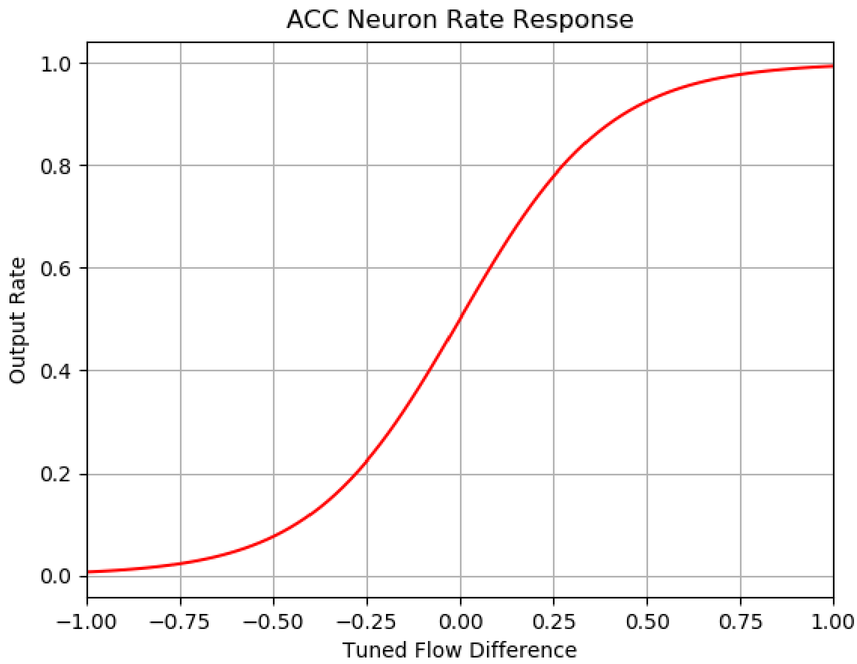


Figure 13: Function of firing rate response of the single ACC neuron against difference input; sigmoid (Eq. 10) with $a = 5$, $b = 0$. The input to this neuron must be scaled down to lie between -1 and 1.

If the flow disturbance observed on both sides is equal, then the rate response should lie around 0.5. A looming object detected on the right-hand side results in a negative input drawing the output closer to 0. Conversly, a looming object detected on the left-hand side will result in a positive input, driving the output closer to 1. We have, in a roundabout way, ended up with an optical-flow balance strategy - a flawed, simplistic, yet not entirely implausible hypothesis for collision avoidance in insects [10]. If our output remains close to 0.5 we need not react, close to 1 we must turn right, and close to 0 we must turn left. Thus, with a single neuron we can react to both scenarios; furthermore, the signal is proportional to the disturbance observed and we can therefore scale our steering command as necessary. This method is simple and flexible, though it is highly susceptible to noise, as we will see this is a critical flaw.

MRSP Neurons: There are 16 MRSP neurons laid out in the same fashion as the CPU4 layer. The MRSP connects to CPU1 using the same connection pattern as CPU4-CPU1 in the existing CX model. For clarity, the MRSP neurons always use a rate representation. In all cases the MRSP layer functions by generating a sine wave with the peak in the desired direction of travel; the method for constructing and positioning this sine wave is purely algorithmic, we did not investigate a way to model this behaviour using neural connections.

We first create an array of eight evenly distributed points on a shifted sine wave. The curve is shifted by $-\frac{\pi}{2}$ in the x-axis such that $\sin(\frac{\pi}{2})$ is in the first element (this is simply housekeeping, it makes index arithmetic easier when creating the output). The peak of this sinusoid array will denote the desired direction of travel, we therefore need to shift the array so that the maximum lies in the correct place. This *shift* is calculated as $shift = current\_direction + offset$. The current direction can be read from the TB1 layer (the TB1 neuron with the highest firing rate). The offset is computed based on the response from the ACC layer; this computation is different in the I&F and Rate cases.

In the I&F case, the offset will be -2, 0, or 2 depending on which ACC neuron fires (ACC_{right}, both/neither, or ACC_{left} respectively). This is a coarse but functional reaction which generates a saccade of approximately 30° in either direction. We included the case where both fire to act as a balance case, however, in practice this never happens. This is functionally identical to the behaviour displayed by the CA system in [6], piped through the CX structure.

In the Rate case, we instead receive a single input from the ACC layer. This input will lie in (0,1) and we need only a simple linear transform to choose an offset from this value. The transform has the conditions that $f(0.5) = 0$, $f(1) = 2$, and $f(0) = -2$.

$$f(x) = 4x - 2 \tag{27}$$

satisfies these conditions. The output of Eq. 27 is then rounded to the nearest integer to determine the offset.

In both cases, once the offset has been determined the sinusoid array will be shifted such that the maximum is in the desired element. The shifted sinusoid array is then extended to sixteen elements to match the CPU4 representation. The final extended sinusoid array is used as input to the MRSP neurons whose output is computed as:

$$r_{MRSP} = \frac{1}{1 + e^{-(5I-2.5)}} \tag{28}$$

The computed firing rates are then supplied to the CPU1 layer of the CX by utilising CPU4-CPU1 connection array. The CPU1 layer then generates the steering command. In fact, as the MRSP layer utilises the same neural architecture as CPU4, we can give the input to CPU1 as:

$$I^{(t)}_{CPU1} = r^{(t)}_{MRSP} - r^{(t)}_{TB1} \tag{29}$$

Note that this is in the case where we only the CA system is contributing to the output (i.e. no path integration or visual navigation). The TB1 subtraction is still required in the input to generate an avoidance saccade relative to the current heading of the agent.

An example of the signal propogation from ACC (rate-based) response to MRSP output can be seen in Figure 14. Input ($I = 0.25$) is supplied to the ACC neuron, its firing rate ($r = 0.78$) is put through the offset computation function ($offset = 1.11 \rightarrow 1$), and this offset is added to the current direction ($current\_direction = 4$ in the example) to give the total offset. The sinusoid array is rotated such that the maximum indicates the desired direction of travel (index 5), the sinusoid is extended to sixteen elements and supplied as input to the MRSP neurons; the MRSP firing rate is given by Eq. 28, and an example MRSP layer output is shown in Figure 14.

4.2 Path Integration

The path integration circuit in use is a slightly simplified version of that presented in Section 2.4. This version does not model the TN neurons or the Pontine neurons. The model is the same as that used in [9, 19], and the robot-based tests from [13].

We use the CX firstly for a round of experiments in which we wish to address a key methodological flaw observed in the experiments of *Zhang* [19]. We then use the CXMB variant (the 8-MBON model) as the basis for the ECX model. *Zhang* does not make it entirely clear (though it is stated) that the 8-MBON model is actually an adapted CX model. Distance measurements are arbitrary, there is no access to wheel encoder measurements. Unlike previous works, the agent moves only in the forward direction so we can reasonably assume that the robot will travel roughly the same distance in a set time interval.

4.3 The Complete System

A complete model has been constructed. Present on the robot is a singular model which, in theory, will perform collision avoidance, path integration, and direction-associated visual memory (simultaneously); however, we did not manage to collect formal results due to technical and time constraints (though some informal testing has been conducted). We present the model here and present the issues experienced and learning points in Section 6.

In an effort to keep the integration simple, the first version of this model allowed the collision avoidance system to fully overwrite any steering instruction in the event of a detected obstacle. While straightforward and intuitive, this method proved too coarse. If a full model was to be constructed, it was clear that a full neural representation for the collision avoidance system would need to be constructed; this and tested variants have been described above.

The final model takes the 8-MBON CXMB model and inserts the additional ACC and MRSP neurons required for the integrated collision avoidance system. The connections between TB1 and MRSP are the same as the connections between TB1 and CPU4, likewise the connections between MRSP and CPU1 are the same as those between CPU4 and CPU1. In fact, as the eight MBONs are expanded out to a sixteen neuron representation (see Section 2.5) the model can be visualised as the CX in Figure 5 (Left) with two additional layers of sixteen neurons between CPU4 and CPU1 plus one or two ACC neurons depending on the neural representation used. The final version implemented on the robot uses the rate representation with a single ACC neuron.

The output of the network can be selected by changing the mode of the input to the CPU1 layer. In this fashion, one can isolate each system and produce the steering output using specific systems. The key modes currently present on the agent are CA-only, and CXCA which are the isolated collision avoidance system and the combined output of the path integrator and the collision avoidance system respectively; adding further combinations should be trivial. The selection currently available is due to the component level testing that was in progress towards the end of the project. The weighting applied to each layer can be specified and ideally it should be dynamic. While the visual memory infrastructure is entirely present, it was not tested; however, as stated, it is identical to that presented by *Zhang* so we have no reason to think that this subsystem should not work to the degree presented in [19]. As all three systems produce a signal which is passed to CPU1 for the final steering output, the combined input to CPU1 could be stated as:

$$I^{(t)}_{CPU1} = a \cdot r^{(t)}_{CPU4} + b \cdot r^{(t)}_{MRSP} + c \cdot r^{(t)}_{MBON} - r^{(t)}_{TB1} \tag{30}$$

where $a + b + c = 1$, and $a, b, c \in [0, 1]$ represent the proportion of the output being governed by PI, CA, or VN respectively. Each subsystem uses the same connection array between itself and CPU1 (namely, W_{CPU4} from Section 2.5) so without weighting, each would have equal influence. For example, in the CXCA case, we use $a = 0.1$, $b = 0.9$, and $c = 0$ (see Results).

4.4 Tussocks

During [6], we note that the tussocks were extremely small in the view of AntBot; furthermore they were incredibly fragile, requiring repairs multiple times during the experimentation phase of [6]. For convenience, we construct new synthetic tussocks. The basic idea is the same, but the wood blocks were made larger, and the synthetic foliage sits higher. The two pieces were stapled together resulting in a much more robust structure. From video data it can be seen that they are far more prominent in the image frame, as required.

