Developing AntBot:
Visual Navigation based on the insect brain

Robert Mitchell

Master of Informatics
Informatics
School of Informatics
The University of Edinburgh
March 22, 2018

Supervised by
Dr. Barbara Webb

Acknowledgements

I would like to take the opportunity to thank my supervisor, Dr. Barbara Webb, for her guidance, and invaluable insight on the subject matter. My gratitude also extends to Zhaoyu Zhang and Leonard Eberding, two of my predecessors on this project; both have been extremely useful in explaining the existing codebase and operations of the robot where they were not always clear. Finally I should like to thank my parents, for their unwavering support throughout my education; I could not have made it here without them.

Abstract

The abstract will be written as one of the last parts of the report to allow an accurate summary of the project as a whole.

Declaration

I declare that this disseration 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

Contents

List of Figures

List of Tables

1 Introduction

Desert ants (*Cataglyphis velox*) have the remarkable ability to navigate through complex natural environments, using only low-resolution visual information and limited computational power. It is well documented that many species of ant, and other hymenoptra are capable of very robust visual navigation; however, it is as yet unclear how the insects perform this seemingly complex task with such little brainpower. In this paper, we will focus on using and extending an existing model for visual navigation in ants using the Mushroom Body circuit, an artificial neural network which emulates the Mushroom Body neuropils in the ant brain. We will also discuss biologically plausible methods of visual Collision Avoidance using Optical Flow. A robot (AntBot) has been constructed [6] to allow us a testing platform on which to implement, and experiment with, the algorithms in the *Ant Navigational Toolkit* [14].

1.1 Motivation

Though we are able to observe and mimic algorithmically the visual navigational capabilities of insects, we still do not understand the precise methods by which this process takes place. The model we will look at was propsed by *Ardin et al.* [2], which takes the Mushroom Body (whose function was thought to be primarily for olfactory learning), and shows that this provides a plausible neural model for encoding visual memories.

The MB circuit has been implemented and tested on AntBot by Eberding and Zhang respectively, however the existing MB circuit is fairly simple. It uses binary weightings for the connections between the visual projection neurons and the Kenyon Cells, and a single boolean Extrinsic Neuron denoting image recognition. A modification was made by Zhang, whereby eight ENs were used, one for each of the cardinal directions in the Central Complex model. This will be discussed further in 2.3. The reader should note that the Central Complex (CX) model is primarily used to model the task of Path Integration and will not be discussed further (see [11]).

We would also like to look at methods for collision avoidance (CA) which do not involve specialised sensors such as a LIDAR or SONAR, the luxury of which, ants do not have. Models have been proposed which use Optical Flow (OF) properties to determine whether or not a collision is imminent. These models have been propsed both in purely robotic contexts [12], and biological ones [9].

1.2 Goals

The project aims for the following experimental scenario to be possible: We want to send the robot on a run through an obstacle course, allowing it to navigate however it chooses through the environment. From here, we want the robot to be able to replicate this route using only visual memories, which it should store on that initial run. Finally, we would like the robot to be able to navigate home following the reverse of this route. It should be noted that this final step is not strictly accurate to the behaviour of the desert ant. As noted by [3] in their familiarity-driven study of ant route navigation, *Wehner et al.* [15] demonstrated that the remembered routes have a distinct polarity, so knowledge of a route from nest to food, does not imply that the ant has knowledge of a route from food to nest. In this case, we make the outward and homeward route the

same.

The first stage of the project will focus upon obtaining a working collision avoidance system as a pre-requisite to gathering the route information. This CA system should be based on visual information readily available to AntBot with no additional/specialist sensors. For this paper, we assume that CA is a low-level reactionary behaviour, in that, we do not use any further processing of the detected motion; we react based on the immediate stimulus of the flow field. We will look at two different optical flow techniques used to build CA systems. We will also discuss the effects of using different types of flow field, how the different flow techniques behave in the same situation, and different methods of response.

We then move to the Mushroom Body circuit; this model for visual navigation was demonstrated by [2] in simulation. Previous iterations of this project which focussed on the Mushroom Body circuit have struggled to achieve similar performance on a robot using the methods originally presented by *Ardin et al.*; certainly, no previous iteration has dealt with a non-deterministic route through a cluttered environment. We aim to get the original *basic* model from [2] working on the robot in a robust fashion and hope to establish a baseline agains which future iterations can work while modifying the model. We will use a modified scanning behaviour to establish this baseline; ants have demonstrated use of scanning in visual navigation but it is generally accepted that this is not the primary method they use to determine a direction after having recognised a scene, rather, this scanning behaviour only occurs in certain scenarios (e.g. when the ant becomes lost) [8]. This will require debugging of the existing model and any interacting factors; for example, *Zhang* managed to achieve good results with the MB model using klinokinesis as the route following behaviour while a scanning struggled to produce robust routes. It should also be noted that the scanning behaviour performed poorly using a Perfect Memory model (the highest performing model used by *Ardin et al.*) which may suggest that the model is not all that contributes to the accuracy of recapitulated routes.

