The University Of Sheffield.

A VISION BASED STUDY ON COLLECTIVE BEHAVIOUR IN MAMMAL-LIKE ROBOTS

by

Matthew T. Whelan

Supervisor - Prof. Tony J. Prescott

A dissertation submitted in partial fulfilment of the requirements for the degree of *MSc in Computational Intelligence and Robotics*

Department of Psychology
Department of Automatic Control and Systems Engineering
University of Sheffield

August 2017

Abstract

Collective behaviours are varied yet ubiquitous both within groups of biological and robotic agents, with a vast amount of these behaviours requiring that the agents can recognise their own kind. Presented in this dissertation are two vision algorithms that have been implemented in a biomimetic robot named MiRo, in order that MiRo may recognise another MiRo using its two front facing cameras. The first vision algorithm utilises the histograms of a greyscale image space and inputs them into a perceptron neural network that is pre-trained off-board. The second vision algorithm adopts the SURF process combined with Bayesian inference for estimating a probability of an image space containing a MiRo. Vision algorithm 1 achieves satisfactory classification rates of 90.1%, whilst algorithm 2 manages to compute significant probabilities correctly 87.8% of the time. The most contrasting difference between the two algorithms is the relative time taken to run each, with algorithm 1 performing 313 times faster than algorithm 2, due predominantly to algorithm 2's long computations when solving for Bayes' inference. Subsequent development was then conducted for a MiRo-MiRo following strategy, with the vision algorithms enabling MiRo's to follow each other. This success proved collective behaviours requiring MiRo recognition are now feasible.

In loving memory of my cousin, one of my best friends

Contents

List of Figures

List of Tables

Nomenclature

Roman

b	Baseline distance for stereo lenses (m)
C	Centroid of a thresholded image space
$\mathbf{c}$	Correlation matrix
D	MiRo wheels track distance (m)
$D(\cdot)$	SURF approximation of $L(\cdot)$
d	Euclidean distance between two vectors; Haar-wavelet response
e	Error term
f	Focal length of lens (m)
G	Gaussian function
$H(\cdot)$	Hessian matrix; Object class
$H(n)$	Modified logistic operator as function of n
$h(\mathbf{x})$	Classifier for the input vector $\mathbf{x}$
I	Pixel intensity
$ii(\cdot)$	Integral image
$\mathbf{K}$	Kinematic controller gain matrix
k	Constant used in Sauvola's threshold formula; Logistic operator constant
$L(\cdot)$	Second order Gaussian derivative convoluted with an image space
M	Image space moment
$m(\cdot)$	Mean image space pixel intensity
n	Number of thresholded image pixels

R	Normalising term used in Sauvola's threshold formula
r	MiRo wheel radius (m)
S	MiRo forward wheel speed (m/s)
$t(\cdot)$	Threshold value for pixel intensities of images
V	Velocity (m/s)
$\mathbf{v}$	SURF feature descriptor vector
$\mathbf{w}$	Weight vector in the perceptron ANN
w	Weight in approximated Hessian determinant with SURF
$\mathbf{x}$	Single image histogram vector
y	Output in the perceptron ANN

Greek

α	Angle in polar coordinate frame (rad)
β	Threshold in the perceptron classifier
γ	Wheel speed controller constant
η	Learning rate
θ	MiRo global angle position (rad)
ρ	Radius in polar coordinate frame (m)
σ	SURF scale
$\sigma(\cdot)$	Standard deviation of image space pixel intensities
ω	Angular velocity (rad/s)

Abbreviations

(A)MAS	(Adaptive) Multi-agent System
ANN	Artificial Neural Network
(D)DOF	(Differential) Degrees of Freedom
SURF	Speeded-Up Robust Features
TMiRo	Target MiRo. The target MiRo is the MiRo that the current MiRo is aiming to follow

Acknowledgements

My warmest thanks are offered to my supervisor, Professor Tony Prescott, for the constant support provided throughout, and for the opportunity I have had in working with a truly exciting development in the MiRo robot. I'd also like to extend my gratitude to Dr Ben Mitchinson of Consequential Robotics, for whose support was so crucial in the progress of this project, and to all else at the Sheffield Robotics lab whose help has in no small part contributed to the success of this project.

To Dr Roderich Groß are my kindest of thanks given, whose lead and organisation in the MSc Computational Intelligence and Robotics has resulted in the course being a yearlong source of intellectual challenges and inspiration.

Finally, I shall be enduringly grateful for having being awarded the John Beadsmoore Sheffield Postgraduate Scholarship, which has provided me with the opportunity to dedicate the entire year to my studies without financial concerns. For this my only means of repaying such kindness has been and always will be to dedicate many of my efforts and abilities to the endeavour of scientific understanding and discovery, in the hope that many more will benefit from such undertakings.

Seeing is not a direct apprehension of reality,
as we often like to pretend.
Quite the contrary:
seeing is inference from incomplete information...

