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
March 30, 2019

Supervised by
Dr. Barbara Webb

Acknowledgements

Pending . . .

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

Pending . . .

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 huge 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 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, is not biologically infeasible, 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) [14], and Optical Flow Collision Avoidance (OFCA) [7] for each task at a low level, then combining their outputs to get a form of higher visual processing (similar to the weighted "base model" described in [17][1]). 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 are known to be physiologically plausible. [1, 14, 7].

This project primarily extends the work done in [7]. As such we continue using the AntBot platform; a robot constructed for the express purpose of experimenting with the algorithms in the *Ant Navigational Toolkit* [3, 18].

1.1 Motivation

Currently, a full base model for insect navigation does not exist [17]. 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 [10, 7, 3, 20], and a model combining the two has also been attempted by *Zhang* in [20]. This is used as an inspiration and will be discussed further in Section 2.4.

The previous AntBot implementations have demonstrated good performance of the CX and MB models individually [10, 7]. Performance of a combined system has also been shown to be reasonable, however, it is less consistent than we would desire [20]. In the combined model tests from [20] we note two key methodological problems: A fixed outbound route, and fixed component weightings. We address the former by adding the OFCA component to our model; as in [7], the AntBot will follow a non-deterministic outbound route through a cluttered arena. The latter brings up the more complicated question of plausible synaptic plasticity which, while undoubtedly interesting, lies outside the scope of this project (though some modelling of plasticity will be included). It is worth noting that here may also have been unknown tecnical issues with the robot which affected the results of [20], making them less consistent than they should be in reality (see [7] and later in this work).

[1] [DRAFT] This is the review paper that you sent me in prepreint. So far as I can tell, it has not yet been published; will it be out by April? How should I refer to this if the paper has not been published?

While [7] provides a reasonable collision avoidance system based on optical flow, it does not fit so neatly into the ECX model. We therefore aim to explore an alternative, yet still biologically plausible collision avoidance system which will fit into the ECX model.

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

1.2 Practical Goals

We aim to build upon the experimental scenario from [7]. The robot will be tested by allowing it to navigate through a cluttered environment using a collision avoidance system. The navigational systems will then be tasked with bringing the robot home through the cluttered environment using a combination of visual information, a path integration vector, and collision avoidance.

In order to achieve this experimental goal, we break the project down into four components:

1. The first stage will involve solving some technical issues picked up by [7]; making any hardware/software adjustments required to provide a solid foundation on which to develop.

2. We can then begin investigating existing systems and establishing experimental metrics. This stage will involve research and review of new topics (the main one being the Central Complex model for Path Integration), and their implementations on the robot (if they exist). This stage will look to establish appropriate metrics by testing the existing CX model in a non-deterministic navigational task (see Section 4).

3. This stage will involve the set up and testing of the individual components of the ECX model. Building the modified optical flow system, adapting the work from *Zhang* to combine the MB model with the CX, and finally, putting the three pieces together.

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. [3, 10, 20, 7].

Significant contributions of this project:

1. Research and installation of a new compass sensor for the AntBot control systems.[2]

2. A major code refactor has taken place to make AntBot more usable for this project, and make it more accessible for future students.

[2] [DRAFT] practically, its looking like this will be less and less useful, I've not had a chance to actually try re-writing the control code to use the new sensor. That said, time went into this early on, and it would be nice to include this.

3. Addition of a calibration system to allow the user to auto-detect the position of the camera lens attachment (see Section 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. Results definitively showing the impracticality of a focus of expansion based system for collision avoidance and justifying continued use of matched filters.
7. Identified and fixed a bug in the visual preprocessing system to raise the framerate from 2fps to approximately 10fps under normal usage.
8. Construction of a neural collision avoidance model based on the matched-filter system from [7, 13].
9. Implementation of a biologically plausible "base model" for insect navigation, the Extended Central Complex (ECX) model.
10. Justification for the recommendation that the AntBot be retired at the earliest opportunity.

2 Background

This project builds directly upon [7]. We first provide a review of the relevant background topics from that paper, before developing the relevant ideas further for this project. This will be a very brief summary of the work that took place and any relevant results or new perspectives. For more detail, please consult [7].

2.1 Optical Flow for Collision Avoidance

Optical flow is a large and diverse area of study. As such, we will not provide a complete background on the fundamental principles. Relevant terms will be explained, however, a succinct background of the concepts necessary for this paper can be found in [7]. A comprehensive introduction is given by *O'Donovan* in [8].

The main driving point in this paper is the integration of multiple navigational systems into a single model, namely, the Central Complex. Optical flow flitering worked well for a standalone collision avoidance system, however, it does not fit so neatly into the CX model. We therefore require a different approach. The relevant background revisits the concept of the *focus of expansion* (FOE) from [7, 8]. The FOE is the point from which all optical flow vectors originate. The location of the FOE can tell us things about the motion, and depth of the image.

In [7], the FOE was used only to compute time-to-contact with an obstacle. In this work, we instead use it directly to determine the potential location of an obstacle. As stated in [2] [3], computing the FOE is not trivial. Following [7] 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. This makes the problem more difficult; the basic from-flow method for computing the FOE given by [8] is computationally complex for a dense flow field [7]. The time taken to compute may become prohibitive. We will therefore look at using a *subset*[4] (see Section 4) optical flow field for the method provided by *O'Donovan*.
We will also revisit the matched-filter collision avoidance system from [7], though this system remains wholly unchanged at its core. For details, please see [7].

2.1.1 The Least Squared-Error method

This is the method given by *O'Donovan* and discussed in [7]. The FOE is computed simply as:

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

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

[3] [DRAFT] This is the source for the 'fuzzy' FOE; not included in this submission but I may revisit it, time-allowing; I have most of the section written

[4] A subset of flow vectors computed as part of a dense optical flow field.

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 more or less copied from [7] for the benefit of the reader. For more details, the reader should consult [7], or [8].

This method for estimating the focus of expansion was originally given by *Tistarelli et al.* in their paper *Dynamic Stereo in Visual Navigation*[15, 8], and it serves as an excellent example for the reason the FOE is so difficult to compute. In theory, we should be able to take any two vectors **u**, **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[8]. While wonderfully simple, 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 try to fit the best FOE to the flow field given.

2.2 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.* is given here in Figure 1. In the implementation by *Ardin et al.* and previous AntBot implementations, captured images were converted to greyscale and downsampled [1, 3, 20, 7]. 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 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 learing process; thus, the lower the MBON output, the more familiar the image (and vice versa). Different route following strategies have been implemented (scanning [1, 3, 20], Klinokinesis [20], combination with the CX model [20](see below), and visual scanning [7]); but, most commonly, some form of scanning is used whereby the agent scans an arc for the most familiar direction.

This has been a brief explanation for context. Part 1 gives slightly deeper insight[7] (particularly in relation to AntBot projects), and of course the work of *Ardin et al.* can give the full details [1].

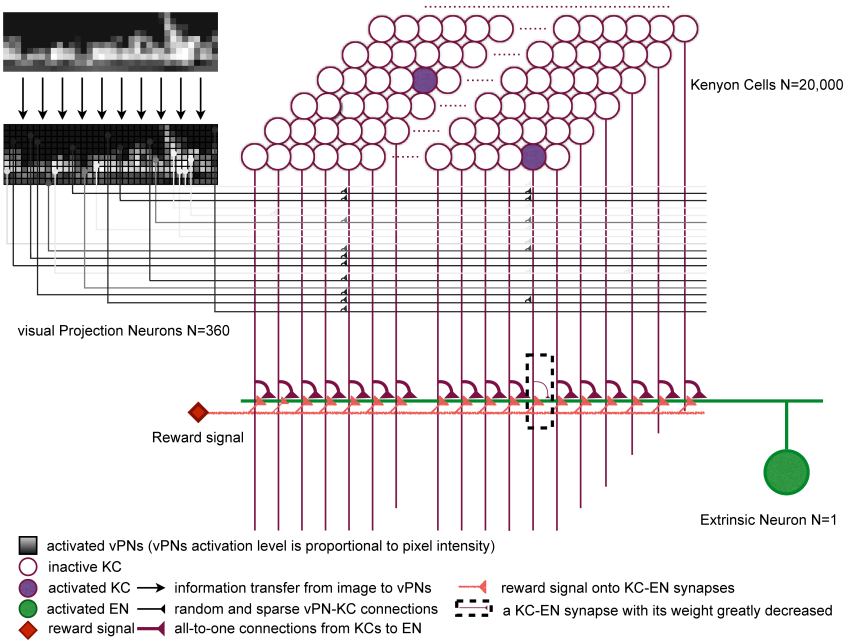

Figure 1: The Mushroom Body circuit: (Caption from *Ardin et al.*, Figure 2; note, their description and figure uses "EN" instead of "MBON"): Images (see Fig 1) activate the visual projection neurons (vPNs). Each Kenyon cell (KC) receives input from 10 (random) vPNs and exceeds firing threshold only 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.

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 [3, 20, 7] differs only in the number of PNs present (900 vPNs are used in all three works), and the KC modelling (*Ardin et al.* use a spiking model[5] for the KCs [1]). More relevant to this work, is the proposed modification given by *Zhang*, whereby, 8 MBONs are present as opposed to 1 (see Section 2.4).

2.3 The Central Complex for Path Integration

The Central Complex (CX) is a highly conserved structure present in the insect brain[9, 14]. Though the finer structural details and component position may vary, the basic composition is more or less the same across species [9]. Organization and function of the various parts of the CX are given by *Pfeiffer and Homberg* in [9].

We instead 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 structure drawn purely from 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 [6]. We present this also, purely for the sake of interest.

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

2.3.1 The Evolved Model for Path Integration

Prior to the CX model described below, there were several candidate neural networks proposed for PI in ants [6]. *Haferlach et al.* report the two best-known candidates being hand-designed, with more recent research into an evolutionary approach to network design [6]. The approach presented in [6] follows this trend and employs a Genetic Algorithm (GA) (see Appendix A for a brief overview) to evolve a neural model which is suitable for PI with biologically plausible inputs.

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) [6]. An example encoding from [6] can be seen in Figure 2.

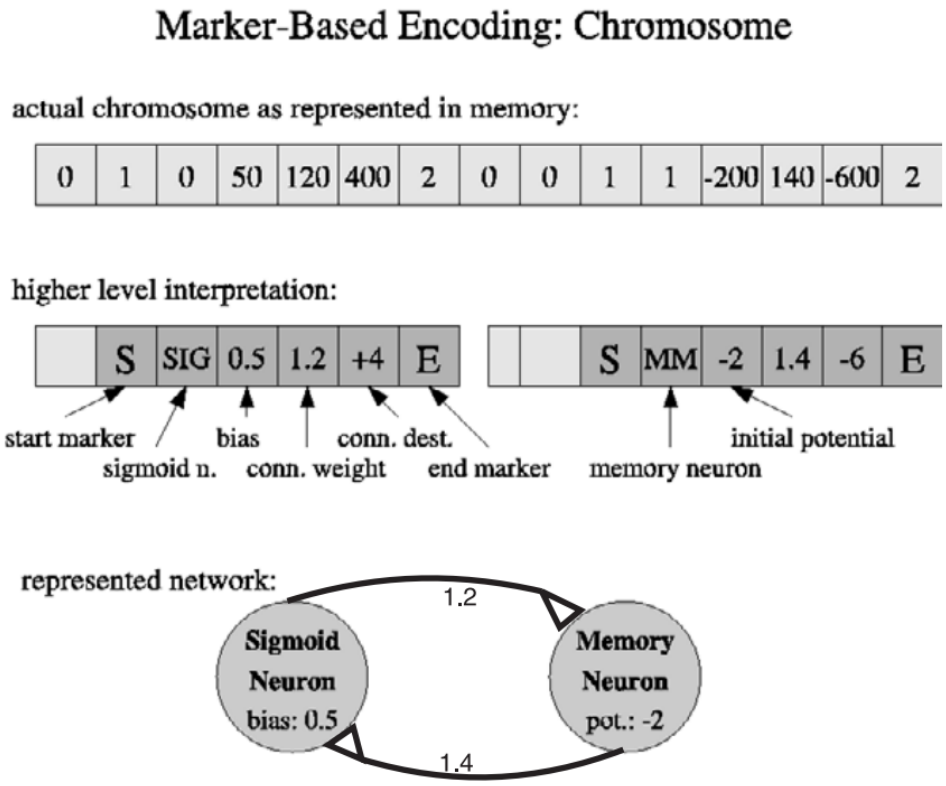

Figure 2: (Figure 2 from [6]) The Marker-Based encoding scheme used by *Haferlach et al.*, demonstrating how a simple neural model would be encoded as a sequence of integers.