Finally, we will report the results of the experiments and testing performed at different stages during, and post development; we will compare these to relevant results from previous iterations of this project. We will end with a conclusion of our findings and contributions to the project, as well as discussing technical limitations and potential for future developments.

1.3 Results

This work is based on work done previously by Leonard Eberding, Luca Scimeca, and Zhaoyu Zhang [6, 11, 16].

Significant contributions:

1. An optical flow based system for Collision Avoidance,
2. Results indicating the impracticality of a time-to-contact based system for Collision Avoidance,
3. Implemented hardware upgrades to AntBot to make the platform more robust,
4. Implemented a new type of scanning behaviour for AntBot,
5. Successful replication of a route through a cluttered environment using Visual Navigation,
6. Implemented a basic but extensible statistical logging utility on AntBot with a complementary parsing and plotting utility written in Python.

2 Background

2.1 Optical Flow

Image flow is defined as being the 3D velocity vector of an object, projected onto a 2D image plane[10]. Optical flow is an approximation of this, working from a series of images to compute the projected velocity vector for a pixel. A single *flow vector* shows the displacement of a single pixel from one image to the next. A set of these flow vectors creates a *flow field*, a series of vectors which describe the motion in the complete image.

Broadly, there are two types of flow field: dense and sparse (also known as differential and feature-based repectively[9]). A *dense* flow field tracks the motion of every pixel in the image. A *sparse* flow field tracks the motion of a subset of pixels in the image. The sparse field may track a uniform subset of pixels, such as a grid, or a set of important points in the image, such as the prominent features in the image (object corners). See 4 for implementation.

2.2 Optical flow models for Collision Avoidance

Collision avoidance is an important component in navigation. Ants do not have dedicated sensory systems or the ability to create a visual 3D map of their environment using stereoscopic vision or motion paralax. It has been demonstrated that optical flow is used by honeybees in performing visual navigation [4], so it is not unreasonable to think that bees and other hymenoptera may use this information for other purposes. Indeed, optical flow has been shown to be a viable model for collision avoidance in *Drosophila* [13]. We visit two models in this paper which we will term the *time-to-collision* model and the *filtering* model.

2.2.1 Time-to-Collision

This model, proposed by *Low and Wyeth*[9], aims to replicate the CA capabilities of larger animals such as birds and humans. The model relies on accurately computing the *time-to-collision* or *time-to-contact* (TTC), which is the computed time until the robot collides with an obstacle. The TTC is computed as follows:

$$TTC = \frac{d}{v} \tag{1}$$

Where d is the distance from a point object on a collision course with the robot and v is constant speed at which the robot and object are closing.
Low and Wyeth then alter this equation by taking the direction of the velocity vector v and the direction to a point on the object and computing the angle between these two vectors ϕ. Their TTC equation then becomes:

$$TTC = \frac{cos\phi \times sin\phi}{\phi} \tag{2}$$

The TTC is then used to generate range information (image depth), which can be used to generate an appropriate reaction.

An alternative, yet equivalent, method for computing the TTC is given by *Souhila and Karim* wherein the time-to-contact is given in terms of the distance of all pixels from the *focus of expansion* (FOE); the point from which all flow vectors originate. *Souhila and*

Karim do not give a method of explicitly computing the FOE of a flow frame however, they do describe their method of estimating it. All flow vectors are in two dimensions with a horizontal (x) and vertical (y) component. Focussing on the horizontal case; flow vectors to the right of the FOE will have a positive x component, and flow vectors to the left of the FOE will have a negative x component. They then tally the signs of the horizontal components of the flow vectors computing a difference between the horizontal components on the left of the FOE, and the horizontal components on the right of the FOE. Intuitively, at the point where this difference is minimised, the divergence of the horizontal components is maximised; this gives us the horizontal component of the FOE. The vertical component is computed in a similar manner.

Finally, they compute the TTC as:

$$TTC = \frac{\Delta_i}{|\vec{V_t}|} \dots \tag{3}$$

Where Δ_i is the distance of a point $p_i = (x, y)$ from the FOE, and $|\vec{V_t}|$ is the translational velocity of the camera computed from optical flow[12].

This time-to-contact is then appropriately thresholded and a reaction is generated based on the magnitudes of the vectors surrounding the FOE. A "balance strategy" is then used as the control law by which the robot is directed through the environment; this strategy will be discussed in Section 4 as a modified version was used during development.

2.2.2 Filtering

The filtering method asks the following question: Given my current motion, what visual changes do I expect to see? Much of the following explanation was provided by [13]. A model proposed by *Stewart et al.* for CA in simulated fruit flies takes advantage of the fact that expanding patterns will trigger an avoidance manouver away from the focus of expansion[13]. Their model uses two offset flow filters (the *expected* flow). Each filter is constructed as a frontally centered expansion pattern with the same spatial extent to either side of the expansion pole (the central vertical axis of the pattern). These filters are then offset by $+3°$ for the right and $-3°$ for the left. The left and right filters feed into leaky accumulators, and if the accumulator exceeds a given threshold, a saccade in the opposite direction is triggered. The model from [13] is shown in Figure 1. We discuss a slightly modified version of this model in section 4.