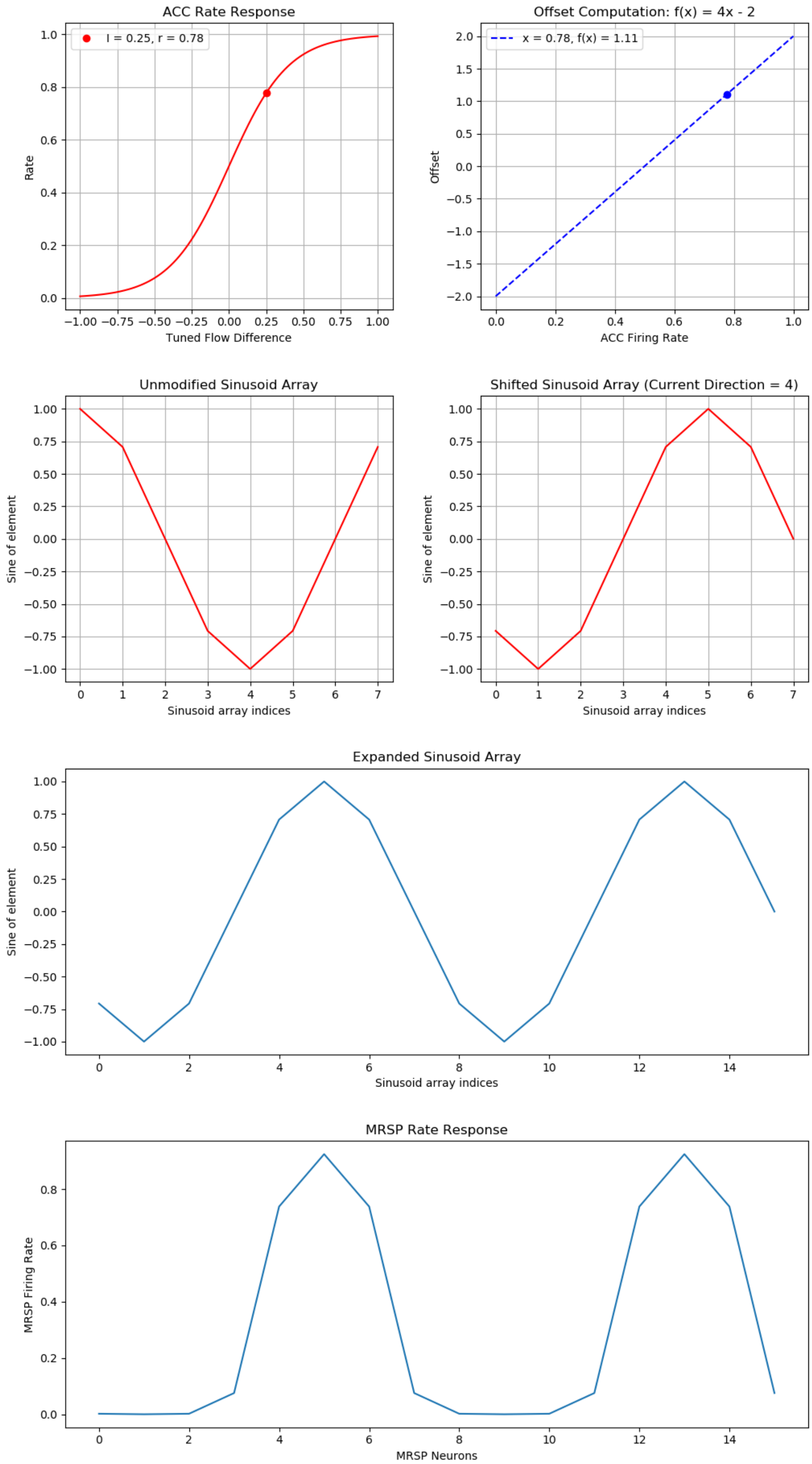


Figure 14: An example of signal propogation and generated output from the rate-based system (left-to-right, top-to-bottom). An example flow input is given as 0.25. The ACC rate response is then put through the offset computation function (Eq. 27). The default sinusoid function is generated over eight array elements. The sinusoid is then shifted such that the maximum lies at *current_direction + round(offset)* which in this case is 5. This sinusoid is then extended to a sixteen elements to match the representation in CPU4. Each element of the sinusoid is then supplied as input to the corresponding MRSP neuron to give the rate output in the final plot; as there are two neurons for each cardinal direction, the two maxima indicate a single direction. This MRSP response is used as input to the CPU1 layer, in this example a small right turn will be generated.

5 Experimentation and Testing

We re-use the experimental environment from [6]. A collapsible boundary wall is constructed on the robot-football pitch in room G.17 in the Informatics Forum. Small "tussocks" are used as objects in the environment (e.g. to avoid or to provide visual information). A sequence of VICON motion-capture cameras are arranged in a ring above the pitch and allow accurate tracking of the agent's movement and heading.

5.1 Path Integration

The first round of tests aims to establish firmly that the CX model is capable of robust path integration in the real world. The CX has undergone reasonably extensive testing on the AntBot, however, two key issues were noted across all tests.

1. The outbound route of the agent is always the same. While this route was randomly chosen, it was the same in all trials. It is entirely possible that any parameter tuning was performed with respect to this route. Good performance on a single route does not indicate good performance as a general path integrator.

2. The outbound route always finishes with the agent directed at the nest. This means that the robot could potentially navigate home by making little to no change in its trajectory.

The second problem is more of a symptom of the first; to remedy both problems we need only fix the first. This can be done in a few ways. Firstly, we could simply generate a randomised turning command at set (or even random) intervals. There are many variations on this theme, however the original goal of this paper was to provide a complete navigational system. It makes more sense for us to use a collision avoidance system to generate the outbound routes.

We use the collision avoidance system from [6] to fill this role. For the standard CX experiments, this collision avoidance system was entirely unmodified and separate from the path integration circuit. The outbound route was governed by the CA system (with the path integrator updated on each step) and the inbound route was governed entirely by the CX. These experiments pre-date the neural collision avoidance system so there is no way to provide CA and PI behaviour simulaneously. Thus, the experimental arena was carefully chosen based on known properties of the CA system.

For half of the runs the agent was started from the tape markings in the South West corner of the arena and for the other half the agent was started from the marking at the South end. We place only two tussocks in the South East corner of the arena to ensure the agent is always directed into the arena. The full reasoning behind this obstacle placement is given in [6]; while navigation towards the arena wall is still successful CA behaviour, it is not useful for testing higher navigational systems. The rest of the arena is left clear. The CA system from [6] is sensitive enough to provide non-deterministic reactions where no direct obstacles are present whilst successfully avoiding the arena walls (see CA experimental results from [6]). We therefore manage a non-deterministic outbound route based on CA while keeping the arena clear for the PI stage of the test. These experiments were conducted before the framerate improvement so CA behaviour can be considered comparable to [6]. Any test in which the CA system failed catastrophically (direct collision) was to be ignored as only the CX was to be tested, though there was only one such case.

The agent was permitted to make a 180° (approx.) turn at the end of the outbound route. In most cases, this does not make the robot point directly towards the nest. The justification for this procedure can be seen in Figure 15. The output from the CX model is a left/right turning instruction with some strength; the greater then angle between the current and home vector, the greater the output signal in the required direction, however, the output is not the precise angular correction required. As such, the agent required a large turning arc (and a long time) to reorient itself with the home vector before it could even begin the journey. Encouraging as this behaviour is for the performance of the CX circuit, we felt it more practical to avoid it for formal experiments as most outbound routes did not leave enough space for these arcs.

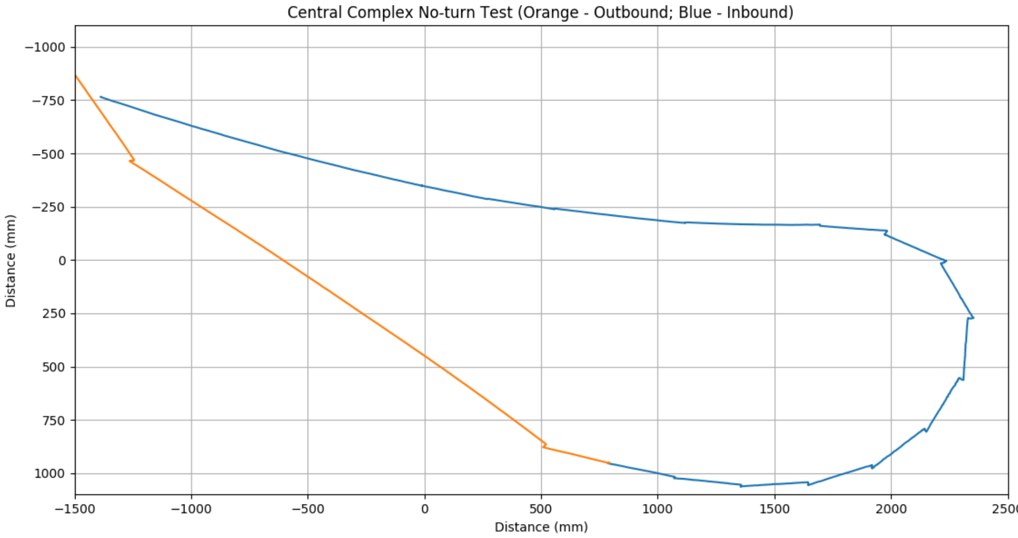


Figure 15: A test recording from the Central Complex PI experiments. The agent successfully navigates home but requires a large turning arc to point back towards the nest. This is because the output of the CX is not a precise angle but instead a turn direction with some strength. This behaviour was our prompt and justification to include an about turn on completion of the outbound route; the space present in the arena simply did not permit such routes for formal experiments.

Success was measured by the ability of the agent to touch the tape square from which it started. This could be any part of the robot (e.g. a tread running over the corner of the square). The agent was stopped manually once either a success state was reached or a success state could not be reached; for example, if the robot collided with the arena wall during PI, or passed the start point without touching it.

Finally, we should note that the locomotion of the robot is different to that of previous tests. Previous works with the CX model look to keep the motion of the agent continuous. The impracticality of such motion on the AntBot is noted in [6] and below in our own discussion. Instead, we keep the motion in discrete steps; the agent moves forward, stops, turns on the spot, and continues. In the case of the PI experiments, the oubound route is made up of steps broken by turns generated from OFCA at irregular intervals, the inbound route consists of steps broken by regular course correction turns (e.g. every two seconds, the CX output is checked); this behaviour can be seen clearly in Figure 15. The distance measurement is still updated continuously; we use time-travelled as the distance metric (assuming we travel the same distance in the same amount of time). While this is generally considered a poorer way to measure distance, many problems are mitigated by the fact the agent only moves in the forward direction, and to a degree this model should be robust to *rough* or *noisy* inputs. We are pleased to report that the CX performs well, even with a problematic distance metric.