Localised *tournament selection*[6] is used to pick solution 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) [6]. The solution with higher fitness wins the tournament and progresses to mutation and *crossover*[7] This localised selection allows

[6]Tournament Selection - Two solutions are picked randomly, then their fitness compared to determine if they will be selected

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

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 is evaluated until some maximum time limit m is reached, and the distane to the start point $dist_t$ is computed at each time step. The fitness is then the inverse sum of squared distances [6]:

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

The fitness is then augmented to penalise homeward routes which spiral (circling the start point, getting closer with each pass), and also to reward networks which are simple in structure; shown in Equations 3 and 4 respectively[8].

$$f = \frac{1}{\sqrt{\int_0^m dist_t^2 |\theta - \omega| dt}} \tag{3}$$

$$f = \frac{1}{(1+k_n)^{c_n}(1+k_c)^{c_c}\sqrt{\int_0^m dist_t^2 |\theta - \omega| dt}} \tag{4}$$

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) [6]. These are applied as a two-stage evolutionary process; evolve a solution first using Equation 3 to compute the fitness, then use this solution to seed a new population and evolve a solution from here using Equation 4. The two-stage evolutionary processalone 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 [6].

Haferlach et al. present the model shown in Figure 3; notice the evolved structure bears a striking resemblance to the CX model presented by *Stone et al.* (Figure 4). Indeed, this network operates on the same principle; encoding a heading vector across multiple neurons with directional preference. 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) [6].

2.3.2 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

[8] [DRAFT] I didn't have time to fix the equation formatting (equation number bumped out of position). This will be sorted in the final version.

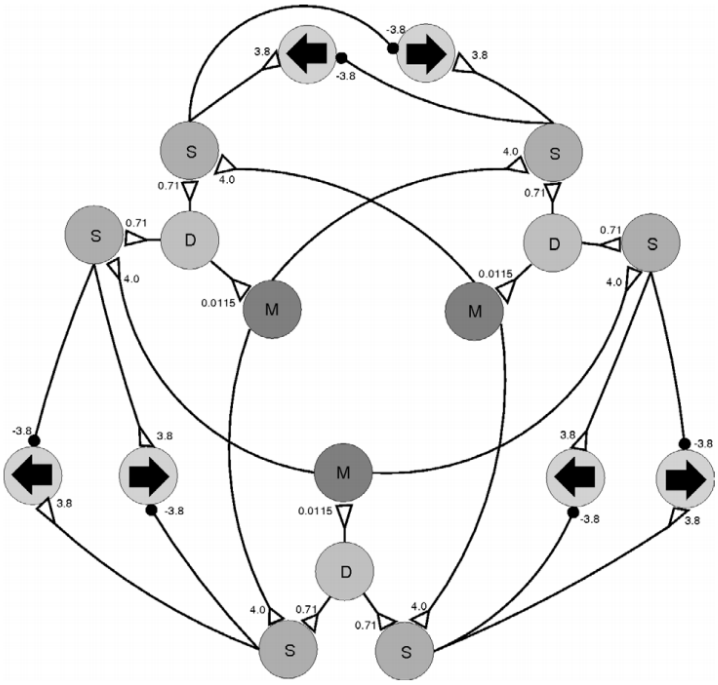

Figure 3: (Figure 3 from [6]) The high-fitness network, consistently evolved using the constrained two-stage evolution process illustrated by *Haferlach et al.*

(PI) both in simulation and on the AntBot platform [10, 14]. The model presented is shown in Figure 4. Splitting the model into its six layers, we get a breakdown of functionality:

- Layer 1: Heading preprocessing (TL), Speed (TN)
- Layer 2: Heading preprocessing (CL1)
- Layer 3: Heading (TB1)
- Layer 4: Memory (CPU4)
- Layer 5: Normalisation and Inhibition (Pontine Neurons)
- Layer 6: Steering/output (CPU1)

Figure 4 shows four types of neuron: TN (Tangential Neuron), TB1 (green), CPU4 (yellow and orange), and CPU1 (dark blue, light blue, and purple).

While Figure 4 (Left) shows a distinction 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 [14]; thus, the distinction makes no difference in the model. Similarly the normalisation and inhibition function 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 also; in our case, the Pontine Neurons would have the

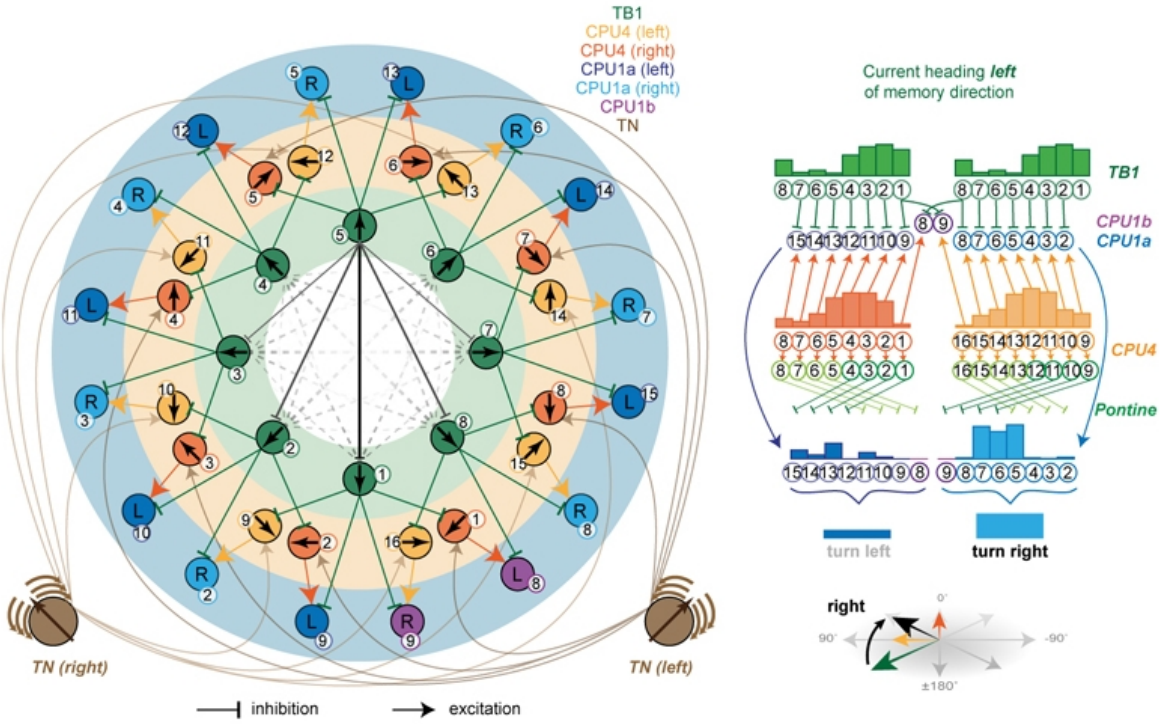

Figure 4: The Central Complex model presented by *Stone et al.*. (Left) This graph demonstrates the basic structure of the CX model (Figure 5G from [14]). Pontine neurons have been excluded for clarity. (Right) This graph shows how signals propogate through the network where the current heading lies to the left of the desired heading, i.e. a right turn should be generated (Figure 5I from [14]). The numbers given at each layer on the right correspond to the numbers given for each neuron in the graph on the left.

same activity patterns as the CPU4 neurons [14]. Figure 4 (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) [14]. We also add implementation information for the AntBot where appropriate; occassionally the AntBot implementations are slightly different.

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

$$r = \frac{1}{1 + e^{-(aI-b)}} \tag{5}$$

where a and b are parameters which control the slope and offset of the sigmoid function [14]. Optionally, Gaussian noise may be added to the output. The input to each neuron is a weighted sum of the activity of each neuron that synapses onto it; say neuron j, the input is:

$$I_j = \sum_i w_{ij} \cdot r_i \tag{6}$$

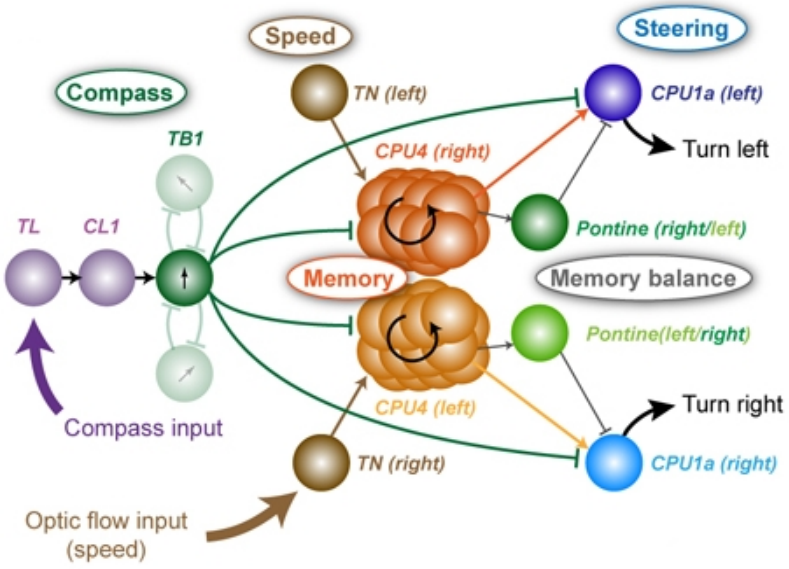

Figure 5: Here we can see the layers of the CX model and how they fit together. A heading signal is input to the TL neurons, propogating through the CL layer to TB1 (heading ring-attractor) and CPU4 (memory). TN neurons (speed sensitivity) input directly to CPU4. So, the combination of heading and speed 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.

The weights used by *Stone et al.* can only be 0, 1, or -1 for no-connection, excitatory, or inhibitory respectively [14].

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 direction from vision. 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 [4, 5]. The TL neurons have been shown to be polarisation sensiive across multiple insect species [14] (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. $\theta_{TL} \in \{0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°\}$) [14]. Together, the 16 TL neurons encode the heading of the agent in a single timestep; each neuron receives input activation as:

$$I_{TL} = cos(\theta_{TL} - \theta_h) \tag{7}$$

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 [14]. *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 [20, 10]. 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 [14]. 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 short, 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 [14]. *Stone et al.* give the speed calculation from the TN neurons as:

$$I_{TN_L} = [sin(\theta_h + \phi_{TN}) \quad cos(\theta_h + \phi_{TN})]\mathbf{v} \tag{8}$$

$$I_{TN_R} = [sin(\theta_h - \phi_{TN}) \quad cos(\theta_h - \phi_{TN})]\mathbf{v} \tag{9}$$

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 [14].

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 is 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* [10] 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 4) forming a *ring attractor*[14]. The weighting for an arbitrary inhibitory connection from neuron i to neuron j is given by:

$$w_{ij} = \frac{cos(\theta_{TB1,i} - \theta_{TB1,j}) - 1}{2} \tag{10}$$

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:

$$I_{TB1_j^{(t)}} = (1 - c) \cdot r_{CL1_j}^{(t)} + c \cdot \sum_{i=1}^{8} w_{ij} \cdot r_{CL1_j}^{(t-1)} \tag{11}$$

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 [14]. The CPU4 neurons accumulate distance with direction. The input the CPU4 neurons is given by:

$$I_{CPU4}^{(t)} = I_{CPU4}^{(t-1)} + h \cdot (r_{TN2}^{(t)} - r_{TB1}^{(t)} - k) \tag{12}$$

where $h = 0.0025$ determines the rate of memory accumulation, and $k = 0.1$ is a uniform rate of memory decay [14]. All memory cells are initialised to $I_{CPU4}^{(0)} = 0.5$ and are clipped at each timestep to fall between 0 and 1 [14]. As shown in Figure 4, each TB1 provides input to two CPU4 neuron, 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) [14]. Interestingly, *Zhang* showed that the network can be initialised to an arbitrary state, allowing the agent to navigate along arbitrary vectors [20]. 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 [17] (see Section 2.4).

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:

$$I_{CPU4}^{(t)} = I_{CPU4}^{(t-1)} + s \cdot ((1 - r_{TB1}^{(t)}) \cdot g - l) \tag{13}$$

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 12 as the TN neurons are present in the model.

Pontine: The pontine neurons project contralaterally connecting opposite CBU columns (shown in Figure 4 (Right)). The 16 pontine neurons each receive input from one CPU4 column [14]; pontine input can be given simply as:

$$I_{Pontine}^{(t)} = r_{CPU4}^{(t)} \tag{14}$$

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 4). 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:

$$I_{CPU1}^{(t)} = r_{CPU4}^{(t)} - r_{Pontine}^{(t)} - r_{TB1}^{(t)} \tag{15}$$

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 $I^{(t)}_{CPU1} = r^{(t)}_{CPU4} - r^{(t)}_{TB1}$).

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

$$\theta^{(t)}_h = \theta^{(t-1)}_h + m \cdot (\sum_{i=1}^{8} r_{CPU1R_i} - \sum_{i=1}^{8} r_{CPU1L_i}) \tag{16}$$

where $m = 0.5$ is a constant [14]. The angle computation is the same on the AntBot.