2.3 The Mushroom Body for Visual Navigation

The Mushroom Body neuropils are structures present in the brains of all insects though they are largest in the brains of hymenoptera. They are known to play a critical role in olfactory learning, and have been thought to play a role in visual memory in hymenoptera since 1982 at the latest[1]. In 2016 a Mushroom Body circuit was proposed by *Ardin et al.* to allow emulation of the structure in a simulated desert ant [2]. The simulated ant's view is taken from 1cm above the ground and has a field of view of 296° azimuth by 76° elevation. A ratio of $4°/pixel$ is used to give a 19×74 pixel image. This image is then downsampled to 10×36 pixels to give a realistic resolution for ant vision. A 1×360 vector is used for further processing.

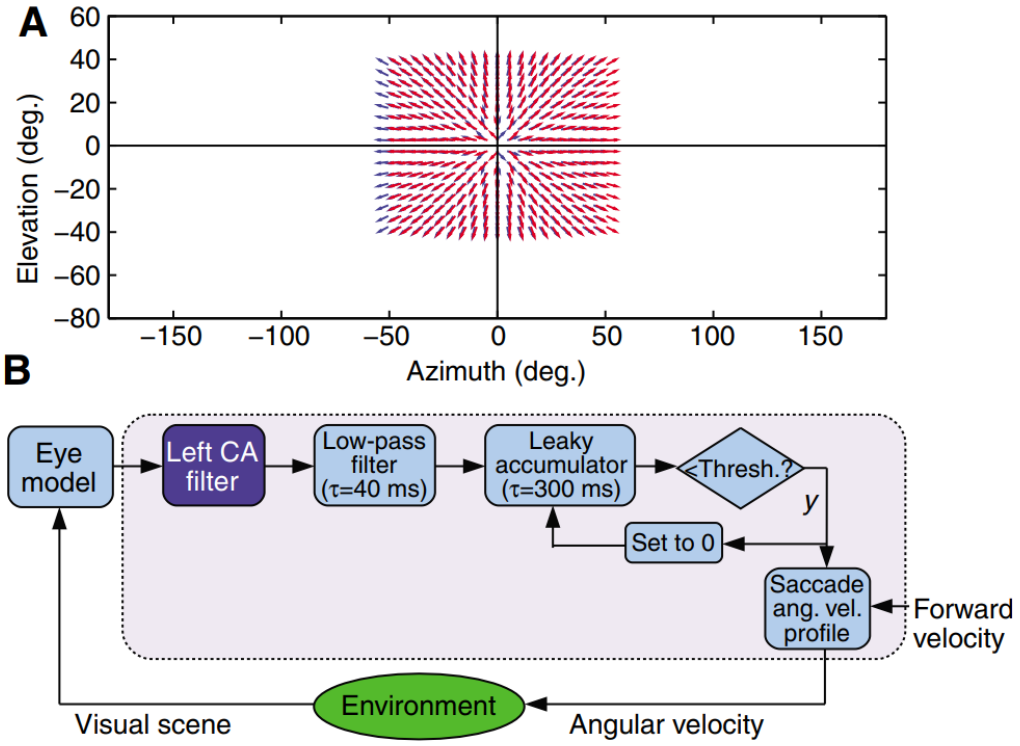

Figure 1: The OF filter model: (Caption from *Stewart et al.* Figure 7): Collision avoidance (CA). (A)The CA filters used in the model. Each covers 105 deg. of azimuth but they are centred at ±3 deg. (elevation = 0 deg.). (B) Control diagram for collision avoidance. Only the half of the system that triggers rightward saccades is shown for clarity; the other half has an identical configuration. The dark blue box represents the blue wide-field filter in A. The reset operation also applies to the other half of the system, i.e. a saccade in one direction sets both accumulators to 0. Thresh., threshold; ang. vel., angular velocity.

The generalised MB circuit is a three layer neural network: The first layer consists of a set of visual Projection Neurons (vPNs), these connect to the second layer of standard artificial neurons referred to as Kenyon Cells (KCs), and finally these KCs connect to a set of Extrinsic Neurons (ENs). The reader should note that any reference to the weight of a Kenyon Cell herein is specifically referring to the weight of the connection between that KC and the Extrinsic Neuron; this abstraction is made for ease of reference.

The model by *Ardin et al.* (shown in Figure 2) consisted of 350 vPNs (one for each pixel in the downsampled image). In the second layer we have 20,000 KCs each of which receives input from 10 randomly selected vPNs; each KC requires coincident input from multiple vPNs to fire. Every KC is then connected to a single EN which sums the number of KCs which are activated by the input image. The network is trained by providing a reward signal at regular intervals. If KC activation coincides with a reward signal, the connection strength to the EN is greatly reduced. The single EN simply gives a familiarity measure for the image seen. The agent decides on its next action by scanning to find the direction of greatest familiarity.