5.2 Collision Avoidance

5.2.1 Shifting Expansion Patterns

The initial tests for the shifted expansion pattern system involved computing a FOE and checking for its horizontal position in the image frame. Our original hypothesis was that the FOE would be drawn to the deepest part (frontal) of an image, and therefore its horizontal position would dictate the desired direction of travel. We would, of course, take any predictable behaviour which could be used for collision avoidance.

The agent was set on a collision course with a cluster of tussocks and watched for consistent reaction behaviour. No consistent behaviour was observed. To test our hypothesis completely, it was clear that further work was needed to verify that there was no usable behaviour. The processed image frame (the view of the agent) can be viewed on the smartphone screen; the view is too small to pick out any details of a flow-vector field. As a general point, testing optical flow behaviour on the robot is quite difficult due to the field of view. Even outputting an FOE location and moving an object in front of the robot cannot tell us if we are producing the desired behaviour; firstly, because the object will not move realistically in the frame; and secondly, because the robot will observe motion from all directions (due to the 360° view) which would not normally be present; for example, the body and arms of the tester manipulating the object will not match the pattern of motion which would be observed during locomotion, thus affecting the output. This is true for all visual testing with the robot; it is hard to produce reliable output as the input is likely unrealistic.

Two solutions to this problem have been developed over previous years: *LogToFileUtils* developed by *Zhang* to provide textual logging utilities where it was not normally possible to use such logs [19] and *StatFileUtils*, a variant of *Zhang's* logging utility which produced regular file output which could be easily parsed to produce graphical plots from recorded data [6]. Neither were useful in testing the FOE position; we could log it, but there was be no way of correlating its behaviour with the agent's view of the environment. Instead, we chose to implement a video recording tool which would allow us to record controlled runs of the agent through an environment. Furthermore, this video could be processed offline using the same systems implemented on the robot, then scaled up to give a much clearer view of what the robot was actually seeing. This proved incredibly useful and allowed us to immediately dismiss shifting expansion patterns as a method for collision avoidance on the AntBot.

The recording tool was used to show the infeasibility of this method informally, however, we present below some frames from formal recordings, taken to demonstrate our findings for the purpose of this paper. Two such recordings were taken, both in the experimental arena. The first was taken with no objects present, and in the second, the arena contained two objects staggered to the right and left of the robot's path. These objects were not in the robot's direct path. In both recordings the direction of motion was entirely translational in the direction of view; no turns took place. The robot was permitted to move from one end of the arena to the other without interruption, aside from a manual stop which was done with minimal disturbance. The arena can be seen in Figure 16, the robot's intended path is shown in red.

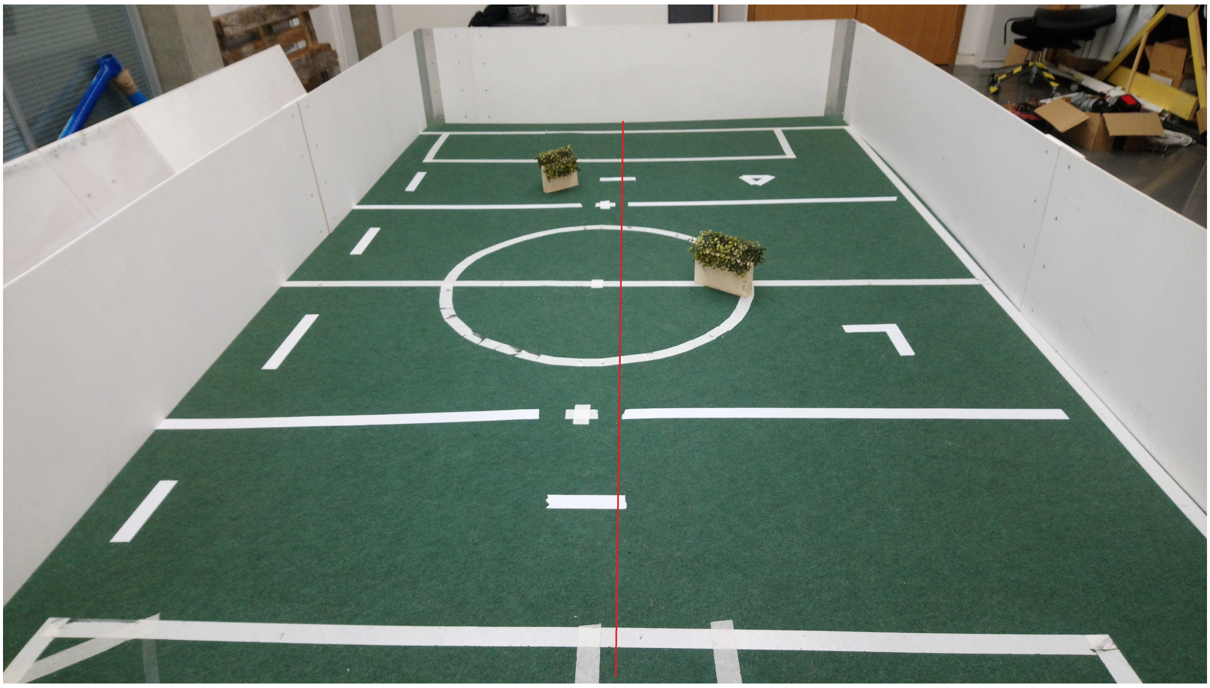


Figure 16: The non-empty arena used for testing the shifting expansion field system. Tussocks were set to the left and right of the robot's trajectory (shown in red) so that they were not on a collision course but should still affect the expansion field. We had hoped the FOE would either remain central or drift horizontally (in a consistent fashion) as each object was passed. Note that the agent was not started directly on the tape square, it was offset as shown by the path; this was oversight on our part.

5.2.2 Neural Matched Filters

The testing for the neural matched filter system was far more simple. In this case, we know that the underlying method (matched filter collision avoidance) is sound, we just wish to translate it into a neural representation. Thus, the first thing we need to test is the ability to generate an arbitrary turn by inducing activity in the MRSP layer of the model. We take the current direction from the TB1 layer, generate our sinusoid array and shift it to the left or right of the current heading. We can then check if we can generate consistent steering commands.

We then test the three different implementations of the ACC neurons separately. Each type was tested initially in an empty arena. This basic test ensured that reactions were being generated and that the system was responding in a sensible, sensitive fashion. We then add an obstacle. For each ACC implementation, the agent was started in the South end of the arena, on a collision course with a cluster of tussocks positioned in the centre. We wished to see if the neural system would replicate the performance of the neural matched filter system from [6] by starting with a variation of the most basic obstacle avoidance scenario presented (Table 2, Arena 2 from [6]).

No system performed consistently well in the presence of an obstacle, however the I&F variant[7] provided sufficiently consistent reactions to observe some runs of the ECX.

5.3 ECX (CXCA)

The testing methodology for the ECX was to test each individual system and then the full combination. There is a set order to this testing as each step of the integration requires some prerequisite (except CA). The CA system must be tested first, then the combination of CA and PI (CXCA) as we need working PI to learn homeward visual routes, finally, the MB can be connected and the full ECX can then be tested.

All systems are implemented and available; we present testing only up to the CXCA implementation. The CXCA was tested (informally) by allowing the combined system to navigate homeward following a successful collision free outbound run of the CA system. The combined model was only tested using the I&F variant of the neural CA model as this version managed successful collision avoidance behaviour. As CXCA tests followed successful CA tests, the CXCA is tested in the same environments: an empty environment and one with a single large cluster of tussocks.

[7] As behaviour between the two I&F variants was identical, we will discuss them together going forward. Most testing took place with leaky-I&F ACC neurons.

6 Results

6.1 Path Integration

We are pleased to report good performance from the CX model. We observed successful homing in 7/10 tests performed, though in all cases the agent proceeded in the correct general direction. Of the failures, two were minor and one was more significant. The full results can be seen in Table 1.

Test	Start Point	Success/Failure	Distance from Start Point (mm)
AB_CX_1	SW	Success	306
AB_CX_2	SW	Success	348
AB_CX_3	SW	Failure	517
AB_CX_4	SW	Success*	325
AB_CX_5	SW	Failure	581
AB_CX_6	S	Success	356
AB_CX_7	S	Success	304
AB_CX_8	S	Success	296
AB_CX_9	S	Success	311
AB_CX_10	S	Failure**	458

Table 1: The compiled results from the CX path integration experiments. (*) While this test succeeded, it was close. The CA system failed at the last section of the outbound run which could be the cause of the near-failure. (**) We consider this the worst failure; while this run is not the furthest from its start point, it can be seen from Figure 17 that the agent actively deviates from its route.

VICON recordings from all tests are available in Appendix B. While the results appeared good under observation and in the figures, we must note the distance measurements in Table 1. These measurements are taken from the central axis of the VICON skeleton applied to the robot, approximately the central axis of the robot. Due to the size of the robot, we must note that our success criterion may be deceptive. The closest arrival at the nest is 29.6cm; while the robot did indeed make it back to its starting box, this is still a reasonably large deviation. Given that the agent's general direction is almost always correct (except Figure 17), we think it is reasonable to presume that the deviation could be due to the inaccurate odometry; counting timesteps where an agent is in motion is generally considered a crude and inaccurate method of tracking distance travelled, we use this method as it was all that was immediately available on the robot (see Discussion).

An example of favourable behaviour (under experimental protocol) can be seen in Figure 18. In this case, the agent is already close to its home vector but still corrects to achieve more accurate homing. This is also the run which achieves the best performance by the distance-to-start measurement.

While we still consider these experiments to be a successful demonstration of the capability of the CX (especially with the early test run in Figure 15), we must acknowledge certain methodological issues. Firstly the deviation and secondly the run length. All runs can be seen to be very short. We think it would be prudent to repeat these experiments with longer runs and more accurate odometry (see Discussion).