There are some details which have been omitted for clarity. The stack presented 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 [14] (the STAR Methods section contains the full technical details of the model).

2.4 The Eight MBON Model (CXMB)

In [20], 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 2.3).

Image memory now has 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). This process is visualised in Figure 6.

While, in theory, this eight MBON model could function independently, *Zhang* uses it to (rather neatly) augment the CX 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 [20].

The MB circuit is shown an image in the usual way; 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:

$$\bar{M_i} = \frac{M_i}{\Sigma^8_{k=0} M_k} \tag{17}$$

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):

$$\bar{M_i}^{-1} = 1 - \bar{M_i} \tag{18}$$

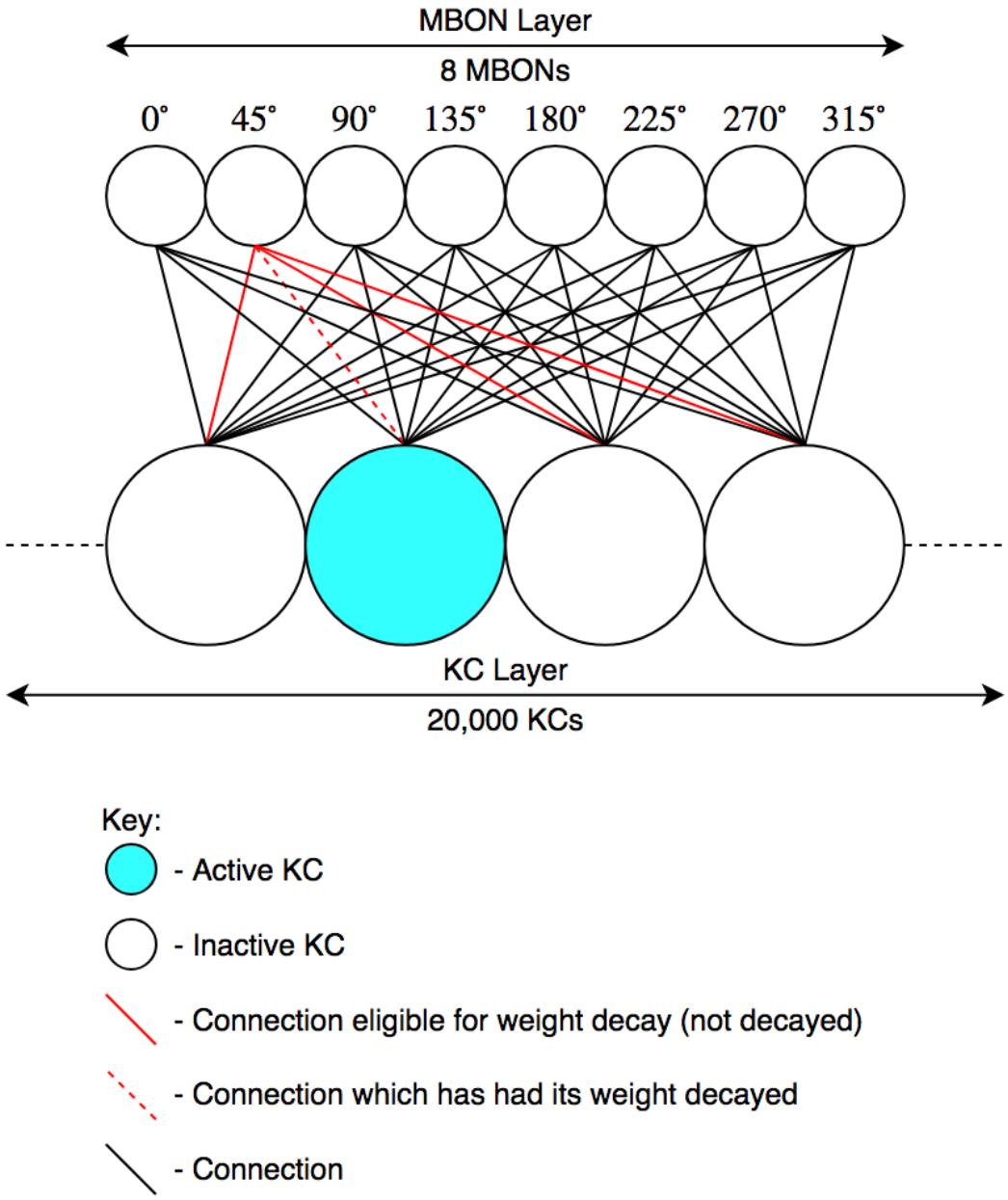

Figure 6: Our interpretation of the eight MBON model proposed by *Zhang*. Every KC connects to every MBON. All connection weights start out at $w = 1$. Following the example presented in the text, if an image being learned corresponds to facing a direction of 45°, then only the conncetions to that MBON (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).

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:

$$CPU1_{output} = k \cdot W_{CPU4} \cdot CPU4 + (1 - k) \cdot W_{MBON} \cdot \bar{M}^{-1} \tag{19}$$

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 [20]), $\bar{M}^{-1}$ is the collection of all inverse normalised MBON responses, W_{CPU4} is a custom matrix[9], W_{MBON} is an identity matrix

[9]This is the term used by *Zhang* to describe the W_{CPU4} matrix. More specifically, this matrix describes the

that will expand the MBON response array from 8x1 to 16x1 [20].

In oreder 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 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 [17]. Indeed, this concept of vector memory is shown to be quite a useful tool in insect navigation [17]. It is worth noting that, the precise biological mechanism employed by insects to perform this task, is unknown at time of writing. It would be neat to reuse the CX circuitry, but this is an open question.

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[20]. It should also be noted that the average performance of both models in *Zhang's* work was good [20].

We note two methodological problems with *Zhang's* work: The outbound route, while randomly chosen, was the same accross 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.

2.5 Review of Part 1

The work in [7] specifically covers the Mushroom Body and Optical Flow Collision Avoidance. The Mushroom Body model used in [7] is the same as that used by *Ardin et al.* in [1], except for the small differences noted in Section 2.2. The model tested in [7] 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.

connections between the CPU4 and CPU1 layers of the CX model.

Multiple OFCA systems were tested, however, in final experiments an optical flow filtering system is used. In short, a pattern of expected motion is created, then the actual observed motion is projected on top of it. An absolute difference is computed between the two and this can tell us if part of the image is moving faster than we expect. This can be used to detect obstacles. This strategy proved simple, but effective. While we move away from it for the bulk of 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 a small 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 [7].

3.1 Hardware

The robot is composed of three main parts: a Dangu Rover 5 chassis, an Arduino, 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, and 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.
Additionally, the robot possesses a 360° camera attachment, giving the robot a panoramic view to match that of the desert ant[10]. The robot was also modified as part of [7] 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). The Arduino code can be thought of as the firmware.

3.2.1 Android

Android can provide a nice basis for working on robotics projects. The Operating System permts 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. Unfortunately, in recent years, this structure was abandoned and almost all code is contained within the *AntEye* application. 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.

There are a lot of criticisms we can raise about the robot (in both a hardware and software sense); ultimately the key problem may be lack of developer familiarity with

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

Figure 7: 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.

Figure 8: The Kogeto Dot 360° panoramic lens attachment.

Figure 9: A sample view of the lens from the front facing camera, before any processing.

the Android platform. While Android seems to work well at first glance, it can ultimately be quite restrictive; we leave this to the later discussion.

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 [3] 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.

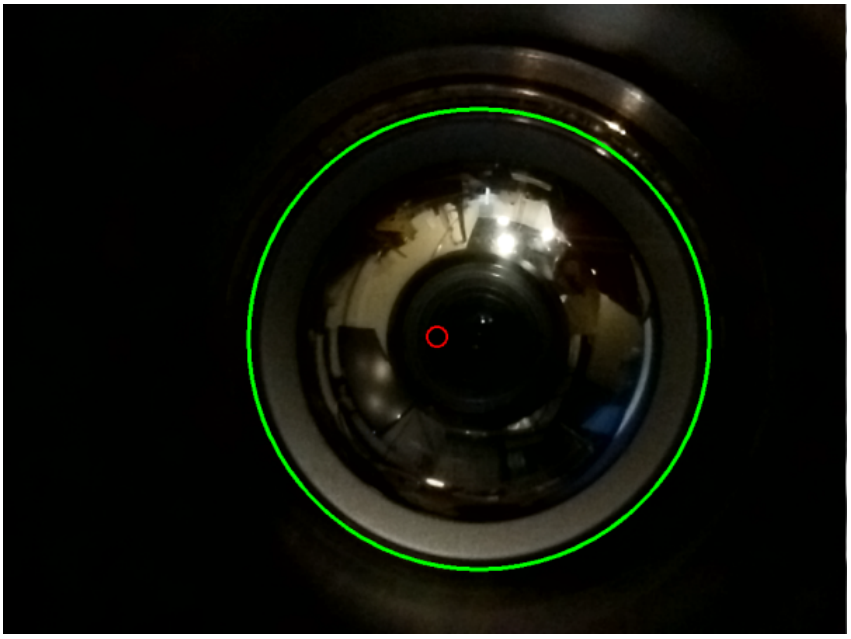

Figure 10: The hard-coded centre used for the polar transform (red), against the circle detected using the Hough transform (green).

3.3 Modifications and Development

The modifications or additions listed below are a small part of the dissertation, however, for one reason or another (see Discussion) these ended up taking most of the project time.

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 in 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 10). 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 ludicrous number of global variables. The resulting file was over 4000 lines long and almost unusable. Particuarly aggregious was the fact that roughly half of the code was irrelevant to this project, but did need archived for reference purposes. As such, it was decided that it was worth refactoring the codebase in order to make it easier to work. 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 purposes. A single thread was left in the main Activity file to act as a test thread. A lot of unused code was removed entirely.

There are still many problems with the existing codebase. This simple refactor ended up taking up a lot of time due to complications from working within the Android framework; furthermore, many concessions had to be made for the same reason.

Video Recording: In order to analyse the visual processing in more detail, a video recording utility was added to AntBot. Again platform constraints mean that the utility is perhaps not as well-made as it could be. 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 real 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. Pleasantly, this does not impact the frame rate. The video frames then have to be retrieved from the robot and stitched together using a custom python utility (a simple wrapper around an FFMPEG command). The recorded video can then be analysed offline. This utility was created to offline analysis of optical flow, however, it also highlighted a major problem with the robot which was not previously known.

Video Pre-processing Pipeline: When stitching the video frames from the recorder, it was noticed that the framerate was not quite right. The framerate on the robot was found to be an unacceptably low 2fps. 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 numebr 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 crash. 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 this to approximately 1fps cost. Similarly, an image downsampling algorithm had been manually implemented which cost around 5fps; again, replacing this with an OpenCV library implementation reduced the cost to approximately 1fps, if that. 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 does drop 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.

This raises an interesting problem. We do not know when this code was added to the robot but in any project following its insertion, the framerate surely affected the performance of the agent. We can say for sure that the algorithms used and tested in [7] were affected. Indeed, in [7] 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 simply not finished processing in time to be stored. In testing the matched-filter collision avoidance system from [7] 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 [7].

4 Methods

4.1 Collision Avoidance

The AntBot cannot be fitted with sensors which feedback to the higher models (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 [7] 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 determined [13, 16]. We also note that the often the FOE seems to shift to the deepest part of the image [16, 12] (though we should also note that in [13] an avoidance saccade is triggered *away from* the FOE).

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, [7] notes difficulty in computing it consistently. Often nonsensical values would be output by the computation. Here however, we managed to compute reasonably sensible values, though they were not consistent or predictable. To compute the FOE we use the OpenCV *calcOpticalFlowFarneback()* function to produce a dense optical flow field then use the computation from [8] (see Section **??**).

While fixing the problems raised by [7] made the computed FOE more sensible in terms of general location, the lack of consistency was cause for 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 11).

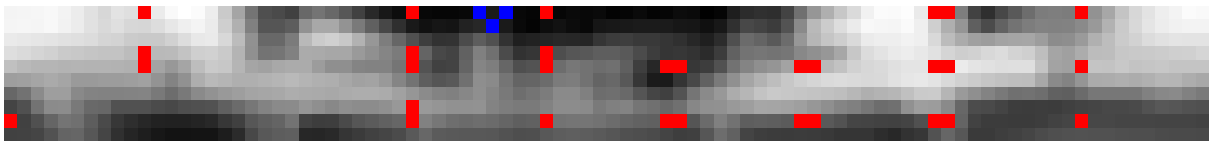

Figure 11: *[DRAFT NOTE] I may remove this figure as it does not feel useful when Figure 18 is shown later on.* A subset of points from the dense optical flow field observed by the agent. The agent is experiencing forward translational motion. These frames were captured after the framerate improvement from Section 3.3. The FOE is also shown in green. There are no obstacles present in the arena.