This version of the MB circuit has demonstrated the capacity to learn scene information, as well as recapitulate routes by using the scanning technique [2]. In 2016, *Eberding* implemented the Willshaw Network (WN) on AntBot, which resembles the

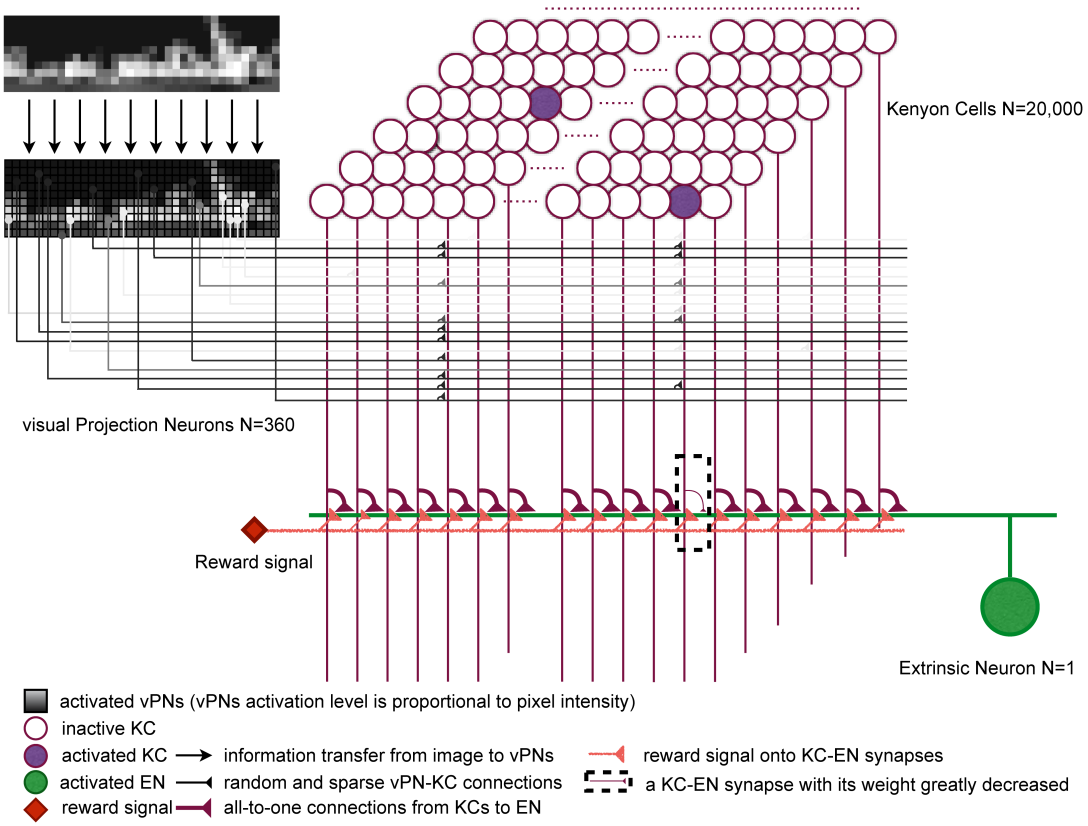

Figure 2: The Mushroom Body circuit: (Caption from *Ardin et al.* Figure 2): Images (see Fig 1) activate the visual projection neurons (vPNs). Each Kenyon cell (KC) receives input from 10 (random) vPNs and exceeds firing threshold only for coincident activation from several vPNs, thus images are encoded 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.

Mushroom Body neuropils (see 2.4); he demonstrated that the network allowed the agent to perform visual navigation through a sparse testing environment[6]. The agent navigates by scanning, computing unfamiliarity, and finally choosing the direction of minimum unfamiliarity. In order to save on computational resources, the KCs are binary rather than spiking (we hope to explore a spiking model in the future). While this does generate a correct route, it is not as continuous as those performed by real ants.

In 2017, *Zhang* implemented a route following strategy originally proposed by *Koszhabashev and Mangan* which employed klinokinesis in place of scanning. Klinokinesis is one of two main forms of kinesis - the movement of an organism in response to stimulus - in which the turning rate is proportional to stimulus intensity. In [16], the stimulus is given by the unfamiliarity metric generated by the MB model. The step size between each turning point as well as the turning angle depends on the unfamiliarity of the current view. The same algorithm from [8] is used to perform klinokinesis on the robot.

A separate model, also implemented by *Zhang*, added seven extrinsic neurons. Each EN is then used to represent one of eight directions relative to the robot's current heading; if we take 0° to be the robot's forward direction then the other seven directions correspond to $+45°$, $+90°$, $+135°$, $\pm 180°$, $-135°$, $-90°$, and $-45°$. This model was implemented as a way of combining the MB model for visual navigation and the CX model for path integration. While, path integration and the CX model are not explored in this project, the model is still worth discussing, as it may still be used to encode a desired response to specific stimuli.