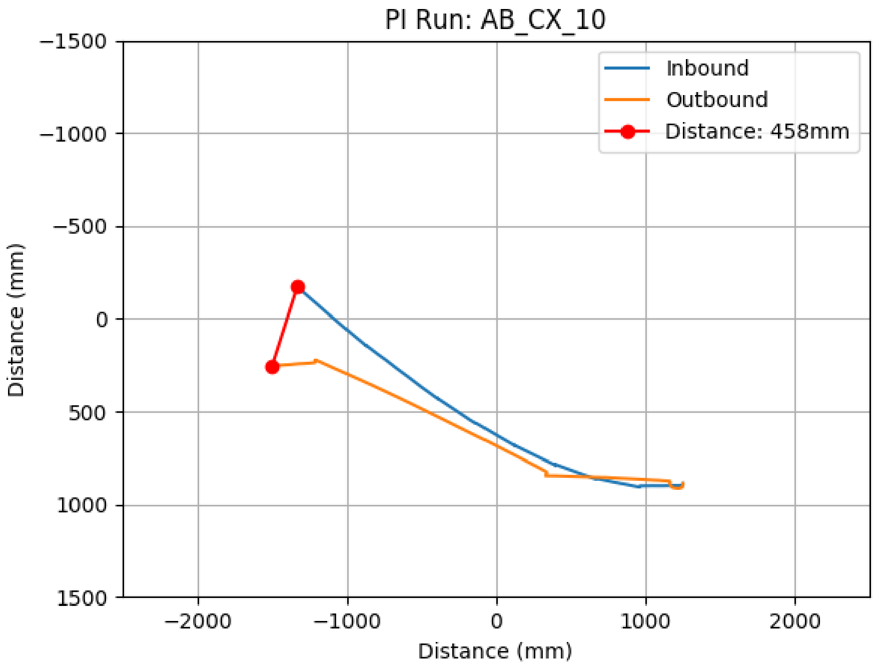


Figure 17: PI test AB_CX_10. This was considered the worst failure of the system despite not having the greatest deviation from the start point.

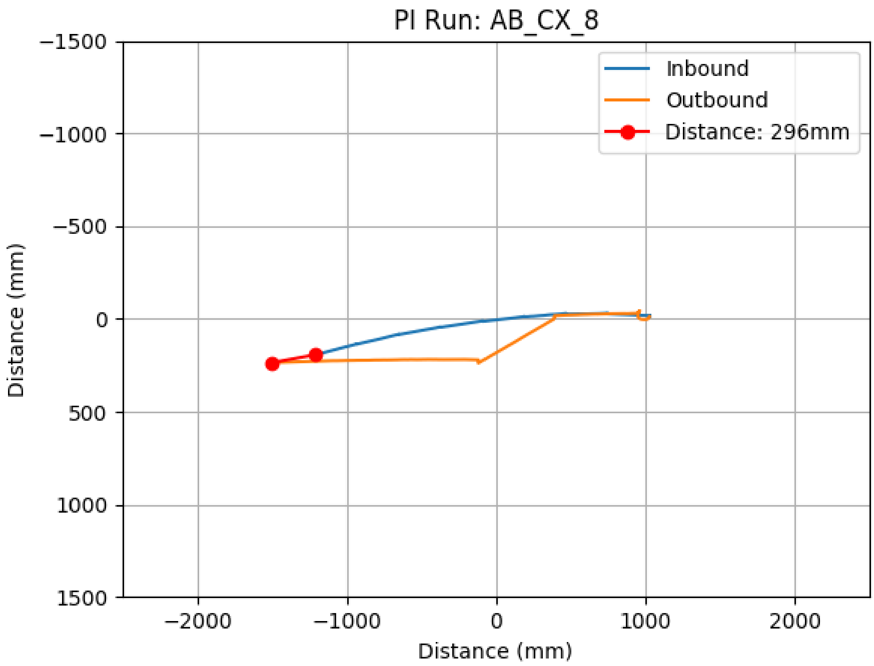


Figure 18: PI test AB_CX_8. Arguably the most successful recording from the formal experiments. The agent still corrects to achieve a better homing path, despite the fact it could likely have travelled in a straight line.

Despite our justification, it may also be prudent to remove the 180° turn at the end of the outbound run; perhaps 90° would allow the network to perform corrections while still removing the need for large loops. In some recordings, it can be seen that the robot is still pointing directly home after its outbound run (Figure 19), though this is not always the case.

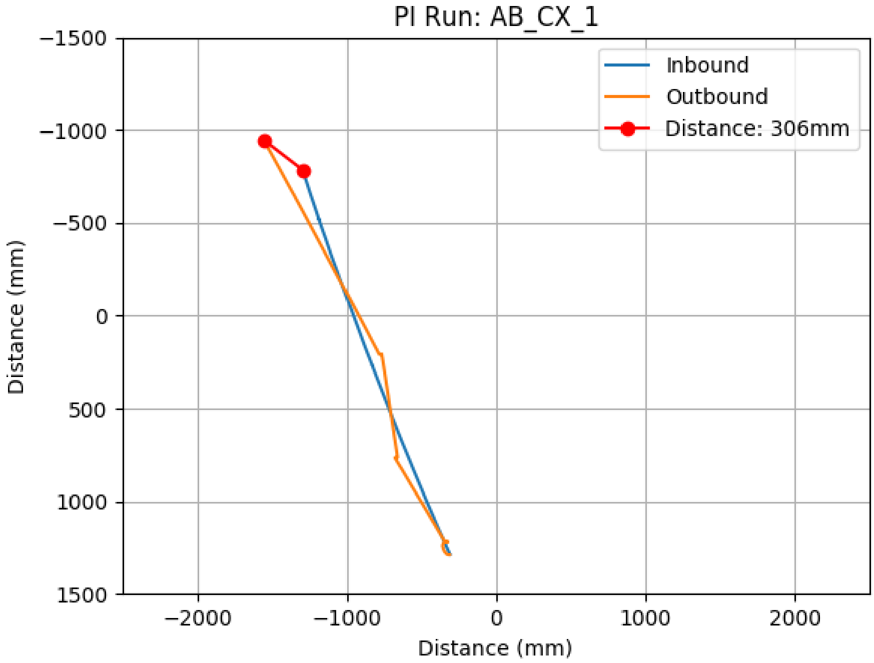


Figure 19: PI test AB_CX_1. In this case, we note the exact same experimental flaw we sought to avoid in which the robot requires no correction to home successfully.

6.2 Collision Avoidance

6.2.1 Shifting Expansion Patterns

The optical flow experiments conducted allow us to state with a high degree of confidence that the FOE cannot reliably be used as the means to generate a steering command in the case of the AntBot. We limit our conclusions to the AntBot as:

1. This is the only platform on which we have tested the predictability of the FOE.
2. The works which inspired this method achieved success in using the FOE to direct motion [12, 15].

Therefore, we conclude that it is likely that the AntBot's unique 360° field of view and low resolution vision play a major role in the problems experienced. For an example, consider the five image frames presented in Figure 20.

These five frames show how the FOE tracks across the azimuth in the course of approximately half a second. This certainly allows us to say that the FOE is not drawn to the deepest part of the image frame, which should be centre to centre-left across all five frames. These frames would also indicate that the FOE position is not predictable given the motion experienced - it is in fact more accurate to say, the fine motion perceived is not predictable. To show this more concretely we can look to another

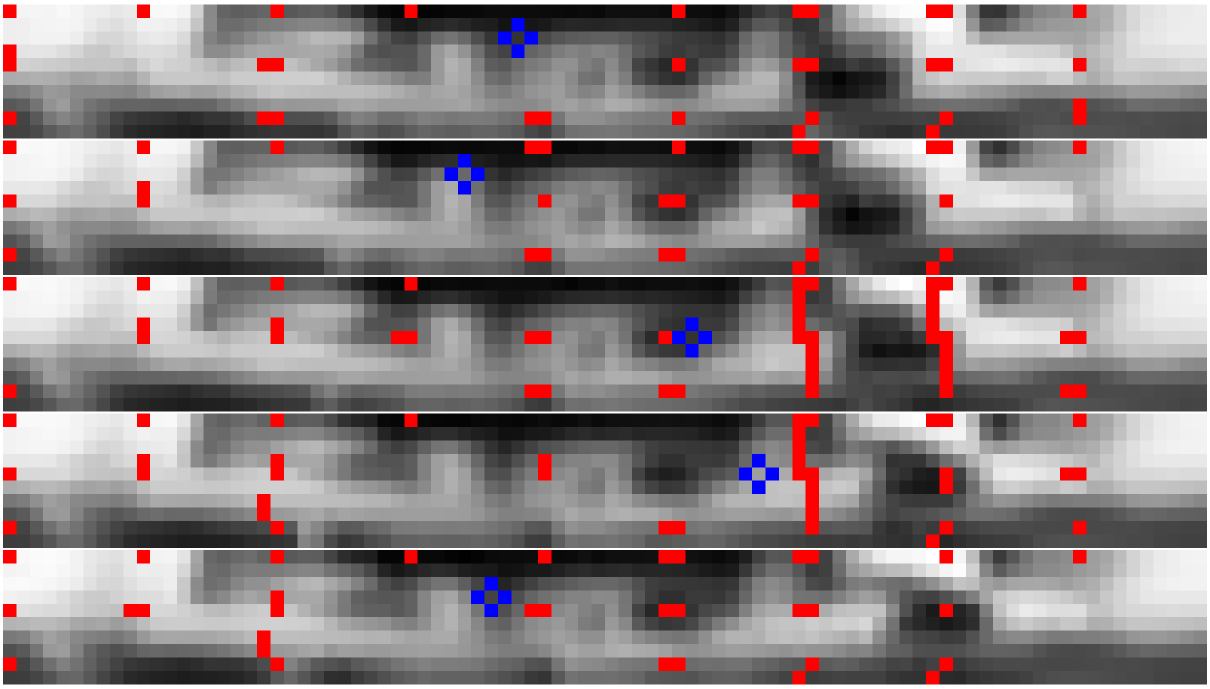


Figure 20: Five consecutive image frames with optical flow information superimposed over the top. It can be seen that the FOE position (central pixel of the blue cross) is not consistent across multiple frames. This figure also shows the disturbance caused by approaching a tussock on the right hand side (3rd and 4th frames). Video captured at approximately 10fps (i.e. these images were captured accross roughly 0.5 seconds).