This analysis produced the following observations:

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

2. The amount of perceived motion is tiny: In simply looking at the video it is incredibly difficult to determine how the motion observed relates to the movement experienced by the camera.

3. The FOE jumps randomly between frames: The FOE is not consistent between frames though it does appear to stay in the same region (though we are not sure 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 [7] when examining a matched-filter system. Looming objects can appear as a large (often single frame) disturbance of the low-level noise produced by the flow field.

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 disturbances result in a large disturbance in a specific region. While this effectively closes the lid on any further investigation into the FOE/depth methods which utilise optical flow, it does give us one semi-predictable property of the flow field which can potentially be used.

4.1.2 Matched Filters

The matched-filter model of collision avoidance as seen in [7] was used without modification for the experimental results in this paper. However, in an effort to establish the ECX model, we present here an adaption of matched-filter collision avoidance which can have its outputs integrated into the CX. We will use the language of artifiial neural networks, but the principle is essentially the same as [7].

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, so we discuss them first.

ACC Neurons: ACC neurons take on the role of the leaky accumulators from [7, 13]. 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 go through 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 bound 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 filter matching, please see [7]. Once the input has been thresholded, an ACC neuron at time t can be described as:

$$V(t) = \gamma \cdot V(t-1) + I_{ext} \tag{20}$$

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 because 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 just 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 some 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 and I&F neuron, it does not function in any biologically plausible manner. It is essentially the same as the leaky accumulators from [7]. We can describe the RI&F neuron at time t by:

$$V(t) = \begin{cases} V(t-1) + I_{ext} & \text{if } t \not\equiv 0 \bmod p \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

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 behavriour 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:

$$r = \frac{1}{1 + e^{-(aI-b)}} \tag{22}$$

where a and b determine the *slope* and *bias* of the function respectively. If we set $a = 5$, $b = 0$ we get the rate function shown in Figure 12. Importantly, we must scale down the input (difference of flow sums) such that it lies in (-1,1).

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 wil result in a positive input, driving the output closer to 1. We have, in a

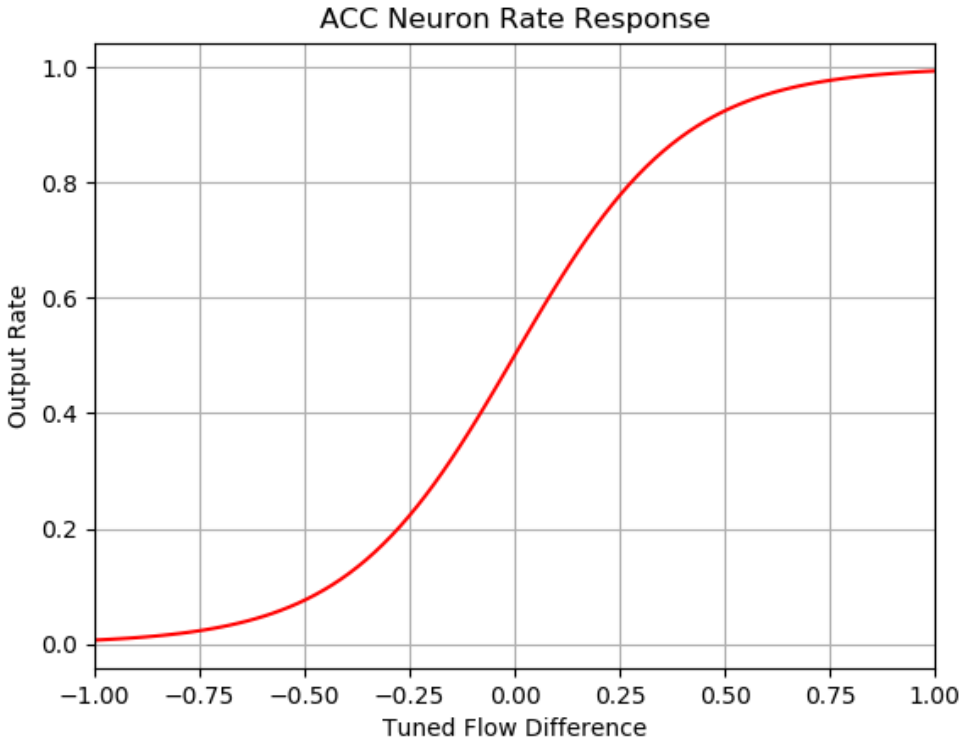

Figure 12: Function of firing rate response of the single ACC neuron against difference input; sigmoid with $a = 5$, $b = 0$. The input to this neuron is scaled down to lie between -1 and 1.

roundabout way, ended up with an optical-flow balance strategy - a flawed, simplistic, yet not entirely implausible method for avoiding collisions [11]. 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.

MRSP Neurons: There are 16 MRSP neurons laid out in the same fashion as the CPU4 layer. The MRSP neurons also utilise the same connection pattern as the CPU4-CPU1 connections. 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). We then compute an offset based on the response from the ACC neurons. This offset will always be an integer between -2 and 2 inclusive; it represents how far we wish to shift the sine wave from the current direction read from the TB1 layer (how far to the left/right of our current direction do we wish to turn). How this offset is generated differs between the I&F and rate models of the ACC layer.

In the I&F case, the offset will be -2, 0, or 2 depending on which ACC neuron fires (ACC_{right}, both, 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 [7], 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 $f(0) = -2$.

$$f(x) = 4x - 2 \tag{23}$$

satisfies these conditions. The output of Eq. 23 is then rounded to the nearest integer to determine the offset. This method is nice and simple, however it is highly susceptible to noise, as we will see this is a critical flaw.

4.2 Path Integration

The path integration circuit in use is a slightly simplified version of that presented in Section 2.3. This version does not model the TN neurons or the Pontine neurons. The model is the same as that used in [10, 20], and the robot-based tests from [14].

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* [20]. 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

Unfortunately, the complete ECX system envisioned was not fully realised. 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 were unsuccessful in performing any meaningful testing due to technical and time constraints. We present the model here and present the issues experienced and reasoning in Section 6.

In an effort to keep the integration simple at a late stage in the project, 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 far 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 2.4) the model can be visualised as the CX 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 used the rate representation for a single ACC neuron.

The output of the network can be selected by simply changing the mode of the CPU1 output stage. 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 limited 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; we had intented to approach this during the project but it was not possible. 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 [20].

5 Experimentation and Testing

We re-use the experimental environment from [7]. A collapsible boundary wall is constructed on the robot-football pitch in 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 sit 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 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 of 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 take the collision avoidance system from [7] 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 [7]; 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 [7] 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 [7]). 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 [7]. Any test in which the CA system failed 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. This does not make the robot point directly towards the nest. The justification for this procedure can be seen in Figure 13. 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 required angular correction. 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.

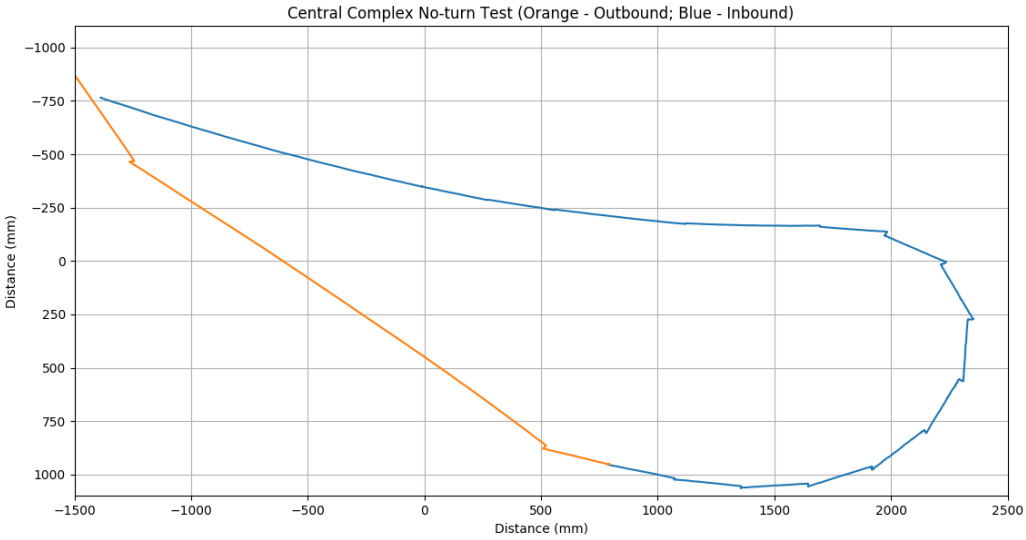

Figure 13: An initial recording from the Central Complex PI experiments. The agent successfully navigates home but requires a large turning arc to point back towards the nest. This is because the output of the CX is not a precise angle but instead a turn direction with some strength. This behaviour was 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 (i.e. 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; in essance, if the robot collided with an arena wall.

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 [7] 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). The distance measurement is still updated continuously; as 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 bad 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 the desired behaviour.

The agent was set on a collision course with a cluster of tussocks and watched for consistent reaction behaviour. No such consistent behaviour was observed. To further test the hypothesis, 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) may be viewed on the smartphone screen; unfortunately, it is too small to really pick out any details of a flow vector field. As a general point, testing optical flow behaviour on the robot is generally 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 which would not normally be present. This is generally true with the robot; it is hard to produce reliable output from algorithms when testing.

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 [20] 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 [7]. Unfortunately, neither were really useful in testing the FOE position; we could log it, but there would 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 almost immediately denounce this method for use 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, 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 immediate path, as such, 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 to the agent. The arena can be seen in Figure 14, the robot's intended path is shown in red.

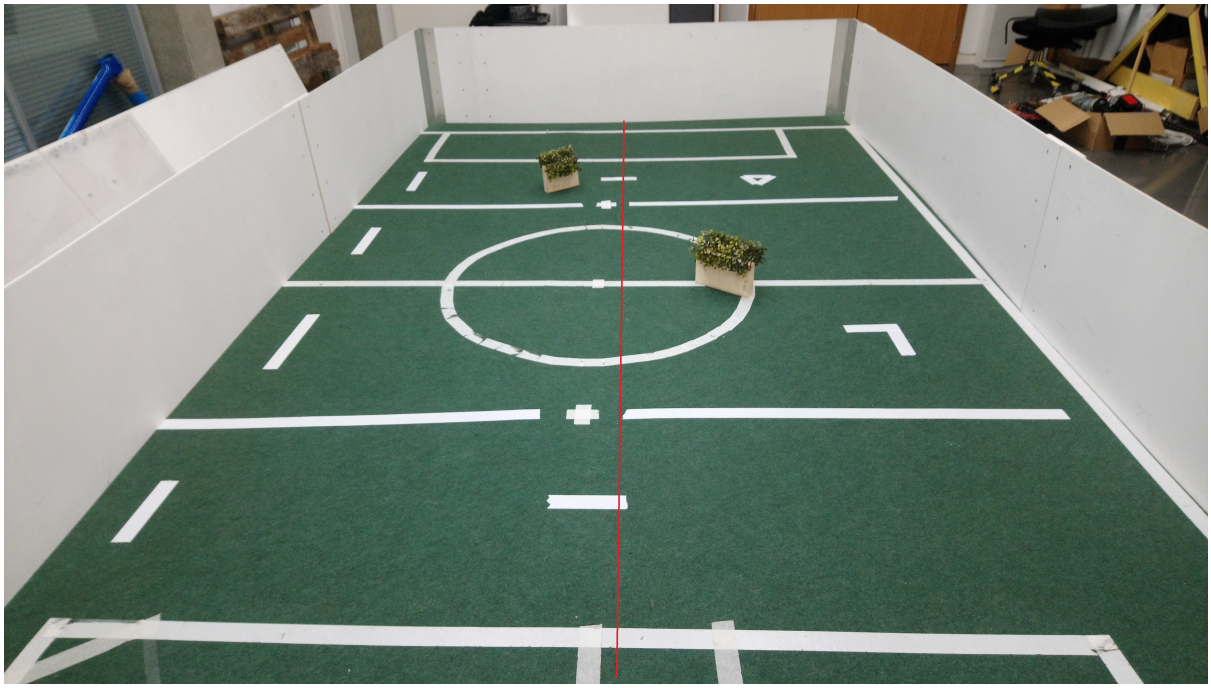

Figure 14: The non-empty arena used for testing the shifting expansion field system. Tussocks were set to the left and right of the robot's trajectory (shown in red) so that they were not on a collision course but should still affect the expansion field. We had hoped the FOE would drift left then right as each object was passed.

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 sin curve and shift it to the left or right of the current heading. We can then check if we can generate consistent turns consistently.

We then test the three different implementations of the ACC neurons separately. The testing for each type did not extend very far. Each implementation was tested in a mostly clear arena with multiple obstacles placed in a large, obvious cluster in the centre. The robot was started from the starting block (marked with masking tape) on the south side of the enclosure, on a direct collision course with the cluster. The agent would then attempt to traverse the arena without collisions using only the neural CA system. The I&F variant[11] of the ACC neurons was the only implementation that functioned consistently enough to take forward to the next stage of testing.