2.4 The Willshaw Network

The Willshaw network is a specific style of neural network with an interesting background; capable of instantaneous learning, high capacity, and importantly, resembling the structure of the Mushroom Body inputs, storage layer, and outputs.

The idea behind the network stems from the effect of passing a pattern of light beams through two patterned pinhole cards and a lens to create a pattern of refraction on a screen. The pattern output on the screen is said to be a correlogram of the two patterns. Intuitively, the light pattern output at A can be passed through filter pattern B to produce pattern C. This idea is then refined by *Willshaw et al.* into a simple, three-layer neural network. As an aside, though the terminology of artificial neural networks is not used in the original paper, the model presented is designed to be interpreted as a biological neural network, so interpretation as an ANN is a case of semantics. The three layers of the Willshaw network are quite simple: a set of input neurons N_B are represented as parallel horizontal lines; a set of output neurons N_A are represented as parallel vertical lines; and finally the storage layer is a map of all of the intersections between the inputs and outputs. The storage layer is made up of artificial synapses which are initially inactive (weight zero). The synapses are activated (weight set to one) when the input pattern and output pattern activate the same synapse. More succinctly, synapse $c_{i,j}$ is set to 1 when a_i and b_j are activated simultaneously.
This style of network can be transformed into something which models the MB circuit presented by [2]. The input neurons N_b become the vPNs with multiple inputs going to each synapse. The synapses themselves become the KCs, though the weighting convention is reversed with zero being active, and one being inactive. We reduce the set

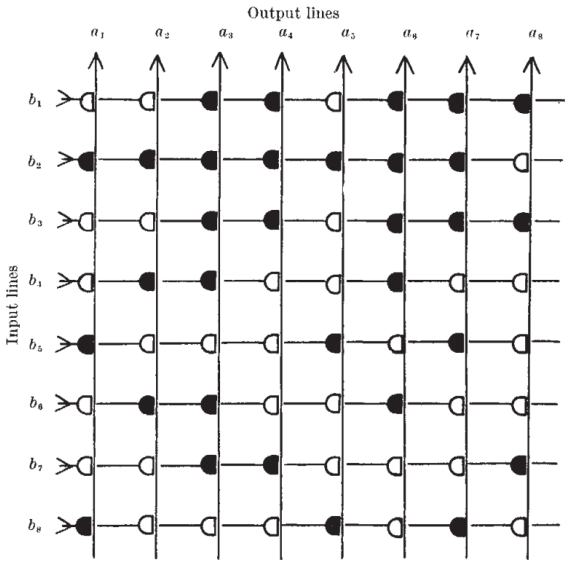

Fig. 4. An associative net.

Figure 3: The original network figure (Fig. 4) from *Willshaw et al.* [5].

of outputs N_a to a single line which represents our reward signal. Coincidence of a pattern of KC activation and a reward signal results in instant learning of that pattern by the learning rule set up in [5], just as in the MB circuit. Finally we must add the Extrinsic Neuron to the model as there is no parallel in the original Willshaw Net. As before the EN simply sums the weights of the KCs activated on presentation of an input pattern. Thus, the Willshaw net can be used to model the Mushroom Body circuit.

Figure 4: The Kogeto Dot 360° panoramic lens.

Figure 5: A sample of the view given by the lense before any processing.

3 Platform

In order to test the hypothesised models for ant navigation we use a simple robot autonomous - AntBot. AntBot was originally developed by *Eberding* in 2016; here we will discuss his design and implementation, upon which we develop our algorithms.

3.1 Hardware

AntBot's predecessor, Roboant, was originally designed by *Kodzhabashev* [7] as a compact Android robot. The robot required only four components: A sufficiently powerful Android phone (A Google Nexus 5 was used) as the brain, the Zumo Robot shield by Pololu as the chassis, an Arduino microcontroller to allow them to communicate, and finally a 360° camera attachment. AntBot uses the same basic structure, however, a Dangu 5 Rover chassis is used as the base, and therefore an alternate motor controller board had to be used.

The Android phone was chosen as the control module for the robot for a number of reasons. Firstly, the hardware; a modern smartphone allows a compact, powerful platform on which to build the software system as well as providing built in sensory systems and the libraries to use them (e.g. the camera). The Google Nexus 5 is more than capable of running image processing software, analysing optical flow patterns, and simulating the required artifical neural networks required for this project. Using an Android platform also allows for modular software design (See section 3.2.1). In order to mimic the near 360° field of view (FOV) given by the ant's compound eyes, we use a panoramic lense (the Kogeto Dot), which uses a convex mirror to give a full 360° FOV. This lense is attached to the front camera and requires some pre-processing to retrieve the desired 360×40 image. As with Roboant, the Android phone is connected to an Arduino using a serial interface. Commands are sent from the phone to the Arduino which then executes the relevant commands on the motor board to provide motion control.