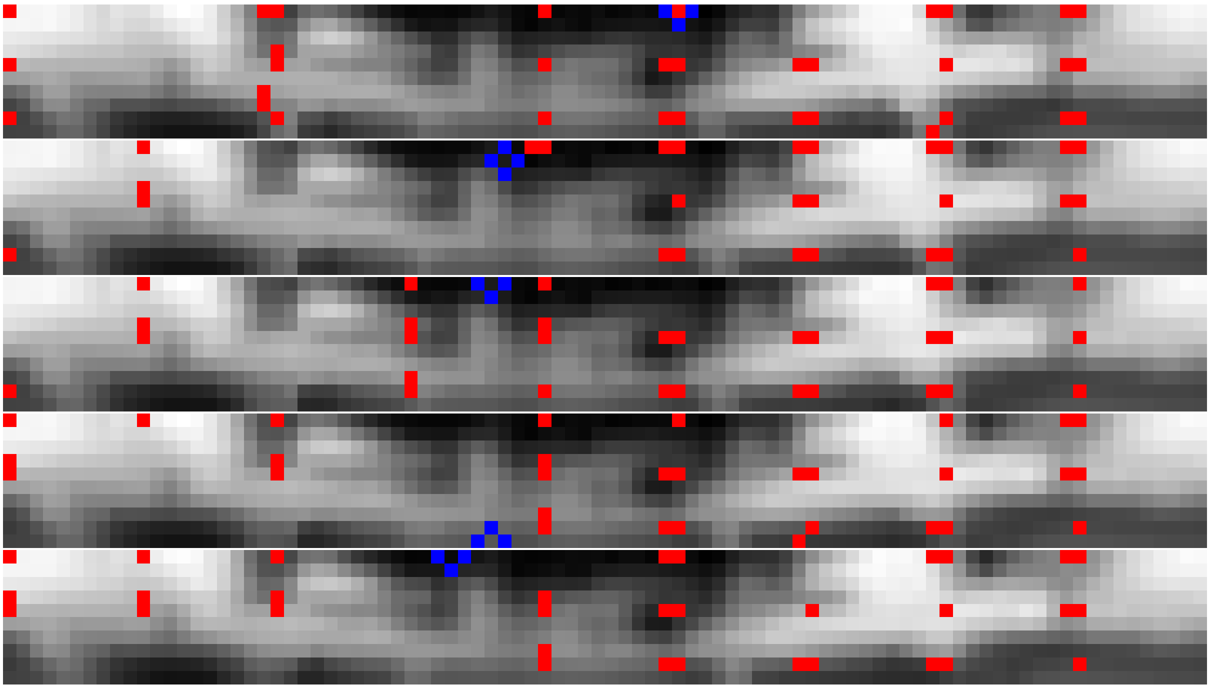


Figure 21: Another set of consecutive frames, this time captured in an environment with no objects present. The FOE is still unpredictable, though it does not move as drastically in the horizontal axis.

example in Figure 21.

In this case there are no obstacles present in the arena, and thus there should be minimal sources flow field disturbance. Nevertheless, we can still see that the FOE jumps, though not quite as drastically. The views are perhaps deceptive due to the pre-processing but for clarity we note that each pixel represents 4° of horizontal arc. Even small FOE jumps in the image frame are large with respect to the world.

While it would be interesting to test empirically such use of the FOE in a different setting, we can conclude that the FOE may not be used to govern any control output on the AntBot; indeed, we would go so far as to conclude that the FOE is of no use to the AntBot at all.

The remainder of our results discussion arises from developmental testing. These are not results from which concrete conclusions may be drawn, however they may provide insight for anyone considering working with the neural filter model for collision avoidance, or the ECX, as a continuation of this work.

6.2.2 Neural Matched Filters

The three variants of the neural filter model involve using the different representations of the ACC neurons. In testing, the I&F versions performed identically under observation, so we will discuss them as one.

I&F: Performance of the system was promising though inconsistent, however collision free behaviour was observed. Behaviour was similar to, though not as consistent as, that observed in [6].

In the obstacle free case, the system mostly managed to avoid the arena walls. The behaviour seemed more sensitive than that given by [6]. The algorithmic matched filter system also experienced problems in an empty arena, so similar behaviour in the neural model did not prove discouraging.

With an obstacle present, the behaviour of the system was initially positive. The agent did in fact manage to navigate around a single large obstacle successfully on multiple ocassions. However, the behaviour was inconsistent; in many cases the agent would turn towards the obstacle coming very close to collision (or actually colliding) with the obstacle. On some occassions the agent would avoid the obstacle and then collide with the arena wall. Performance here was positive enough to warrant testing with a sparsely occupied environment (two tussocks per pitch quadrant); at this point the observed performance broke down, collisions became far more common with many being caused by unnecessary turns *towards* obstacles (i.e. the agent's path was clear but a turn was triggered, often putting the agent at immediate risk of a collision). The architecture is clearly functioning as intended, however, further tuning of the input is required to obtain consistent collision avoidance behaviour (see Section 7.2.1).

Rate: The rate model provided no consistent reaction (in both the empty arena and with a single large obstacle present), however, this is very likely due to the fact that the inputs are not filtered at all. The flow sums for each side are computed, adjusted, and their full difference (scaled down to lie in (0,1)) is supplied as input to compute the firing rate. In the I&F and algorithmic variants this problem is mitigated by the *accumulation threshold* parameter. Such a parameter could be added to the rate model, however, we propose a solution at a lower level.

We believe the solution lies in cleaning the computed optical flow field. This could be done simply by filtering out all vectors which do not meet some threshold for length (this is distinctly different from the thresholds applied in [6]). Whilst simple, this is potentially effective method as it can be seen in Figure 20 that looming obstacles result in a few vectors with large displacements; all other information could be considered noise. Alternatively, different filters[8] or even a dedicated sensor could perhaps be used (see Discussion).

Although the rate model as presented does not behave consistently, it is again clear that the architecture functions as intended, which represents a significant step towards integrating CA behaviour into the CX structure. The rate model would be our preferred approach going forward. We feel it better fits the existing CX architecture and also allows a proportional response to be generated from a single neuron. It is a more flexible aproach to the simulation. Based on our previous experience with optical flow models for collision avoidance we are confident in our conclusion that the behaviour observed can be explained by a noisy optical flow field.

6.3 ECX (CXCA)

As stated in Section 5, the ECX model was tested as an extension of successful CA runs and thus, was only tested using I&F ACC neurons. Futhermore, only the combination of PI and CA was tested. In an empty arena, successful homing was observed in several (though not all) tests; these cases are promising results, as it was clear that the CA and CX systems were interacting to produce an output.

With the obstacle present, homing was never sucessful but desired behaviour was still exhibited in some tests. On these occassions the agent navigated around the obstacle with minimal disruption to the home vector given by the path integrator. Once the agent had passed the obstacle however, the trajectory often deviated from that desired. This behaviour is caused by erroneous outputs from the (hypersensitive) CA subsystem. We reduced the weighting applied to the CA in the output to 50% (originally, 90% of the steering command was given by CA, 10% by PI) and performed the test again. The steering generated was better with respect to the home vector, however, the robot could not home correctly due to greatly increased numbers of collisions.

The 90% weighting applied to CA seems high however, in an ideal scenario, the CA system should only be contributing output in an immediate collision scenario (i.e. when it is critical that the CA system governs the steering). We therefore considered it appropriate to set the weighting high; it is understandable that this would cause problems in the cases where the CA system is hypersensitive. We also note that, even with only a 10% weighting, the path integrator was still contributing noticably to the output. The difference between the combined system and the prototype in which the CA generated all of the output was significant.

The modification of component weightings was applied only to check that the CA system was indeed the cause of the unsuccessful homing once the agent had passed the obstacle. The effect of component weightings, or even applying dynamic component

[8] The flow filter in use is still the filter implemented by *Scimeca* for speed retrieval[9, 6].

weighting would be an interesting branch of investigation for any formal experiments using the ECX. We believe the behaviour observed in testing represents a significant conceptual step towards such formal experiments with a complete base model for insect navigation.

7 Discussion and Future Work

7.1 Conclusion

We have presented here an investigation into a unified navigational base model, inspired by the insect brain. We show experimental results demonstrating the capability of the Central Complex model as a path integration circuit over random outbound routes; we also note that this good performance was achieved with what is generally considered to be a poor odometric measurement, further demonstrating the strength of the model. To complete the navigational system, we investigate and present evidence against a collision avoidance model based on the focus of expansion of an optical flow field; furthermore, we present a neural adaptation of the matched filter collision avoidance system from [6]. This neural adaptation investigates three different neuronal representations resulting in two different architectures; the neural matched filter model has been integrated into *Zhang's* CXMB model to provide a conceptually complete base model for insect navigation. The code for both architectures has been added to the AntBot, but currently the rate-based system is in use.

We also developed several technical advancements to the robot which address many of the constraints we experienced during [6] (MInf Part 1) and this work. These include a code refactor to clean up the existing codebase, the addition of a compass sensor at the Arduino level, a calibration system to auto-detect the position of the 360° camera attachment, construction of a video recording tool to record the agent's point of view during tests, and finally, the identification and correction of a major bug in the visual pre-processing system, now known to have affected past work.

7.2 Future work

7.2.1 Building upon the work presented

We are pleased with the overall progress made in the development of AntBot, though there are always developments and improvements which could be made.

CX Experiments: The experiments presented undoubtedly show that the CX model functions well as a path integrator, however there are some issues with our methodology. Firstly, the outward routes are quite short and not as varied as we had hoped. Experiments demonstrating the network's capabilities over a longer, more varied (but still non-deterministic) route would be of value. Our routes were restricted by the experimental enclosure, which in-turn is restricted by the size of the pitch in IF G.17. There is no reason a larger enclosure (which is still viewable by the VICON system) could not be built, we simply used materials which were immediately available during [6]. Alternatively a smaller robot may be used.

While the enclosure itself may seem like the limitation here, we believe it is valuable in terms of providing some degree of control to the visual scene experienced during experiments. As such, we recommend the continued use of some enclosing wall. We also recommend continued use of the robocup pitch as it is lit consistently, is equipped with VICON, and is self-contained within a secure lab (meaning that it will not be changed or occupied without warning).

Neural Matched Filters: We do not present formal results for the neural matched filter system as we encountered unforseen issues after fixing the framerate bug. However, we remain convinced that this sytem is conceptually sound. We encourage others to work with the rate-based system, as, after development, this seems the most intuitive, efficient, and useful representation. We suggest looking at the flow vectors produced during the filter matching stage (*filterCollisionAvoidance()* in MainActivity.java) and looking for a way to filter out low level noise in the flow sum calculation. Furthermore, alternate flow filters could be investigated.

Extended Central Complex: Functioning neural matched filtering (or an algorithmic alternative) is a pre-requisite. We would strongly encourage future work on this model. If some reliable collision detection input can be provided (note this need not be based on optical flow), then the ECX could be comprehensively tested. Our original vision for this was a multi-stage navigational exercise; the robot performs an outward trip using CA alone, homeward using CXCA (during which a visual route is learned), then outward once more using either visual information or an inverted home vector in CXCA to learn a visual outbound route. On completion of the second outbound trip, the robot should have all of the information it requires to make the trip reliably, from visual memory.