E. T. JAYNES, 2003

Chapter 1

Introduction

1.1 MiRo

MiRo was developed as a biomimetic robot, with the intention of serving two primary purposes: the first is in its use as a companion robot (Collins, Prescott, and Mitchinson, 2015a); the second is for entertainment and education (an edutaintment device) (Collins et al., 2015b). It therefore serves as a useful platform for the study of mammalian behaviour in robots, and as such is the platform on which this project is conducted. MiRo can be seen in Figure 1.1.

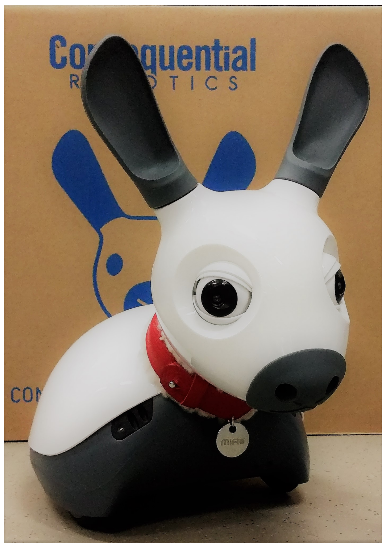


Figure 1.1: The MiRo robot

There are numerous robots in existence that are based upon biological animals, from commercial robots such as the PARO seal (Wada et al., 2003) used primarily as a human-robot interaction device for psychological treatments and FESTO's BionicKangaroo (FESTO, 2017) developed to recreate complex animal locomotion, to research-only based robotics such as for the study of early life forms (McInroe et al., 2016). And this is no surprise, considering that nature has had over 3 billion years to produce mechanisms far more sophisticated and able than the current offerings in robotics and AI. Hence, there seems to be no better place to take inspiration for the design of robots than from nature itself.[1]

[1]There has been made a strong argument that robots should not be restricted to the capabilities of

1.1.1 The Control Architecture

The MiRo attempts not only to mimic mammalian forms in its appearance, but to also mimic the biological control architecture too (Mitchinson and Prescott, 2016). There are three levels of control implemented in the MiRo, each with their own dedicated processors and memory. Each level is intended to replicate the three primary levels of processing that occurs in mammals, that is:

1. Spinal cord – the lowest level of processing. This level has fast execution, but a low level of complexity and minimum amount of memory. This level of processing is intended for executing signal conditioning on signals received from its sensors, and for reflex behaviours, such as a "freeze reflex" that halts/inhibits MiRo's motions, sounds and LEDs if triggered.

2. Brainstem – a mid level processor. This level can complete more complex tasks than the spinal cord and also has a greater amount of memory, but has slightly slower execution time. Most of the core processing tasks are performed at this level, including action selections, management of affective states and of actuator motor patterns, amongst others.

3. Forebrain – the highest level of processing. This level has the slowest execution time, but can perform the most complex tasks. There is an indefinite amount of memory available at this level, with the only restriction being the size of the SD card used. This level performs tasks such as mapping and even abstraction. Perhaps the most exciting development available for this level is the inclusion of a basal ganglia model, shown to effectively act as a central selection mechanism (Gurney et al., 2004).

The reader is referred to (Mitchinson and Prescott, 2016) for a fuller account of the control architecture found within the MiRo.

1.1.2 Sensors and Actuators

MiRo is equipped with a variety of sensors, both exteroceptive and interoceptive, and actuators as shown in Table 1.1. All data is taken from the MiRo developer kit (Con-

animals/humans, and for non-biomimetic robotics this has some validity. But for a biomimetic robot, by definition, this argument is redundant.

sequential Robotics Ltd, 2016). Only sensors and actuators with some relevance to the project are reproduced here, and hence is not a complete list.

Table 1.1: MiRo's sensors and actuators

Sensors		
Sensor	*Quantity*	*Further Specs*
Microphones	2	10-bit @ 20kHz
Cameras	2	128x96 @ 25fps; 192x144 @ 25fps; 320x240 @ 8fps
Proximity	1	Sonar in the nose (30–300+ mm)
Light	4	Distributed about body skirt
Actuators		
Actuator	*Quantity*	*Further Specs*
Drive wheels	2	Differential drive; max forward speed of 400mm/s
Body joints	3	Lift, yaw and pitch
Tail	2	Wagging and drooping
Ears	2	Each ear rotates independently
Eyelids	1	Eyelids open/close together
Sound output	1	Digital stereo audio @ 8kHz 10-bit

1.2 Motivation

MiRo currently does not have the ability to recognise another MiRo, either visually or through other sensing/communication strategies. Yet it has been shown in other work that conspecies recognition is important in the survival of mammals, for keeping stable relationships, recognising mates and offspring, as well as other aspects of social behaviour (Hurst et al., 2001). In light of the importance of biological conspecies recognition in the animal kingdom for certain types of collective behaviours, it is therefore deemed necessary that MiRo should be able to at a minimum recognise another MiRo – a form of "conrobotic recognition", or "MiRo-detection". Developing MiRo's vision system to allow it to recognise other MiRos would allow interaction amongst MiRos, which would enable the development of a wider range of collective behaviours.

Further, once "MiRo-detection" is successful, this new ability will be used in order to develop a basic form of collective behaviour – a *following*[2] strategy, allowing one MiRo to detect, track, and then *follow* another MiRo. It is thought that successful implementation of a *following* strategy will form the basis upon which more complex collective behaviours can be explored, such as herding and flocking.