3.2 Software

3.2.1 Android

The architecture of the Anrdoid operating system is such that applications can (subject to certain constraints) run in parallel while broadcasting important information to one another. This allows for a modular, ROS-like[1] system in which we can have a dedicated application for each navigational subsystem employed by AntBot.

In this fashion, *Eberding* implemented an Android Application Network (AAN) consisting of five applications:

1. The *AntEye* application - This application is the main application in the network and provides the user interface, along with all camera interaction and visual processing. To summarise; the visual processing system takes the 360° panoramic image from the camera, extracts the blue channel information, crops out the ring which contains the actual image and reshapes this into a $360 \times 40px$ image, and finally downsamples this to a $90 \times 10px$ image. This final image is then used by any application which requires visual information.
2. The *Path Integration* application - This application is responsible for performing all tasks related to Path Integration (PI). PI is the process of computing displacement based on a series of consecutive moves. In *Eberding's* original implementation this application did not perform PI, but was instead used as a utility application to record orientation and distance travelled. *Scimeca* extended this application to implement PI, using both a mathematical and neural approach.
3. The *Visual Navigation* application - Similar to the PI application, the VN application houses the necessary components for performing visual navigation tasks. *Eberding* implemented both the Willshaw (Mushroom Body) network, and also a Perfect Memory (PM) module both of which were extended by *Zhang*. There also exists a super-class for visual navigation algorithms.
4. The *Combiner* application - This application is used to combine the output from the

   VN and PI applications in order to compute what action the robot should take. This application governs the movement of the robot based on the two primary navigational systems.
5. The *Serial Communications* application - This application governs all communication from the Android phone to the Arduino and server-interface. Android forbids multiple applications from using a single serial or Wifi port, so this application was developed as an intermediary to allow the other applications in the network to communicate through a single application.

Eberding's implementation included a server interface which was used to control the robot remotely using the phone's Wifi hotspot and Serial Communicatin App, however, this interface has not been used since the original implementation. For more information, please see [6]. It should also be noted that, due to work conducted during previous iterations of this project, it may not be possible to follow this exact structure.

[1] Robot Operating System - ROS: http://www.ros.org/

Command	Message	Action
Heartbeat	x *seconds* n	Feedback sent to verify a stable connection between the phone and the Arduino. A signal is sent every second with a timestamp and checked.
Move	t 0 *distance* n	Travel a set distance in metres.
Turn	t 0 *angle* m 0 n	Turn a set angle (in degrees).
Turn and Move	t *angle* m *distance* n	Turn by a specified angle then move the specified distance.
Turn left	*l*	Turn left indefinitely.
Turn right	*r*	Turn right indefinitely.
Halt	*h*	Stop any command in progress and stop the robot.
Go	g *leftSpeed rightSpeed* n	Move indefinitely with specified left and right speeds.

Table 1: The available commands on the Arduino and the messages sent to invoke them. Those values which are changeable are shown in italics. The *go* command was added by *Scimeca*. This table was adapted from [6, 11].

3.2.2 Arduino

The Arduino software can be split into two sections: the *parser* and the *executioner*. The parser will receive commands from the serial port and convert them into a series of movement commands. These movement commands are then sent to the motor board by the executioner. Encoder information may be gathered by the Arduino and sent back to the phone for processing. For this project, the Arduino code has not been modified. We only required the use of two commands for this project; *go*, which allows the robot to move indefinitely at a set speed, and *turn*, which allows the robot to turn to a desired (relative) angle. The *go* command also allows the operator to specify a speed for the left and right sides allowing the robot to move in smooth arcs. A full list of commands can be seen in Table 1.

Previous works have mentioned a message of the format *e e1 e2 e3 e4* n. This message was used to send wheel encoder information from the Arduino to the phone, however, this message was only sent during the execution of a particular function and it has since been removed from the Arduino code. There are currently no utilities available to retrieve and reset encoder values on-demand from the phone, though such utilities should not be difficult to implement.

3.3 Modifications

Some hardware modifications have been made to the robot. Upon the uptake of this project, AntBot was powered by six 1.2V AA NiMH batteries wired in series (using a simple power pack). This power pack was connected to the motor board (and subsequently the motors) by a pair of 9V connectors. The robot had to be opened up in order to connect or disconnect the batteries from the motors. The batteries themselves had to be extracted from the power pack and charged individually. These actions, repeated often by the current project and predecessors had caused wear on the power pack and the internal wiring. The wires themselves needed re-soldered multiple times. In order to have a more robust platform for testing, the on-board power system was modified. The aim of the modification was to add a power source which could be

charged in-place. As such a new power source which could be charged as required without removal was added, along with external charging ports and a switch. The internal wiring was re-done such that power could either flow from the external ports to the battery, or from the battery to the motors controlled by the external switch. The new power source is a 9.6V NiMH power pack. Motor speeds and systems which depended on them required re-tuning after completion of the modifications.