To apply the different systems dynamically, they must be weighted at each stage with the initial goal of gradually reducing the reliance on collision avoidance and path integration, using vision almost exclusively. A further advancement could see a system which can place some *confidence* in the output from each system, thereby choosing the one which gave the best chance of successful navigation.

7.2.2 Platform

Software: While our code refactor was extensive, there remain aspects of the software which could be developed. There are still many varibles which are global to the application (referenced via the MainActivity class). This is not only bad practice in a software engineering sense, but it makes tracing variable updates difficult. We removed many such variables have been moved where possible but there remains much to do.

Most of the unused code which is still present is commented and/or specifically marked for archival purposes. We would recommend removal of any and all remaining dead code which has been missed. Likewise, many unused variables were removed but some remain; great caution should be applied when modifying these variables. Code which seems unused may be used indirectly (even in cases where Android Studio shows no uses). Liberal use of version control is recommended.

Library upticks may be useful, but we are not sure of the extent to which this may be done. We experienced difficulties in using some libraries during the development of the recording tool due to a version conflict between JavaCV (which contained required FFMPEG bindings) and the version of OpenCV present on the robot (3.0.0, compiled from source). We also experienced issues when investigating the few possible alternatives as the minimum Android SDK version for the libraries was greater than the target for the project; to complicate matters further, a later version of the Android SDK was incompatible with the Google Nexus 5 platform. The project and dependency versioning and package management requires cleaned up. A good start point would be to move the project away from the source version of OpenCV and onto an appropriate Maven[9] package. This would, however, require updating many legacy imports and changing (or removing[10]) legacy code.

Hardware/Firmware: Potential upgrades to hardware are extremely limited. During this project, we note the difficulty in adding a single compass sensor to the robot, with the goal of later adding proportional control code. The Arduino has a shield board connected which governs all of the connections to the motor board. This board occupies the connectors usually used for the I^2C bus, however, it is possible (to a limited extent) to piggy-back on the shield connections. A better solution which could be implemented by the Informatics workshop, would be to have a different shield made up with provisions for a break-out board which would permit easier addition and testing of different sensory systems. We would encourage a future student to make use of the compass sensor to rewrite the control (specifically turning) code such that the locomotion of the robot could be made more accurate.

We considered adding an ultrasonic sensor array to the front of the robot (three sensors, mounted in an arc on a custom 3D printed frame). The technical implications were too great for the stage at which this was considered. However, there is merit in future work looking into the serial broadcast library and learning how to send serial messages from the Arduino to the smartphone; this could be of great benefit. The chassis has built-in encoders which could be queried to provide more accurate odometric measurements to higher models; furthermore, this could be used to introduce an algorithmic alternative to the matched filter model for collision avoidance, which could be useful in testing higher models (e.g. the ECX) where visual collision avoidance may be unreliable.

We strongly encourage further investigation to get the most out of the available platform, however, we remain wary of the serial communications. We note in this work and in [6] that executing commands requires a delay, however, we do not if an event listener on the smartphone which simply listens for specific serial events would require such delays. This may prove to be a faster and more usable approach than we might expect and is worth further research.

A New Platform: We have listed some potential avenues for improvement which could breath new life into AntBot. In particular, if I^2C connections (and other I/O ports) can be made available (or more easily accessible in some cases) on the Arduino and the serial communications improved, this could eliminate many of the constraints we have experienced in this work. Examples of such constraints include: no access to accurate encoder information, no proportional control at a high level, and no algorithmic alternatives to bio-inspired systems which could be usfeul as experimental controls for comparison.

Even if the more problematic hardware/firmware limitations were lifted, there are remain some constraints inherent in the Android OS and codebase. Resolving these problems may require as much work as developing something new from scratch; the existing codebase (including the serial communications system) is almost entirely undocumented and poorly commented; this is something we have tried to fix where we can but we have only touched a small section of the existing architecture.

[9] Maven - The Android Studio package manager.
[10] We think this is inadvisable.

Android can place limitations on how code is written. This is not a significant issue but it tends to rear its head in strange ways; for example, in the current codebase, variables have to be global to the Android Activity to be output on screen. Android versioning may start to cause greater problems in the years to come as the Google Nexus 5 recedes further into obselescence. However, we can say with confidence that the greatest issues are due to the generational nature of the codebase.

AntBot has been in continuous use for the three years since its development by four different students across five projects [2, 9, 19, 6] (and this work). Different coding styles and standards have been put into practice throughout; in addition to the lack of documentation and comments, the differing styles can make some sections exceedingly difficult to work with when required. As two examples we give the ocular calibration and framerate bug; in both cases, a large proportion of the time dedicated to the issue was spent attempting to figure out what legacy code was doing and if it was relevant to the issues being resolved; once function and relevance had been identified, a solution which fit into the AntEye application without breaking existing code had to be designed. This view to design can result in poor software engineering practices, further compounding the readability issue with code which then only makes sense in the context of the AntEye application and the operational procedures of the AntBot.

To the best of our knowledge of the history of the platform (and the codebase itself) the original application structure (Section 3) was abandoned in the year after its creation. This means we have a codebase which has never been used as it was originally designed, resulting in what little documentation there is on the application structure being mostly incorrect. As another concrete example, previous works [2, 9, 19] list a serial message which sends encoder values from the Arduino to the smartphone, however they fail to mention that it was used only as part of a particular Arduino function[11] and furthermore, that this functionality has since been removed[12].

It is with this justification (including the hardware/firmware limitations mentioned above) that we recommend the AntBot be retired and a new platform be constructed. We present some broad suggestions based on the problems and limitations we have experienced with AntBot. A small omni-wheeled design capable of holonomic motion would be useful in testing the bio-inspired models with greater rigour; a smaller platform could make better use of available space, and holonomic capabilities would allow holonomic properties of the models to be tested (e.g. the TN and Pontine neurons in the CX). A singular computational unit, capable of interfacing with a motor board (e.g. a Raspberry Pi) would eliminate the control and feedback latency problems experienced on the current platform; this would also allow a redundant sensor array to be added and used to provide some baseline or algorithmic performance against which biological models can be compared. We feel there is something to be said also for using a general purpose OS, instead of a mobile platform. We feel this presents a shallow learning curve compared to the Android ecosystem and it may also allow more freedom in application design.

[11]Confirmed by observation and correspondence with *Eberding*.

[12]This removal is thought to be part of [9], version control records should be available to confirm.

7.2.3 Closing

With all of this said, we have to note that our predecessors have been extraordinarily helpful in working with the codebase. In particular, *Zhang* and *Eberding* were of great help in establishing an early understanding of the codebase, robot operation, and in confirming some elusive technical details. We also intend to make ourselves available for any questions we can answer about the system. Though our results show promise for the proposed model, the platform has limited the scope of our investigation. We feel the platform is no longer fit-for-purpose in future testing of bio-inspired systems.

AntBot has been worked hard in the years since its creation. It deserves some rest.

8 References

[1] Paul Ardin, Fei Peng, Michael Mangan, Konstantinos Lagogiannis, and Barbara Webb. Using an insect mushroom body circuit to encode route memory in complex natural environments. *PLOS Computational Biology*, 12(2):1–22, 02 2016.

[2] Leonard Eberding. Development and Testing of an Android-Application-Network based on the Navigational Toolkit of Desert Ants to control a Rover using Visual Navigation and Route Following., 2016.

[3] Pauline Nikola Fleischmann, Robin Grob, Valentin Leander Mller, Rdiger Wehner, and Wolfgang Rssler. The geomagnetic field is a compass cue in cataglyphis ant navigation. *Current Biology*, 28(9):1440 – 1444.e2, 2018.

[4] Robin Grob, Pauline N Fleischmann, Kornelia Grübel, Rüdiger Wehner, and Wolfgang Rössler. The role of celestial compass information in cataglyphis ants during learning walks and for neuroplasticity in the central complex and mushroom bodies. *Frontiers in behavioral neuroscience*, 11:226, 2017.

[5] Thomas Haferlach, Jan Wessnitzer, Michael Mangan, and Barbara Webb. Evolving a neural model of insect path integration. *Adaptive Behavior*, 15(3):273–287, 2007.

[6] Robert Mitchell. Developing AntBot: Visual Navigation based on the insect brain, 2018.

[7] Peter O'Donovan. Optical flow: Techniques and applications. 2005.

[8] Keram Pfeiffer and Uwe Homberg. Organization and functional roles of the central complex in the insect brain. *Annual review of entomology*, 59:165–184, 2014.

[9] Luca Scimeca. AntBot: A biologically inspired approach to Path Integration, 2017.

[10] Julien R. Serres and Franck Ruffier. Optic flow-based collision-free strategies: From insects to robots. *Arthropod Structure & Development*, 46(5):703 – 717, 2017. From Insects to Robots.

[11] Kahlouche Souhila and Achour Karim. Optical flow based robot obstacle avoidance. *International Journal of Advanced Robotic Systems*, 4(1):2, 2007.

[12] Finlay J. Stewart, Dean A. Baker, and Barbara Webb. A model of visual–olfactory integration for odour localisation in free-flying fruit flies. *Journal of Experimental Biology*, 213(11):1886–1900, 2010.

[13] Thomas Stone, Barbara Webb, Andrea Adden, Nicolai Ben Weddig, Anna Honkanen, Rachel Templin, William Wcislo, Luca Scimeca, Eric Warrant, and Stanley Heinze. An anatomically constrained model for path integration in the bee brain. *Current Biology*, 27(20):3069–3085, 2017.

[14] Massimo Tistarelli, Enrico Grosso, and Giulio Sandini. Dynamic stereo in visual navigation. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 186–193. IEEE, 1991.

[15] N. van der Stap, R. Reilink, S. Misra, I. A. M. J. Broeders, and F. van der Heijden. The use of the focus of expansion for automated steering of flexible endoscopes, June 2012.

[16] Barbara Webb. The internal maps of insects. *Journal of Experimental Biology*, 222(Suppl 1), 2019.

[17] Rüdiger Wehner. The architecture of the desert ant's navigational toolkit (hymenoptera: Formicidae). *Myrmecological News*, 12:85–96, 09 2009.

[18] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, Jun 1994.

[19] Zhaoyu Zhang. Developing AntBot: a mobile-phone powered autonomous robot based on the insect brain, 2017.