5.3 ECX

The testing methodology for the ECX was to test each individual system and then full combination. Our earlier experiments with the CX demonstrate the capability of the

[11] We take the LI&F representation forward, observed behaviour between RI&F and LI&F was not significantly different.

circuit as a path integrator - and, of course it has been demonstrated in simulation [14]. This model has already been tested in conjunction with the MB (*Zhang's* CXMB)[20]. Similarly, the MB model has been shown to provide robust visual navigation on the robot [7].

Therefore, the only things left to test are the neural CA system, and then each step of integration, CA $\rightarrow$ CXCA $\rightarrow$ ECX. Note that testing CA alongside the MB model would be redundant as the routes recalledf from visual memory would incorporate any obstacle avoidance maneuvers. Fortunately, we can simply use the full ECX model and turn particular outputs on or off depending on the desired functionality.

Unfortunately, we did not advance past the CXCA testing stage. The CXCA was tested only with the LI&F ACC neurons. We used both an empty environment (to check for basic functionality) and a cluttered environment; one with a singular central obstacle and a second with a tussock density of 2 per pitch quadrant.

6 Results

6.1 Path Integration and Evaluation

We are pleased to report excellent 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 15that 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, essentially, 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 15), we think it is fair 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 16. 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 a successful demonstration of the capability of the CX (especially with the informal trail in Figure 13), 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 a platform with better odometric capabilities (see Discussion).

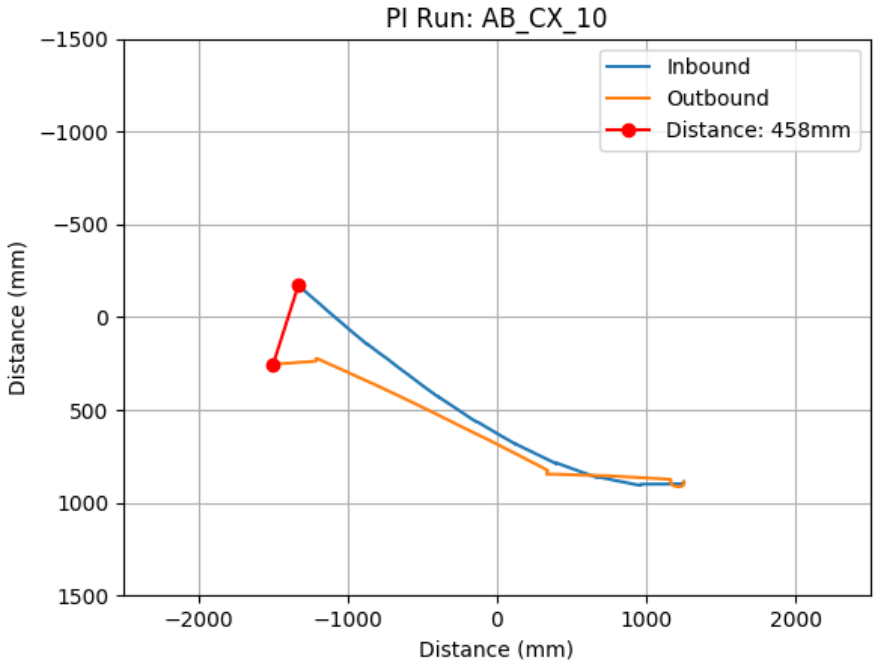

Figure 15: PI test AB_CX_10. This was considered the worst failure of the system despite not having the greatest deviation from the start point.

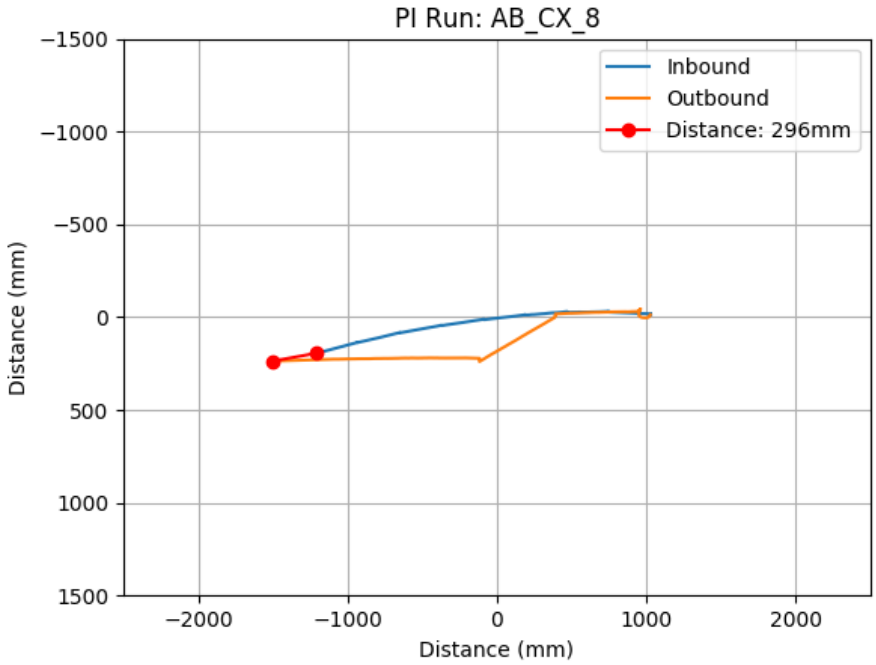

Figure 16: PI test PI test AB_CX_8. Arguably the most successful recording from the formal experiments. The agent still corrects to achieve a better homing path, despite the fact it could likely have travelled in a straight line.

Despite our justification, it may also be prudent to remove the 180° turn at the end of the outbound run. In some recordings, it can be seen that the robot is still pointing

directly home after its outbound run (Figure 17), though this is not always the case.

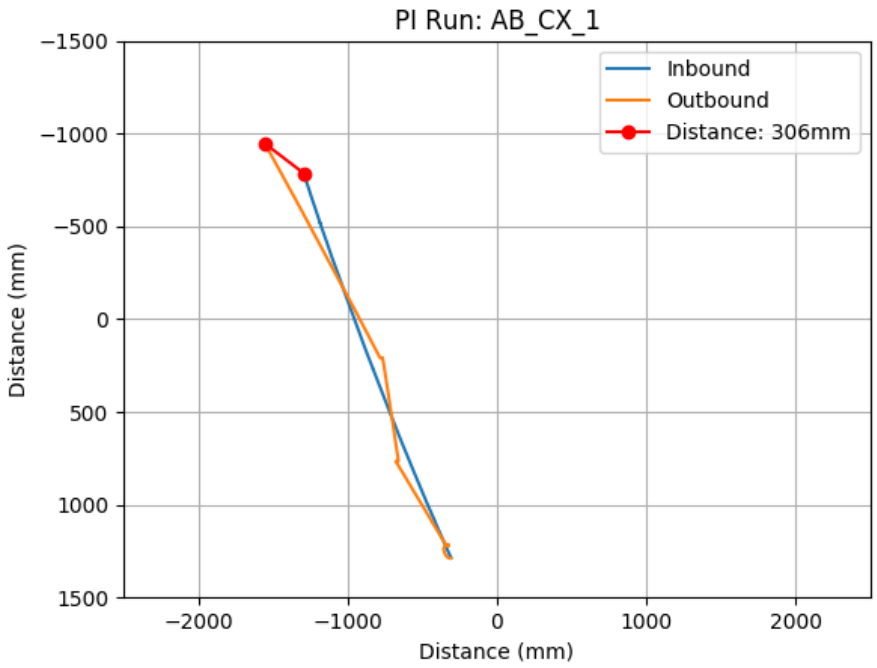

Figure 17: PI test AB_CX_1. In this case, we note the exact same methodological flaw we sought to avoid in which the robot requires no correction to home successfully.

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 to generate a steering command, in the case of the AntBot. We limit our claims as:

1. This is the only platform on which testing has taken place.
2. It cannot be denied that the works which inspired this method achieved success in using the FOE to direct motion [13, 16].

In light of this, 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, we turn to the five image frames presented in Figure 18.

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 example in Figure 19.

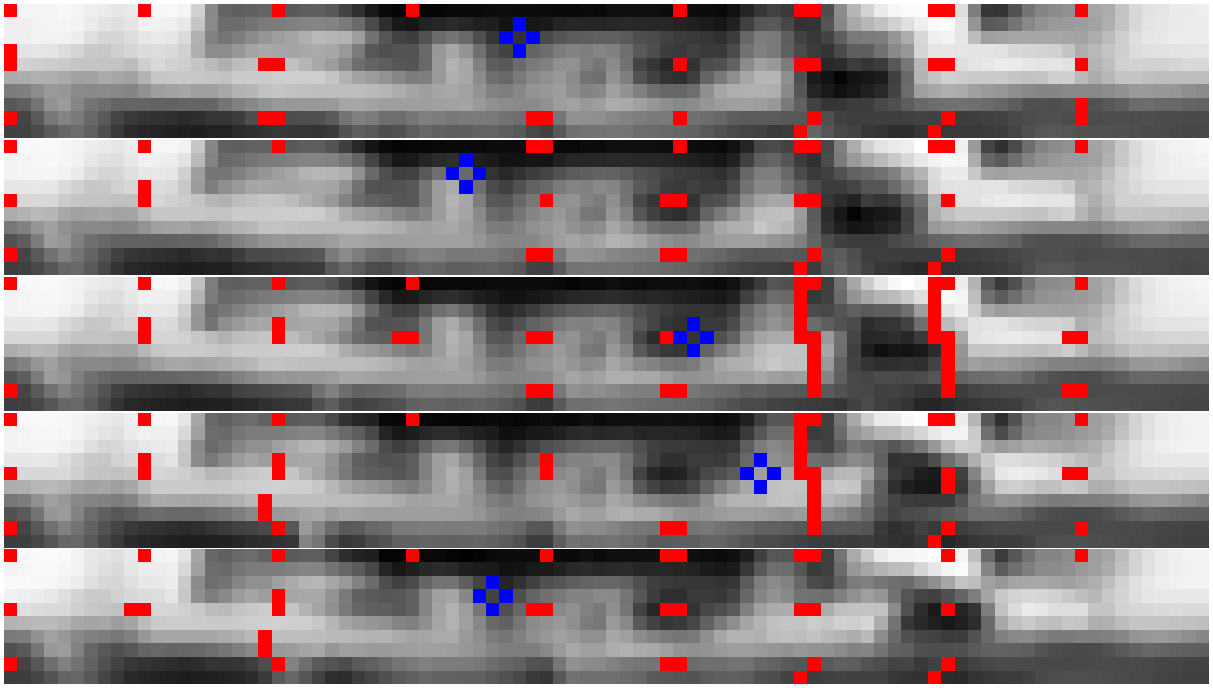

Figure 18: Five consecutive image frames with optical flow information superimposed over the top. It can be seen that the FOE position (central pixel of the blue cross) is not consistent across multiple frames. This figure also shows the disturbance caused by approaching a tussock on the right hand side (3rd and 4th frames). Video captured at approximately 10fps (i.e. these images were captured accross roughly 0.5 seconds).

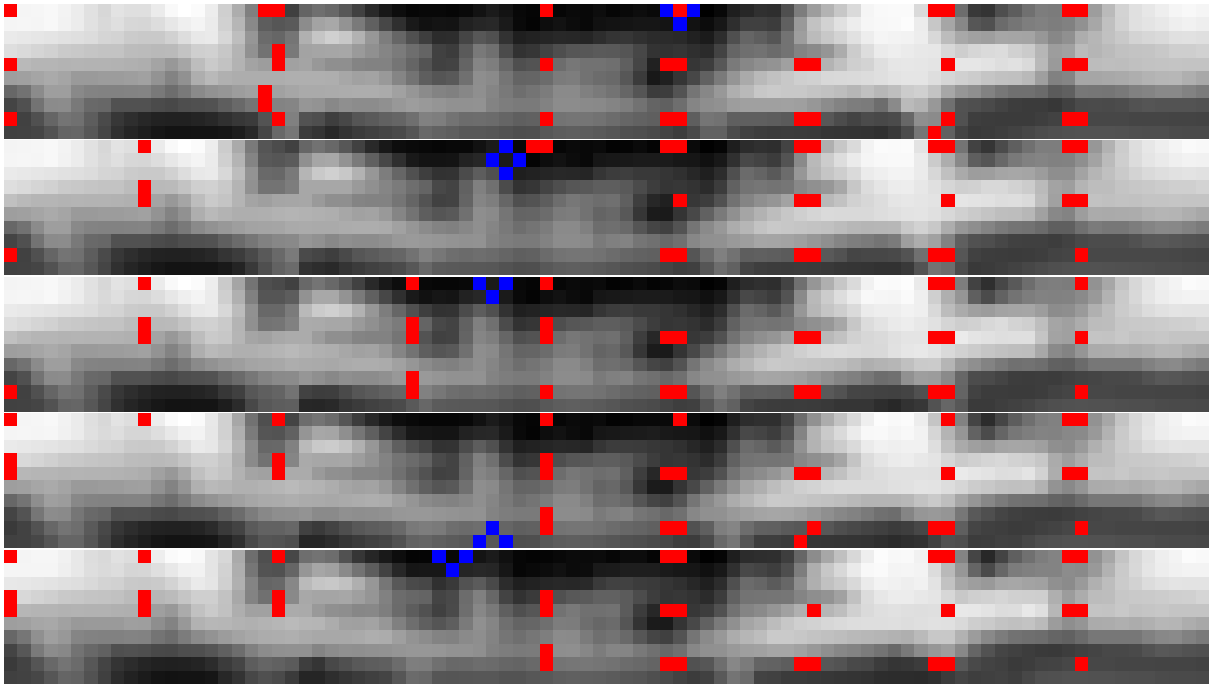

Figure 19: Another set of consecutive frames, this time captured in an environment with no objects present. The FOE is still unpredictable, though it does not move as drastically in the horizontal axis.