4 Methods

4.1 Optical flow models for Collision Avoidance

4.1.1 Time-to-Collision

We attempted to follow the outline presented by *Low and Wyeth* as, at first glance, it seems a straightforward and neat solution. However, this method of computation for the TTC relies on the assumption that the camera view only ever moves in the forward direction which is not appropriate for AntBot (See Section 3). So we look instead at the paper by *Souhila and Karim* for an alternative; for this method, we must compute the *focus of expansion* or FOE, however, their method of computing the FOE again relied on forward motion of the camera and did not translate well into AntBot. A more general method for computing the FOE is given by [10]:

$$FOE = (A^T A)^{-1} A^T \mathbf{b} \tag{4}$$

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

Where, for each pixel $p_i = (x, y)$, the associated flow vector is given by $\mathbf{v} = (u, v)$. We then set $a_{i0} = u, a_{i1} = v$ and finally $b_i = xv - yu$. The TTC can then be computed as:

$$TTC = \frac{d}{v} = \frac{y}{\frac{\partial y}{\partial t}} \tag{5}$$

Where y is the vertical distance of some point $p = (x, y, z)$ from the FOE, $\frac{\partial y}{\partial t}$ is the velocity of translational motion of y, and d and v are as given in (1). A full derivation is given by *O'Donovan* from whom we have adapted this equation.

Finally, we can simplify this to the desired equation from [12]:

$$TTC = \frac{\Delta_i}{|\vec{V_t}|} \dots \tag{6}$$

Where Δ_i is the distance of a point $p_i = (x, y)$ from the FOE, and $|\vec{V_t}|$ is the translational velocity of the camera computed from optical flow[12].
This time-to-contact is then appropriately thresholded and a reaction is generated based on the position of the focus of expansion which provided a simple method of replicating the balance strategy used by [12] as the FOE will be drawn to one side based on the motion parallax observed. Initial instability in the computation of these properties was countered with by averaging the properties across a series of frames to reduce potential noise.

The generated responses were to be halt on detection (for debugging purposes), and triggering of a smooth turn away from the obstacle. This technique was designed to be used with a sparse optic flow field, however, a version using a dense flow field was attempted. The computation across a dense flow field was found to be impractical. No formal results have been collected using this system; any apparant functionality was found to be luck instead of a working system. Analysis of functionality of such a system on AntBot is left for Section 5.

4.1.2 Filtering

Here a dense optical flow field is used, and in order to explain the filtering process, it is important to discuss the flow computation itself. The optical flow is computed using openCV's *calcOpticalFlowFarneback()* function. When given two consecutive frames the function returns a matrix M of arrays of type double. $M_{y,x}$ is a one-dimensional array of length two which contains the displacement in each axis of the respective pixel (i.e. $M_{y,x}[0]$ gives the displacement of the x coordinate, and $M_{y,x}[1]$ gives the displacement of the y coordinate.

An optical flow filter was previously implemented by *Scimeca* for the purpose of speed retrieval. As collision avoidance in optical flow relies on detecting a difference in motion between two sides the same filtering method could be re-purposed for collision avoidance. Though inspired by [13], our method for performing the computation and avoidance was modified slightly for the sake of implementation on AntBot. Instead of creating a left and right filter, we create left and right flow frames by taking rotating the image frame by a certain angle left or right as in [11]. However, we reduce the angle from $\pm 45°$ [11] to $\pm 16°$. It was found that the size of this angle had a mild effect on the region of the image which would trigger responses; a larger angle would result in higher sensitivity to the sides of the robot, a smaller angle results in higher frontal sensitivity which was desired for the CA system. We only want the robot to avoid obstacles to the front, obstacles to the sides should not trigger responses. The offset angle was tuned manually and $\pm 16°$ was found to perform well; smaller angles resulted in an insensitive system, and larger resulted in more noise, though performance was still good. The filter itself is the same as the second filter from [11] which was already implemented on the robot.

The filter is constructed as an $Nx3$ matrix F such that:

$$F_i = [\quad sin(-\pi + i\frac{2\pi}{N}) \qquad 0 \qquad 0 \quad] \quad for \quad i \in \{0, N-1\} \tag{7}$$

where F_i is the ith row of F. Intuitively each row corresponds to a pixel value in the x axis.

Once the filter is computed, we retrieve the flow for each pixel by adding the displacements retrieved from *calcOpticalFlowFarneback()* to the pixel value. For each value of x we apply an offset of $\pm 4px$ which corresponds to our left and right frames centred at $\pm 16°$. For the ith pixel, with filter vector $\bar{F_i}$ and flow vector e$\bar{O_i}$ we apply the filter simply by computing the projection:

$$P_i = \bar{F_i} \cdot \bar{O_i} \tag{8}$$

This allows us to measure the geometric difference between the expected and actual pixel motion. The computation for a single frame simply sums all of these projections:

$$FlowSum = \sum_{i=0}^{K-1} P_i \tag{9}$$

where K is the number of pixels in the image, in our case $K = 900$. This sum is computed for both shifted frames and a tuning factor is applied for the simple purpose of making the numbers returned easier to handle.