A Genetic Algorithms

Genetic Algorithms (GAs) are metaheuristic optimisation algorithms which aim to optimise some (arbitrary) objective function. GAs perform this optimisation by taking inspiration from the theory of evolution by natural selection wherein species advance by genetic mixing (breeding) and a small chance of mutation with each offspring. The theory of evolution by natural selection suggests that mutations which are useful to the species become more potent over multiple generations as they allow the individual to live longer and/or reproduce more than those without the mutation; mutations which are useless die out or remain inert, and those which are harmful tend not to survive. This is an extremely simplified view.

With appropriate representation, this evolutionary concept can be translated into an optimisation algorithm. Say we choose to represent our data as 4-bit binary strings. To start with, we generate some population of n random 4-bit bitstrings in the set range; two solutions are picked using some selection process, usually based on *fitness*[13] and "breed" with probability p_c; the offspring are then mutated with (very small) probability p_m; finally the offspring/mutant-offspring/original bitstrings [14] form the next generation and the process repeats until some termination criterion is satisfied; this could be time, number of generations, stagnation of the population, etc.

This short explanation is merely to provide a high level idea of the concepts employed when working with GAs; there is a great deal of depth and nuance not captured by (and not necessary for) this paper. A tutorial by *Whitley* can be found in *Statistics and Computing, Volume 4 (1994)* [18].

[13]Fitness - some measure of how *good* the solution is - e.g., minimisation would mean a lower value is fitter.

[14]Note that those chosen may not breed with probability $p_c - 1$, and similarly, they may not be mutated with probability $p_m - 1$