In this case there are no obstacles present in the arena, and thus minimal sources flow field disturbance. Nevertheless, we can still see that the FOE jumps, though not quite

as drastically. The views are perhaps deceptive 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.

Going forward, we have no further formal results to present. We will instead discuss behaviour experienced during developmental testing. These are not discussions from which concrete conclusions may be drawn, however, they may provide insight for anyone considering implementing the neural filter model for collision avoidance, or the ECX, as a continuation of this work.

6.2.2 Neural Matched Filters

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

I&F: Unfortunately, the performance was generally poor, though this was the only variant which managed successful collision free runs (though this was not consistent enough to gather any formal data). It is no surprise that this variant came closest to replicating the behaviour (if not the performance) of the original model from [7] as it is, functionally, a dressed up version of this model (with the addition of neural steering generation).

With this in mind, one would reasonably ask why this model does not function consistently where the one from [7] did. Indeed, we used the model from [7] successfully in this work for our path integration experiments (and it was tested successfully alongside the MB model in a cluttered environment during the code refactor process). The neural OFCA model was implemented after the improvements to the visual pipeline; as this was the only change we suspected this was the problem. In testing the original model from [7] *after* the improvement to the framerate, the behaviour was noticably different (and degraded).

As mentioned in Section 3.3 we suspect this is due to the drastic increase in flow information which is now fed into the model. As can be seen from Figures 18 and **??**, the flow information is relatively noisy with the features indicitive of an obstacle being fairly coarse. The tuning methods applied in [7] where the accumulation and reaction thresholds are adjusted are clearly no longer sufficient; in testing, the model was either hypersensitive or hyposensitive. The line for a functional middle ground is clearly very fine. Tuning at the sensory level is clearly required; fortunately, such tuning could also prove vital for the rate model.

Rate: The rate model provided no predictable reaction, 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 is supplied as input to compute the firing rate. In the I&F and algorithmic variants, this problem is (or was) mitigated by the

accumulation threshold parameter. It is clear that such a parameter could be added to the rate model, however, we propose a solution at a lower level.

One should look to clean the optical flow field in some way. 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 [7]). A crude but potentially effective method as it can be seen in Figure 18 that looming obstacles result in a few vectors with large displacements; all other information could be considered noise. Alternatively, different filters[12] or even a dedicated sensor could be used (see Discussion).

Despite the fact that the rate model as presented does not behave consistently in the presence of obstacles, it would still be our preferred approach going forward. We feel it better fits the existing CX architecture and also allows a proportional response to be generated. It is a more flexible and perhaps, a more biologically plausible approach to the simulation. Based on our previous experience with optical flow models for collision avoidance we are confident in our assertion that the erratic behaviour can be (mostly) explained by a noisy optical flow field though we should reiterate that this is conjecture.

6.3 ECX (CXCA)

As stated in Section 5, the ECX model only got as far as the CXCA (Central Complex with Collision Avoidance) step in testing, using the I&F CA model. This parameterisation of the CA model was consistent enough to allow specific tests in tailored environments such that we could at least check that steering signals were being generated as a combination of the path integrator and the collision avoidance system. As such we could observe (mostly) successful collision avoidance behaviour on the outbound run. The inbound runs proved less successful, however. Due to the hypersensitivity of the collision avoidance system it proved difficult to find a reasonable weighting to balance the two systems.

Intuition would suggest that we generate the vast majority of the steering signal from collision avoidance; any steering signal generated by this system is clearly an immediate danger to the agent, the path integration circuit can compensate later. However, under observation the path integrator never gets the chance to compensate. An early prototype of the CACX integration allowed the CA subsystem to completely overwrite the steering command in the event of a collision detection; with a hypersensitive CA system this presents the obvious problem that CA saccades become the only steering output. Later versions which allowed integration of the outputs experienced the same problem; lowering the precedence of CA either did not mitigate the problem or resulted in an unacceptable degradation of an already poor CA system. We can at least conclude that the CXCA and therefore the complete ECX are plausible once the problems are ironed out of the CA system. We regret not making more progress towards making this a usable model, though a complete implementation is now available on the AntBot.

[12]The flow filter in use is still the filter implemented by *Scimeca* for speed retrieval[10, 7].

7 Discussion and Future Work

This section will largely delve personal opinion and feelings towards the work completed. Ironically, the word that comes to mind when thinking on this year's work is: incomplete. While we have presented and implemented the ECX, a complete theoretical base model for insect navigation, we did not manage to gather and present any formal experimental data. The only real barrier to testing towards the end of the project was the tuning of the collision avoidance system, something not predicted to be problematic as we already had a working system from [7]. This became a problem as the work completed in [7] was completed under the assumption that the robot's basic systems functioned sufficiently well that they need not be changed. This assumption was seriously flawed (as noted towards the end of [7]), but such an assumption is inherent in a generational project or platform such as the development of AntBot.

It is our view that the AntBot should be retired. We believe that the platform has fulfilled its usefulness and it would be worth having a new platform constructed, perhaps as part of a new project. There are certainly still tests which could be conducted on the AntBot; a Mushroom Body circuit with real-valued weightings could be tested, the ECX could be tested if the CA system could be tuned, in fact, likely all of the work from [7] could now be meaningfully repeated. Something as simple as a framerate change could be enough to effect significant change in the MB circuit performance; we have already reported the changes seen in the OFCA sytem. However, the benefit of having the AntBot available to test such system is vastly outweighed by the potential technial difficulties the researcher could experience; it is not an exaggeration to say that we lost weeks worth of work due to ultimately simple problems. We justify our recommendation by reporting our own experiences from [7] and this work.

7.1 Matrix behaviour

During [7], we noted that one of the main issues we experienced while implementing visual scanning was matrix manipulation (shifting rows and columns). When checking the manipulation alone, it functioned perfectly fine, however, when moving into its own function problems arose. As it happens, OpenCV matrices are represented by a wrapper object in the Java bindings which essentially acts as a pointer. When treating parameter passing as a reference, suddenly functions involving matrices began to work as expected. This is a technical detail, it is available in the documentation (albeit hard to find), why is it worth mentioning?

It is worth mentioning as neither of our direct predecessors noticed this (as we understand it, the code was changed significantly after *Eberding's* project so we cannot include him). As a result, some truly ridiculous code has wormed its way into the robot over the years. During our code refactor as part of this project, we found numerous unused functions which treated matrices by-value, where identical code suspiciously appeared in the visual pre-processing pipeline. It was clear that previous students had attempted to give each stage its own function, but this did not work and all code was moved directly into, frankly, a monster of a callback function - called for every camera frame.

This is understandable. During a masters or honours project, technical hiccups like this

are easier to bypass than to solve. Time is tight and there are "more important" things to worry about. Each student need only worry about the codebase for one academic year; each student going out does not worry about the code they leave behind and each student going in is told they have a working platform and codebase on which to test their models. Clearly there is some disconnect in these attitudes. Please do not think that we stand upon the mountain top bemoaning the the sins of our predecessors; we are by no means innocent in this department. The code refactor undertaken is an excellent example; it made development far easier for the project, but there were design decisions taken that make no sense in a general software engineering context, but perfect sense if you are experienced with the AntBot. It is inevitable that every student will have to deal with the mess of the previous and ultimately contribute their own. We use the example of matrix manipulation to make this more general point.

This is perhaps not specific to the AntBot, any generational project would suffer from the same issue. Methods of mitigating this effect simply add more work to the student who must document and justify all coding decisions made, limiting the already limited time they have to work on the biorobotic component. We believe that the AntBot is on the verge of becoming unusable.

7.2 Lack of documentation

The only documentation on the platform is available in the form of previous dissertations. These do not go into sufficient detail to explain the function and usage of each section of the code (as technical documentation should). This means that some potential functionality is near impossible to understand. Again, a specific example; we noted earlier that encoders were not available to provide feedback for odometry. Actually, it could very well be possible to get collect encoder information and send it from the Arduino layer to the smartphone. We do not know, because the serial communications system is not documented, the path taken by serial information is incredibly difficult to follow from the code alone. While there are examples to follow for sending data *to* the Arduino and decoding it, there are no examples for encoding data and sending it *from* the Arduino. It was decided that it was not worth the time required to understand and we would default to odometry based on time (as in previous iterations of the project) so as to collect some data on the CX functionality. In fact, the encoders have not been used since *Eberding's* original work, where they were used only for a specific purpose. This is unfortunate considering the encoder capability of the Rover 5 chassis is given as one of the main reasons for moving away from Roboant [3] (perhaps made worse by the fact that optical encoders *are* available for the Polulu Zumo motor board used by Roboant).While we have successful results to present, it is undeniable that the decision not to use encoders likely affected the performance of the model. Encoders are not the most accurate odometry technology, but they would provide a more accurate measure than time - especially on AntBot, where threads are constantly put to sleep to permit time for serial communications.

7.3 Flimsy construction

When we inherited the project in late 2017, the robot chassis was not in one piece; the top plates were not attached as students required access to the battery connection to turn the robot on or off and remove the batteries for charging. This was solved as part

of [7] but it should never have been an issue with a pre-existing platform.

When returning this year for this project, the 360° camera attachment had been removed and used for another project. It was reattached and work commenced. We noticed that the CA system was not functioning as we expected during early tests of component systems for the ECX. The system constantly reacted in the same direction, regardless of the stimulus present. Initially we thought this was due to the code refactor (the code, but not the logic, was changed), then due to parameter tuning, and finally we noticed that the processed image was warped. This warp left a large blind spot on one side of the image which explained the behaviour. Previously this warp had only been experienced in cases where part of the lens attachment was obstructed, however, this time the full view was available. This resulted in at least two weeks of breaking down the image processing and unwrap process before finally tracing the issue. Namely, that the centre of the circle used for the polar transform which unwraps the image was hard coded to a specific pixel value. Despite the fact that we were specifically informed, and legacy applications demonstrate, that there was an automatic detection method in place. Two weeks of available time lost (after finding the image warp) to something that should have been a non-issue.

7.4 Limitations imposed by software (Android)

7.4.1 Library conflicts and limitations

The existing hardware is reaching the end of its life. Specifically, the smartphone is out of date; while mid-tier modern phones could provide better sensors (e.g. compass), faster communication protocols, or faster processing speeds, arguably the biggest limitation is now becoming the software as a result of outdated (and unsupported) hardware.

This was most prominent when developing the video recording tool. To record a video directly required use of an FFMPEG wrapper, available via a specific library (JavaCV). This library also included OpenCV bindings which led to conflicts with a system which already included an OpenCV library. Furthermore, the OpenCV portion could not be excluded. Switching to the JavaCV library would have required rewriting much of the existing OpenCV imports and code.

One other potential avenue existed in importing the required libraries independently of the JavaCV library, however, the minimum SDK version required for at least one of the components was higher than the project target which could not be increased because it is not supported on the version of Android installed on the smartphone. Therefore, we ended up with the implemented hack solution of buffering video frames to be stored as individual images which are copied from the robot and stitched together by a python script (using FFMPEG). What we end up with is a massively roundabout way of accomplishing the same thing by using the same technologies in different ways.

7.4.2 File management

Small gripe, but retrieving files and file management in general can be convoluted. As an example, the videoframes are stored to a particular directory, this directory is

deleted and reformed for every recording to ensure that the old frames are always overwritten, however, this does not happen consistently; why, is a mystery. Often files written during the last run cannot be seen when the phone is plugged into a computer, the user must go into the file manager on the phone and copy the files to a higher directory for them to become visible. This was experienced in [7] when using *StatFileUtils*, but only when dealing with the video frames this year did we note just how ridiculous this process is. The only way to guarantee your video data will be recorded and available is to go into the file manager on the phone, delete the destination (and any copies made in an attempt to read it), run the agent, go back into the file manager, copy the destination directory (usually to the top directory on the SD card). In writing this it occurs that a solution could be to install a Secure Shell (SSH) server on the smartphone to make file transfer easier, but this then requires further development and testing for something that should be available by default.