Now we have defined a way of computing speed differences between the two sides, we can start to define behaviour. In the case an obstacle is seen on one side of the image, we expect the speed on that side to be higher due to motion paralax. Thus, we simply compute the difference between the two sums:

$$FlowDifference = LeftFlowSum - RightFlowSum \tag{10}$$

which gives us our final stimulus to be used to generate a reaction. Here we employ the idea of "leaky" accumulators from [13]. An accumulator is kept for both sides of the frame. The raw difference is thresholded to account for noise such that only significantly positive or negative values contribute to behaviour; this is known as the *accumulation threshold*. Once the accumulator on one side exceeds its *reaction threshold*, an immediate turn away from the perceived obstacle will be performed. The turn is simply a $\pm 20°$ rotation using the robot's *turnAround()* function. The accumulators are reset periodically to avoid a build up of old flow information. A separate method of accumulation was trialled out of interest whereby a single accumulator would be fed the raw difference and a sufficiently large positive or negative value would trigger the appropriate response. No difference was noticed in performance and so separate accumulators were kept. While this implementation was simple, and not as refined as we would have liked, it did demonstrate impressive performance in navigating arenas of varying density.

4.2 Debugging the Mushroom Body

The Mushroom Body circuit (implemented as a Willshaw Net) was originally implemented by *Eberding* when he initially developed the platform. Though the implementation does mirror the model of *Ardin et al.*, the model struggled to perform to the same degree despite potentially having more visual information due to the higher resolution image of AntBot. The nature of our experimental scenario required (at least initially) that a scanning behaviour be used to perform visual navigation and this is the context in which we shall frame our discussion; this because *Zhang* did manage to achieve significantly improved results using Klinokinesis as a route following strategy. Initial attempts at scanning resulted in extremely poor performance with seemingly no capacity for memory present in the network beyond blind luck. The first and most obvious issue was the scanning behaviour itself; while the robot was to turn through a $60°$ arc in increments of $6°$, the actual arc was far larger ($> 100°$). On closer inspection of the control code on the Arduino, the problem could be seen as the control method. The turning command relies on a calculation based on encoder clicks per degree; while simple and appropriate for general use, this style of control generally struggles to make precise movements, such as scanning increments.

To remedy this, we opt for a different style of scanning which we will refer to as *visual scanning*. Since AntBot has a $360°$ view, we instead take a still image, and rotate it, showing each rotated frame to the network and receiving a familiarity measure. These values are stored in an array (as with standard scanning), the minimum value is selected and the angle computed by taking the distance from the centre-point of the array and multiplying by the number of pixels traversed horizontally then multiplying again by the constant 4 (degrees/pixel). In the first iteration, a highly granular scan was used, however, this was found to be less effective and more prone to erroneous familiarity measures. Reducing the number of image angles while scanning over the

same arc resulted in more stable performance. In total 17 different angles (from $-16px$ to $+16px$ in increments of $2px$) are compared.

4.3 AntBotStats

5 Results and Evalutation

Included for skeleton purposes.

6 Discussion

Included for skeleton purposes.

References

[1] The brain of the honeybee apis mellifera. i. the connections and spatial organization of the mushroom bodies. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 298(1091):309–354, 1982.

[2] 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.

[3] Bart Baddeley, Paul Graham, Philip Husbands, and Andrew Philippides. A model of ant route navigation driven by scene familiarity. *PLOS Computational Biology*, 8(1):1–16, 01 2012.

[4] Laura Dittmar, Wolfgang Stürzl, Emily Baird, Norbert Boeddeker, and Martin Egelhaaf. Goal seeking in honeybees: matching of optic flow snapshots? *Journal of Experimental Biology*, 213(17):2913–2923, 2010.

[5] O.P. Buneman D.J. Willshaw and H. C. Longuet-Higgins. Non-holographic associative memory. *Nature*, 1969.

[6] 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.

[7] Aleksandar Kodzhabashev. Roboant: Build your own android robot, 2014. https://blog.inf.ed.ac.uk/insectrobotics/roboant/ *(Last visited 22/01/2018).*

[8] Aleksandar Kodzhabashev and Michael Mangan. Route following without scanning. In Stuart P. Wilson, Paul F.M.J. Verschure, Anna Mura, and Tony J. Prescott, editors, *Biomimetic and Biohybrid Systems*, pages 199–210, Cham, 2015. Springer International Publishing.

[9] Toby Low and Gordon Wyeth. Obstacle detection using optical flow. In *in Proceedings of the 2005 Australasian Conf. on Robotics & Automation*, 2005.

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

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

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

[15] Rüdiger Wehner, Martin Boyer, Florian Loertscher, Stefan Sommer, and Ursula Menzi. Ant navigation: One-way routes rather than maps. *Current Biology*, 16(1):75 – 79, 2006.

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