B CX VICON Recordings

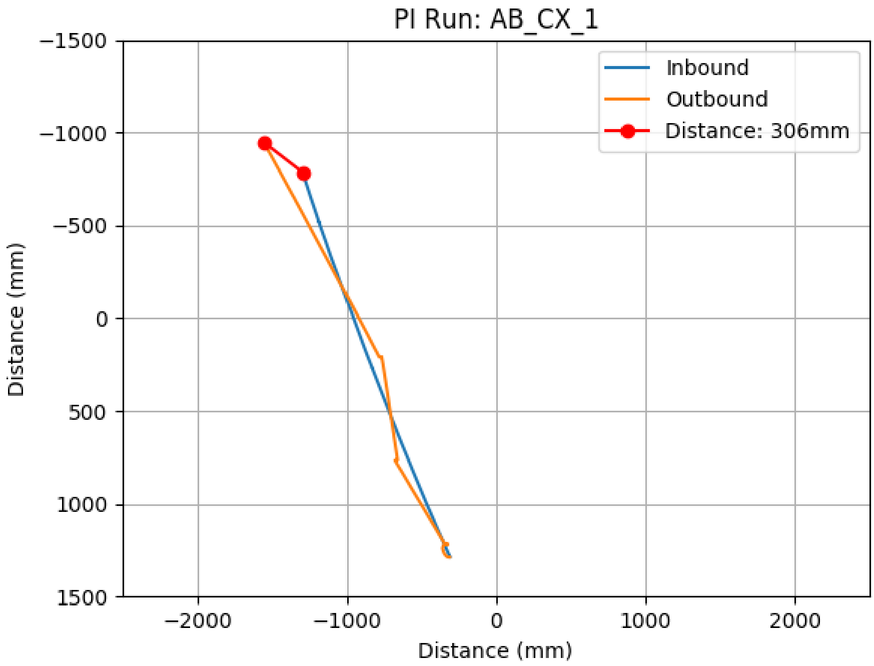


Figure 22: PI test AB_CX_1

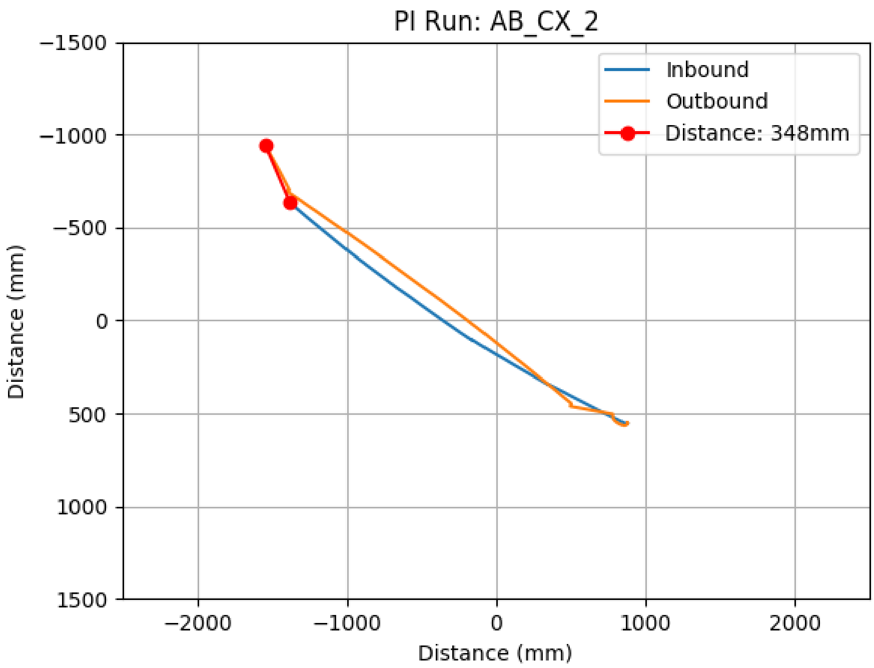


Figure 23: PI test AB_CX_2

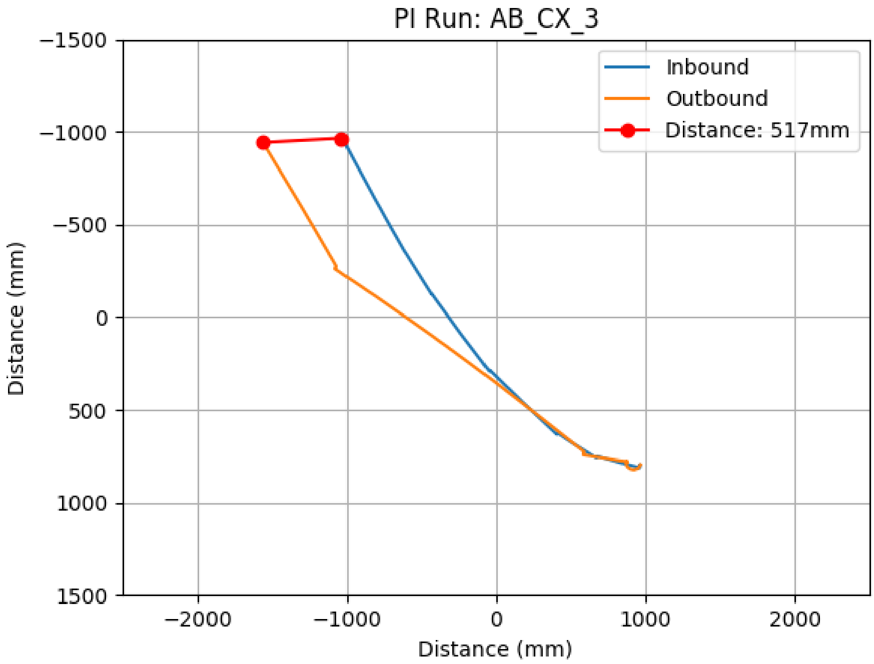


Figure 24: PI test AB_CX_3

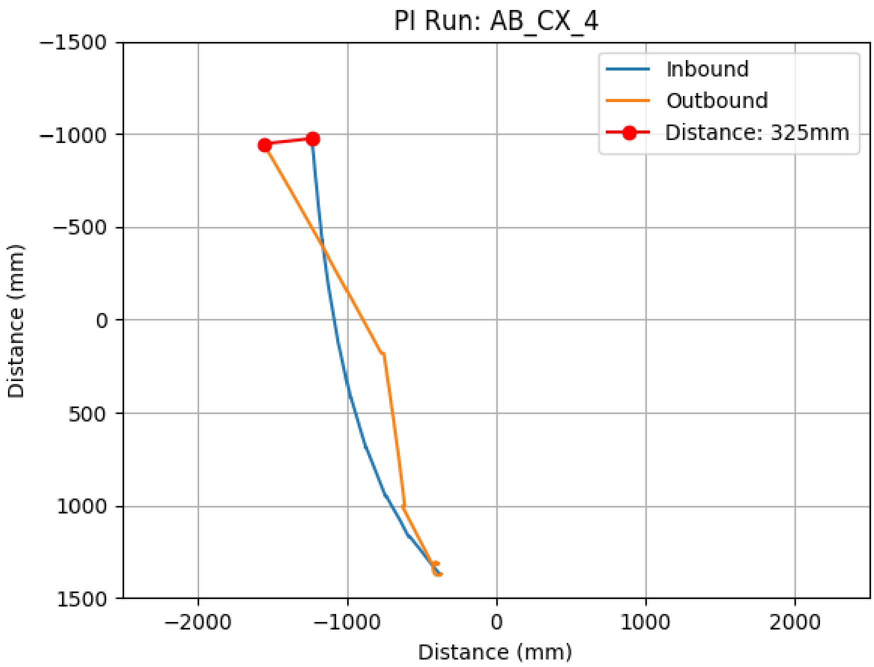


Figure 25: PI test AB_CX_4

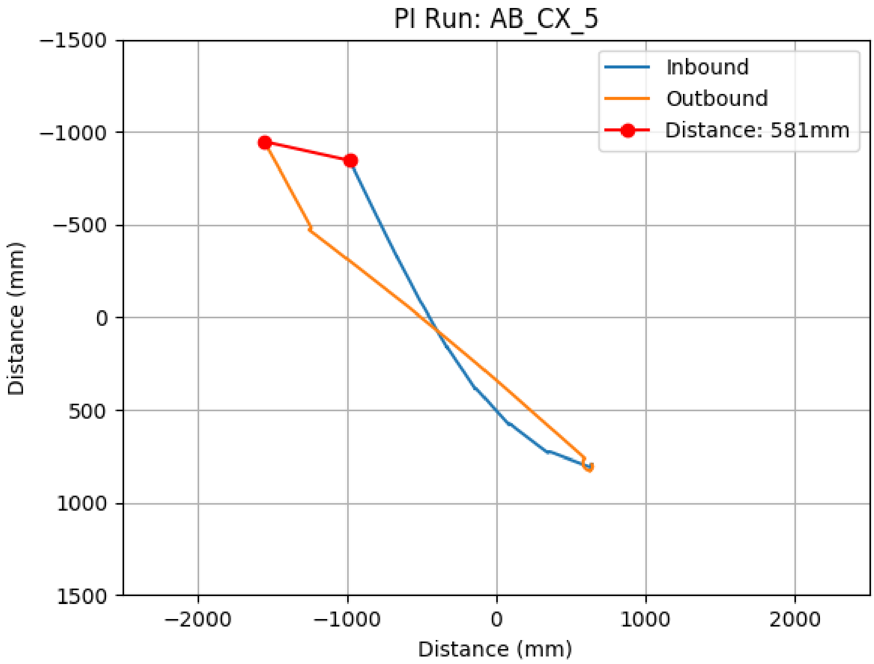


Figure 26: PI test AB_CX_5

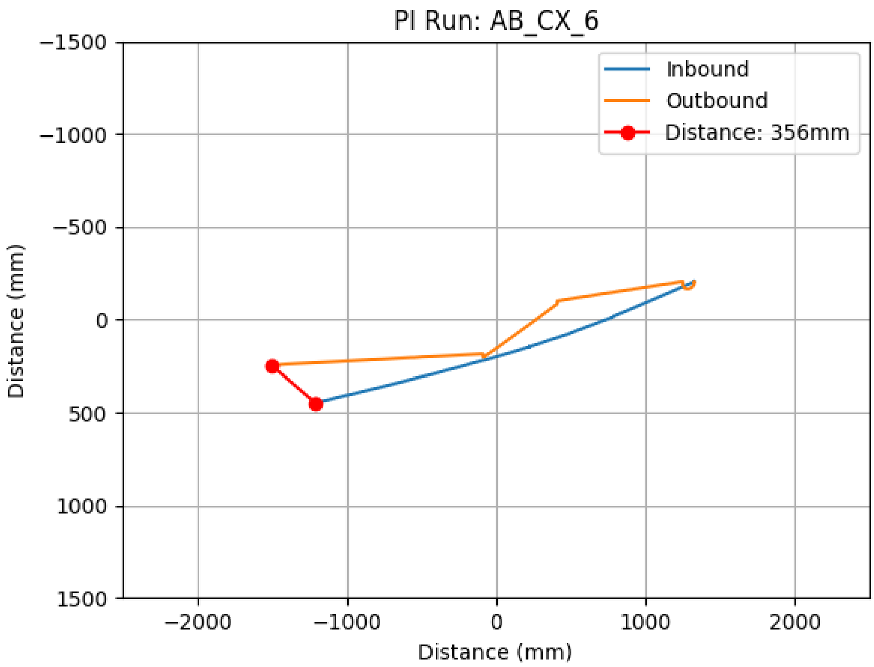


Figure 27: PI test AB_CX_6

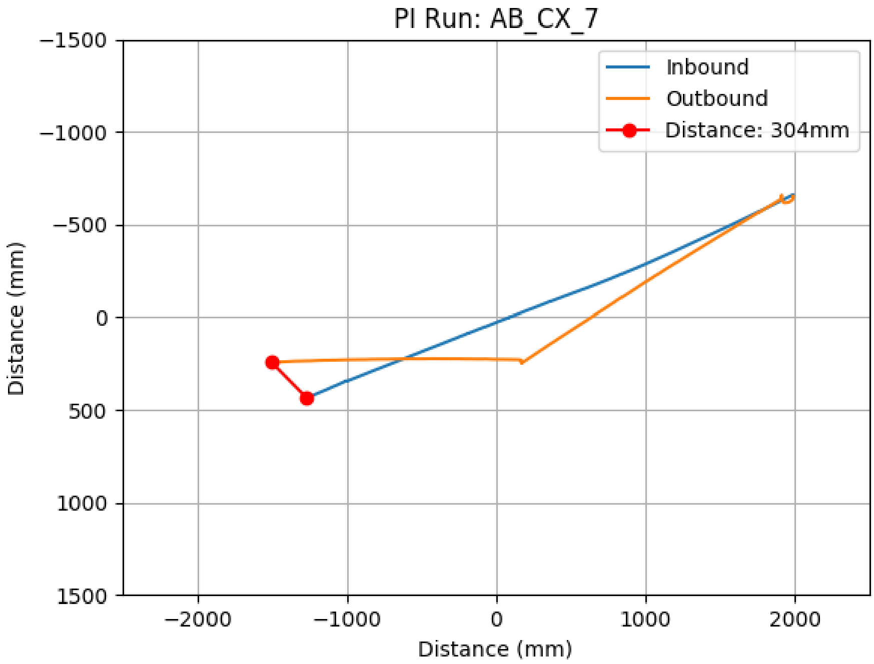


Figure 28: PI test AB_CX_7

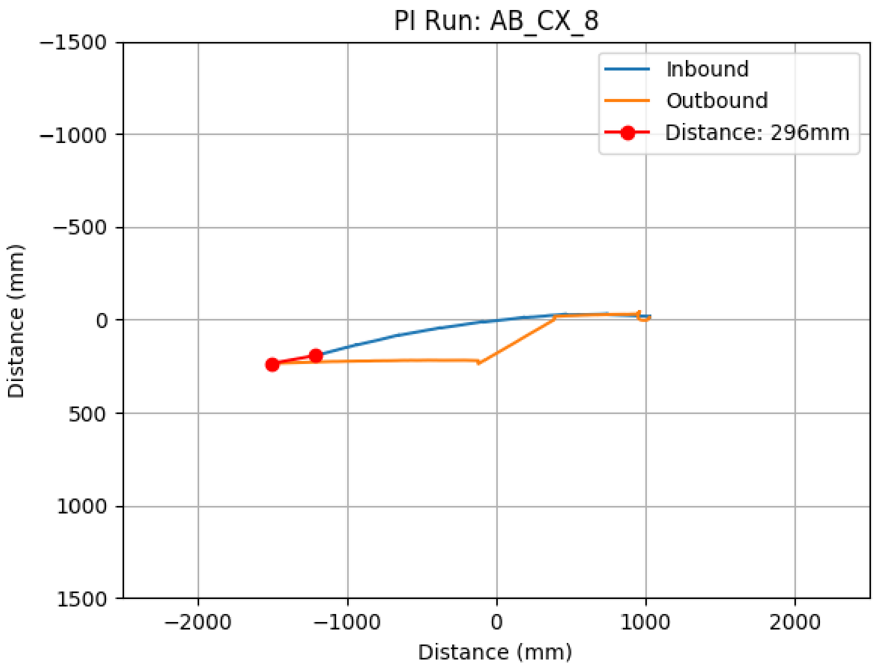


Figure 29: PI test AB_CX_8

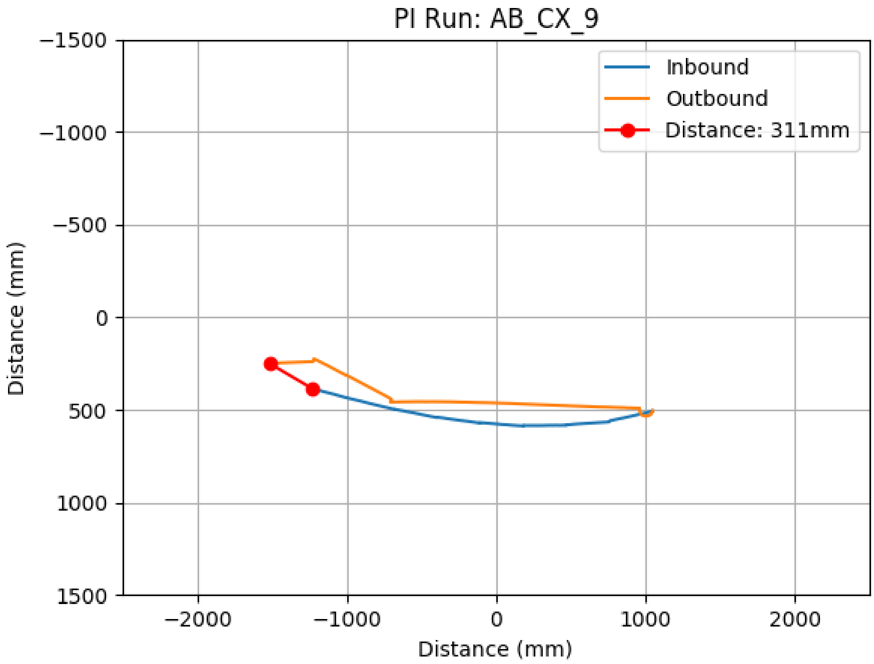


Figure 30: PI test AB_CX_9

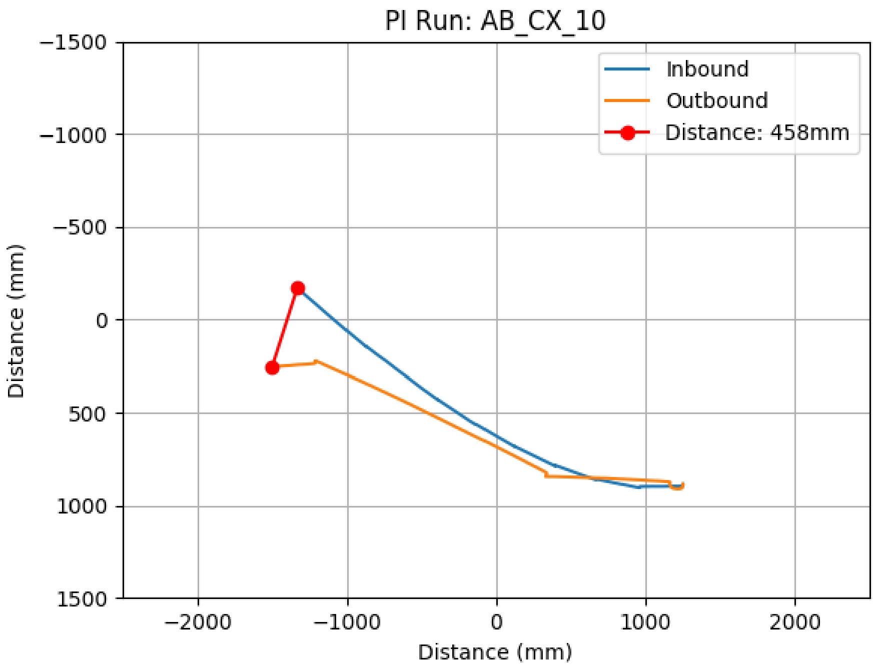


Figure 31: PI test AB_CX_10