7.4.3 Software structure restrictions

Android applications have a reasonably strict structure. Certain things, for example scoping rules for specific variables can be enforced unexpectedly in ways which only make sense within the Android ecosystem. When undertaking the code refactor we discovered reasonable explanations as to why everything had been kept in the same file. Our experience of this is minimal but it causes issues when trying to move code around in *traditional* ways. As an example, consider global variables; generally considered bad practice unless absolutely necessary, but, if we want to display their value on-screen then they are absolutely necessary. All variables to be displayed must be global for reasons we admit we do not totally understand. In many cases these are variables modified in multiple places, at multiple stages, in multiple threads (though in theory, not simultaneously). This is a small example but application structure cropped up a few times in different ways. In another case, a function required a Context object which was not available until particular point in application initialisation.

This is, again, a small problem but it can result in roundabout solutions to problems which would be simple under normal circumstances.

7.4.4 Overhead

The overhead required for learning enough about the Android ecosystem to successfully develop within it is massive. It is no surprise that the original application structure from [3] was almost immediately dropped, as this would require learning how applications in Android interact to pass data back and forth. There may also have been latency involved in inter-application communications but no previous works have reported this as a reason for moving away from the original application structure.

While it is not unreasonable to ask prospective students to learn about Android development, asking them to interpret the existing application structure and legacy code without any documentation, is. While the obvious solution is for a student to reach out to the last *caretaker* of the project, it is also reasonable to expect a student to simply look for the place in the code where they can insert their own, and then build everything around that. It is no surprise that this location was found to be the AntEye application as all visual information is immdiately available.

This raises a problem in that there is still legacy code that must be running alongside AntEye. An Android Toast appears when starting any thread which states "Visual Navigation service started.", and another which appears when stopping a thread to inform you the service has stopped. After multiple years of development it can be almost impossible to figure out what is running where and doing what outside of the small bubble provided by AntEye. Thus, not only does a future student have to understand Android, they have to be able to notice and understand the workarounds applied by previous years (we will bring it up again, with no documentation).

7.5 Limitations of hardware

The construction of the robot further limits development. It is impossible to use any useful form of feedback control due to the communication latency between the smartphone and Arduino [7]. Movements are therefore coarse and inconsistent. Feedback control could perhaps be implemented at the firmware level, but attaching sensors to make this possible is difficult (a compass sensor was added as part of this project, but getting it to show up on the I^2C bus proved difficult). This would futher reqire re-writing most of the firmware on the robot (again this firmware is uncommented and undocumented) to implement control adding more busy work to a potential project. It was actually planned that the control code would be re-written as part of this project but this gradually slid further down the list of priorities as difficulties were encountered with higher level systems.

It could also be argued that implementing proportional control at the level of the firmware is not useful enough to warrant doing in the first place. The motion of the agent would still need to be performed in the same discrete steps. The only potential gain would be more accurate turns which, given the visual scanning implemented for the MB circuit and the lack of scanning in the CXMB/ECX variants, would be of minimal importance.

While it is difficult to add sensors to the Arduino, it is impossible to fit additional sensors to the phone. Dedicated optical flow sensors could provide more useful input than the field computed from the field of view, however, they cannot be added.

Already mentioned are the numerous delays required to execute commands. Stopping the robot after detecting, for example, an obstacle, adds a second of delay to the thread calling the function. It also seems that instructions can remain in the pipeline even after the application has been stopped, to the point that a full reset is required to guarantee consistent behaviour. This reset procedure is:

1. Switch off the chassis/motorboard.
2. Unplug the serial cable from the smartphone.
3. Kill the entire application on the phone (not just the thread that was running).
4. Re-insert the serial cable.
5. Restart the application.
6. Switch the chassis back on before running any thread.

Performing these steps out of order (or skipping any individual step) will result in unpredictable behaviour. The robot behaves as if it were still executing the previous task. Killing the application is made more difficult by the fact that the application can randomly vanish from the list of running applications, despite the fact it is still running. You can only be sure it has stopped if you see the "Visual Navigation service stopped" Toast. At times this behaviour can cause uncertainty as to whether a model does not work, or whether the robot is simply acting out.

The robot is also large, making testing in the available space difficult. Early in the project, we considered reverting to the old Roboant design. The old parts were put together and tested, but again, this would have required re-writing the firmware at the Arduino level as Roboant used not only a different Arduino board, but a different motor board. Given that the angular turning on the Roboant chassis was observed to be less accurate than the AntBot chassis, we decided not to invest the time in re-writing the firmware. Reasonably accurate turning was preferable to a smaller form factor, but with modern systems, a small robot capable of accurate locomotion should be easily constructable.

7.6 Assumed functionality

The final and most serious issue to raise (which most previous points boil down to) is assumed functionality. When taking on the project it is not unreasonable for a student to assume that the robot (having been through many years of development) works sufficiently well for them to test their thesis. In 2017, we started work on the AntBot with exactly this assumption. AntBot is presented as a stable, reliable platform for testing and development. The discussion here presented should show that this is not the case.

The best example to show this definitively should be the frame rate bug discovered as part of this project. A significantly reduced framerate affected the perfomance of the CA system tested successfully and likely affected the Mushroom Body circuit's performance as well [7]. Experimental results can only be trusted if the platform can be trusted to work correctly to provide models with the correct information and actions. The MB received reduced visual information, the CA system received reduced motion information, and the CX received highly inaccurate time-based displacement information. Not one of the systems under test received reliable input, simply due to the implementation of the robot.

Many other platforms could be possible, our favoured approach being a small omni-wheel design capable of holonomic motion with algorithmic redundancies for any biological system (e.g. an ultrasonic collision avoidance system to compare to any visual model). Regardless of platform, it cannot be denied that many of the problems experienced are inherent to a generational project, unless development (software and hardware) is managed more closely.

8 Closing

In progress, intended to be a single paragraph summary of the work presented.

9 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] W. Burger and B. Bhanu. On computing a 'fuzzy' focus of expansion for autonomous navigation. In *Proceedings CVPR '89: IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 563–568, June 1989.

[3] 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.

[4] 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.

[5] 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.

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

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

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

[9] 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.

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

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] Barbara Webb. PLACEHOLDER: JEB REVIEW PAPER; GET CORRECT CITATION", 2018.

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

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

[20] 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)* [19].

[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

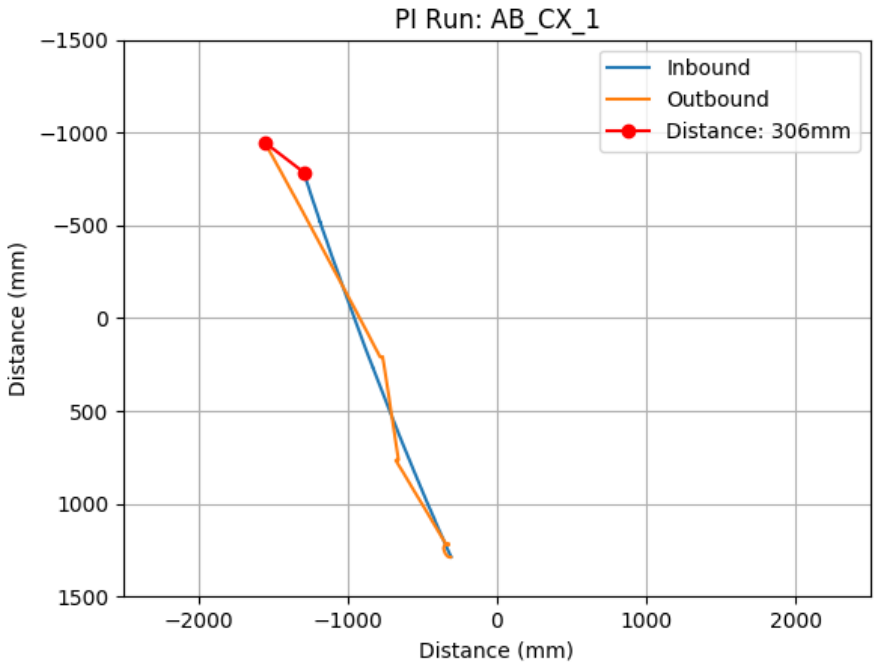

Figure 20

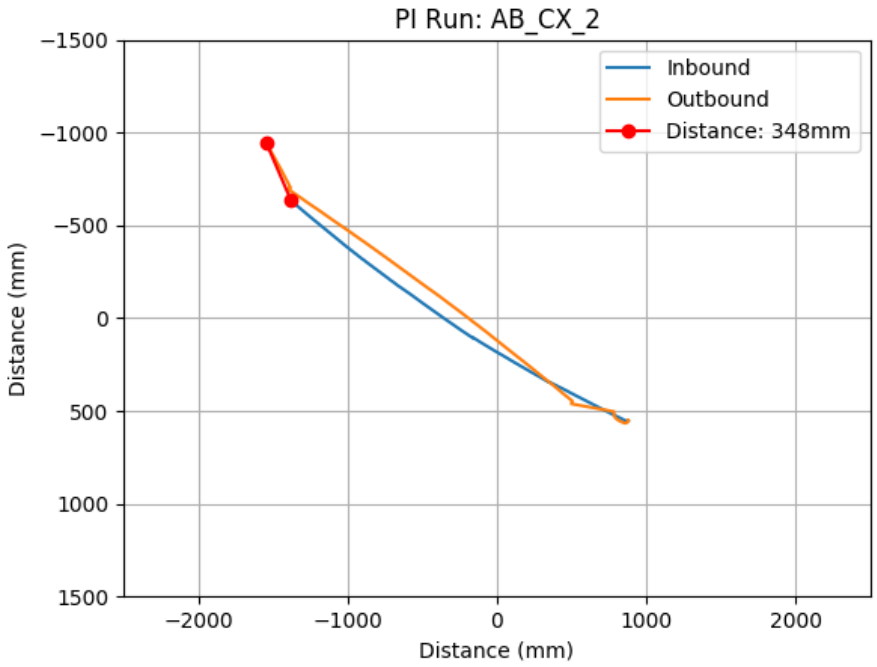

Figure 21

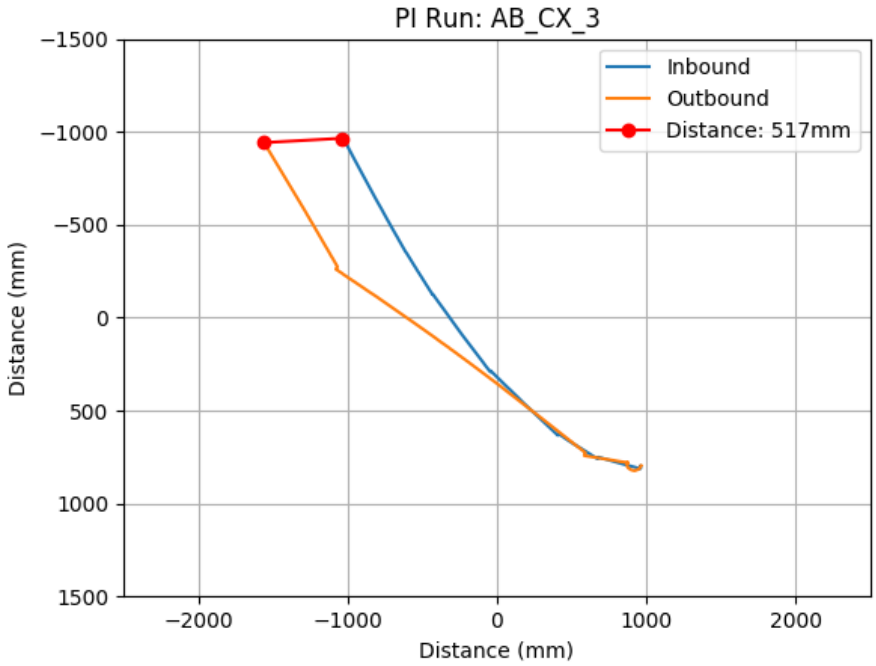

Figure 22

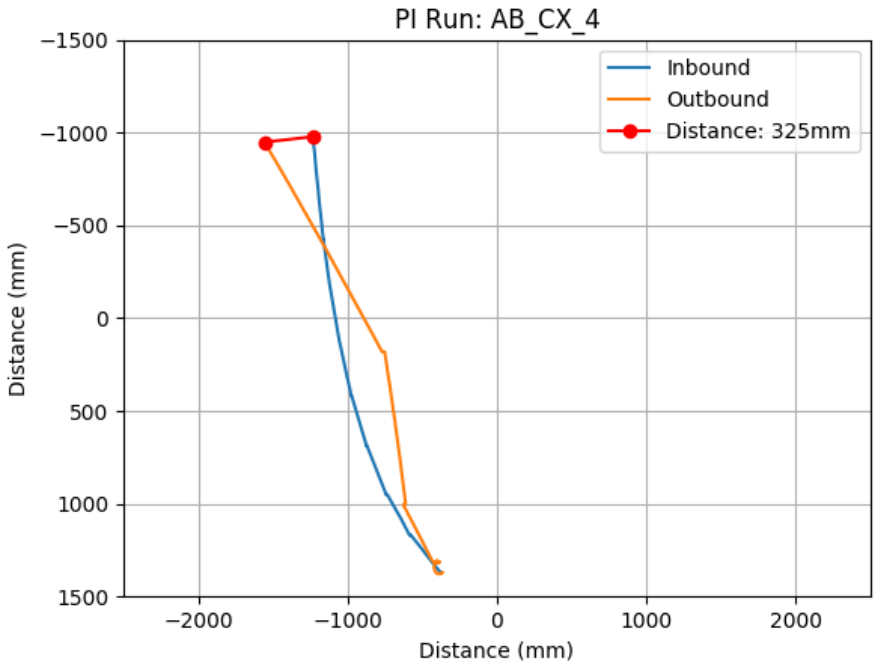

Figure 23

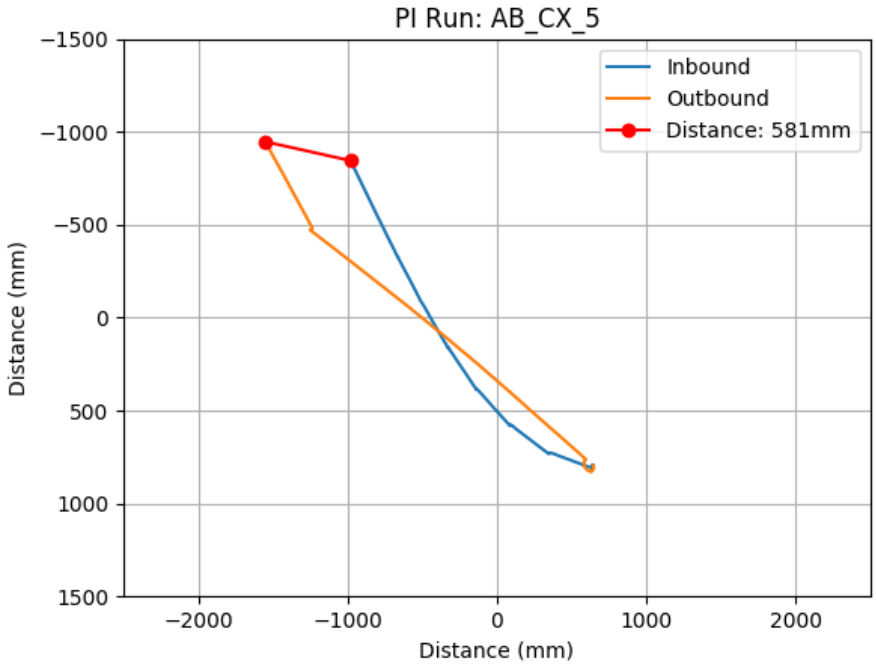

Figure 24

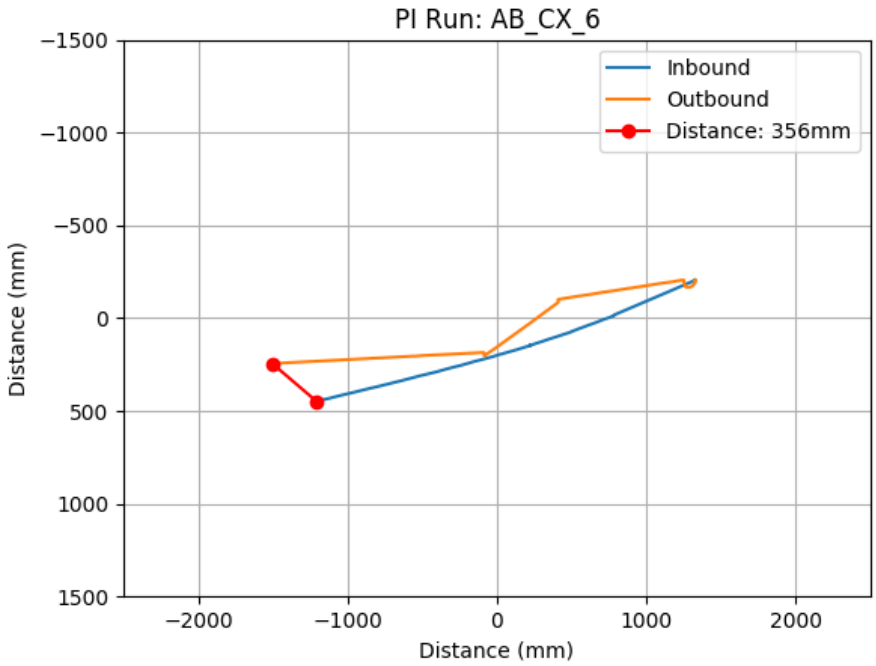

Figure 25

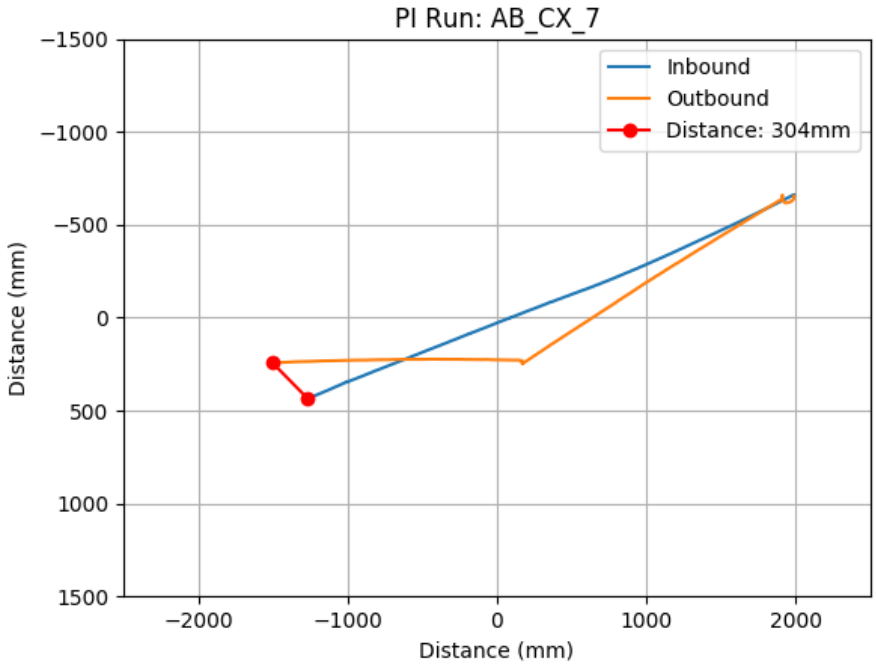

Figure 26

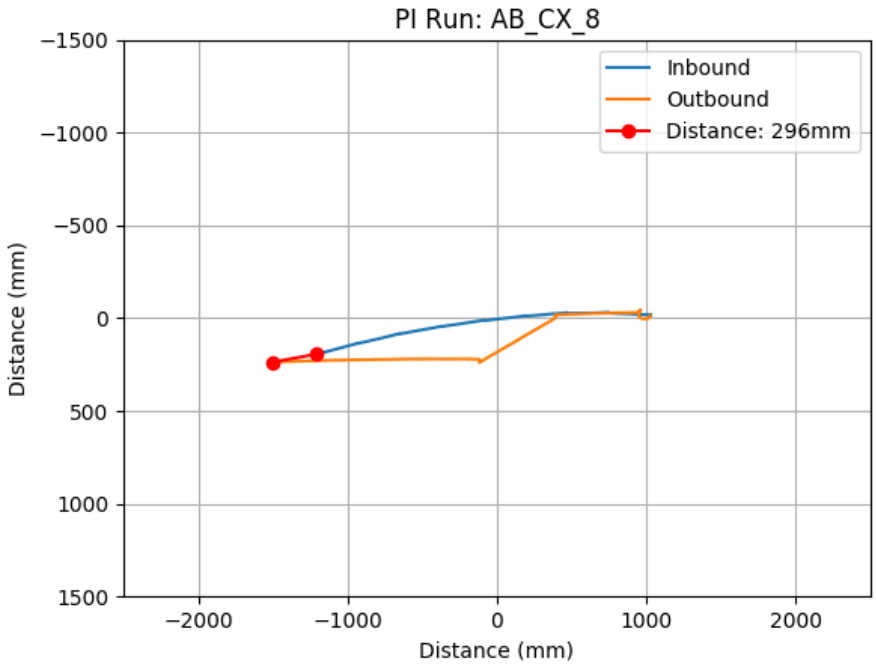

Figure 27

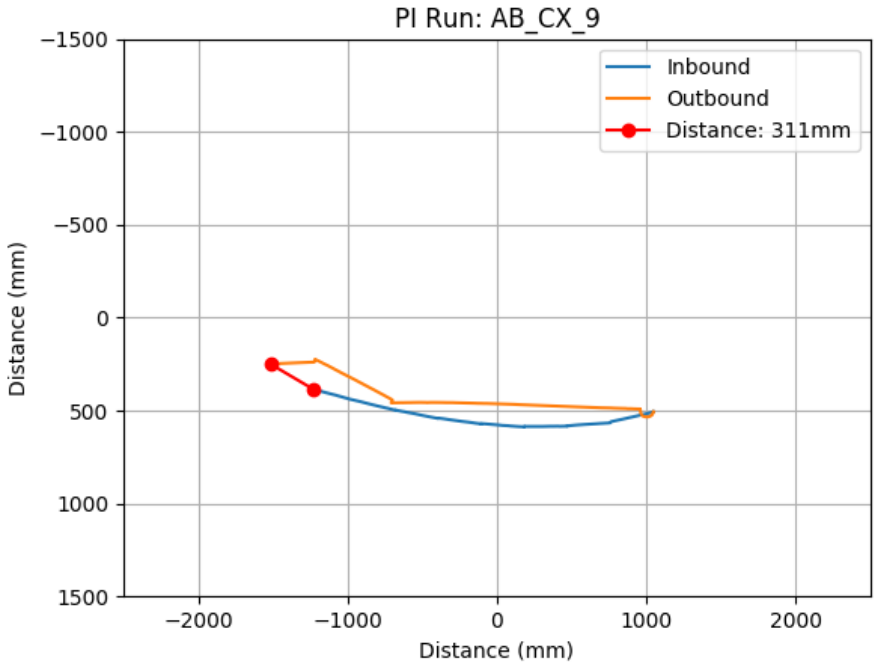

Figure 28

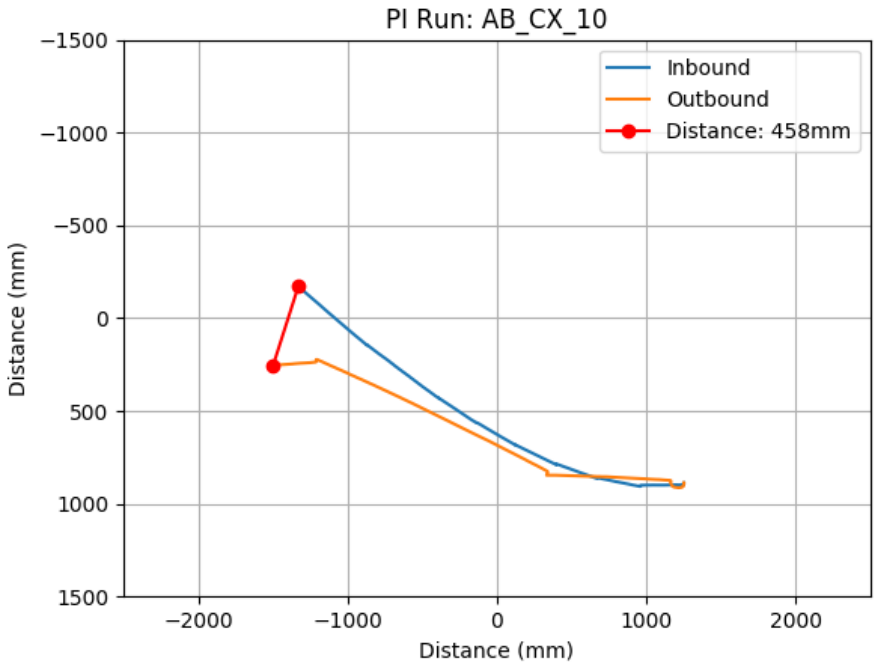

Figure 29