[2]Henceforth italicised for clarity.

1.3 The Challenge

One of the most basic forms of collective behaviour is that of one agent *following* closely behind or alongside another agent, and this is the primary challenge for this project. The problem of one agent *following* another agent (the target agent) can be broken down into perhaps three sub problems: (1) the agent needs to be able to detect and then track the target agent; (2) the agent must be able to determine the distance between itself and the target agent; (3) using the above information, the agent must then have a control strategy that computes the necessary velocities in order to achieve effective *following* of the target agent.

Therefore, in order to accomplish *following* behaviour in MiRo, each of the three problems above must be dealt with individually. These then form the primary sub challenges of this project.

1.4 Aim and Objectives

Aim

To develop *following* behaviour in MiRo by addressing each of the three issues above, through using MiRo's two front facing cameras and a behaviour-based control approach.

Objectives – Basic

- Review literature on collective behaviour in robotics and in mammals/animals.
- Review literature on object detection techniques used in computer vision.
- Apply two or three detection algorithms to achieve "MiRo-detection".
- Statistically compare the generated algorithms and select one.
- Incorporate MiRo's current motion detection ability with the "MiRo-detection" algorithm to track MiRos.
- Develop a MiRo-MiRo following technique and perform simulations.
- Test the final controller on the physical robot to prove feasibility.

Objectives – Advanced

- Formulate more advanced collective behaviour, such as co-operative foraging and/or herding.

1.5 Overview of the Thesis

Chapter 2 presents a review of the literature associated with collective behaviours, exploring the wide range of collective behaviours found both from within biology and robotics.

Chapter 3 then turns to the literature associated with vision, starting with aspects of vision in biology/psychology and followed by a survey of computer vision techniques used for object detection, motion tracking and object depth estimation.

Chapter 4 builds on the vision techniques discovered in Chapter 3 to build two custom "MiRo-detection" algorithms in order to detect a MiRo; the first using image grey-scale histograms and a perceptron neural network, the second using a vision algorithm named SURF combined with Bayesian estimation.

Chapter 5 then investigates mobile robot control strategies, starting with a review of a classical control method, after which a survey of behaviour-based robotics is presented, including reactive controllers and adaptive behaviours. The chapter then progresses to develop a custom MiRo control law that allows MiRo to follow an object it detects from its vision algorithms.

Chapter 6 presents the results of some minor testing performed on the real-world MiRo in order to display the feasibility of the developed controller, showing that the optimised controller works well.

Chapter 7 will then bring the entire work coherently together with a discussion on the combined use of the vision algorithms and the controller. Further, a proposal for an adaptive control strategy is offered, followed by a review of the completion statuses of the project's aim and objectives. The chapter closes by concluding the thesis and offers areas for additional work.

Chapter 8 gives a brief description of the project's management before going on to give a short self-review.

Chapter 2

Collective Behaviours – A Review

Collective behaviour is a varied field that can have various different meanings. The purpose of this literature review is to first briefly describe what collective behaviour means in the context of biological animals, by reviewing literature from within the fields of zoology, biology and other related fields.

Collective behaviour is then studied in the context of robotics, which is more closely related to the work conducted as part of this project. A discussion on the current state of robotics collective behaviour is presented and contrasted with the types of collective behaviour found in animals.

It is worth noting here that more in-depth technical reviews on biological/computer vision and *following* control strategies are given in Chapters 3 and 5 respectively.

2.1 Collective Behaviours in Animals

Mammals and other animal types exhibit collective behaviours for various survival and reproductive purposes, and is found in examples such as insect swarms, bird flocks and mammal herds (Okubo, 1986).

Gautrais et al. (2008) discovered a density dependent-effect that altered the individual behaviours of fish in a shoal, suggesting that the behavioural states of fish alters as the group size changes. The exact mechanism of how group density affects the fish behaviour, such as whether the fish themselves are aware of the group size and thus change their behaviour in response, or whether their behaviour is as a result of information cascaded throughout the shoal therefore population density indirectly affecting behaviour, remains unknown.

King et al. (2012) conducted a study on sheep herd behaviour, and specifically their behaviour when under threat by a predator. They were able to apply statistical methods to discover the motion of the sheep. Results showed that the sheep would move towards the centroid of the group when threatened by a predator (in this case a sheepdog). The sheep would instinctively move towards the centroid without knowing explicitly where the centroid is – their behaviour was therefore based solely on local sensory information, and not the global pattern of the group. One may note that the behaviour exhibited by the sheep is described as part of the selfish herd theory presented by Hamilton (1971).

Okubo (1986) was one of the first to apply techniques to mathematically model the dynamics of generic animal grouping behaviour, and therefore attempts to quantify aspects of collective behaviour. Mathematical models of grouping behaviour is important particularly for developing collective behaviour in robots.

Ballerini et al. (2008) reviewed a number of other papers interested in modelling collective behaviour, and proposed that in all cases, the models agree on three behavioural traits: agents move in the same direction as their neighbours; they keep close to each other; they avoid collisions. The authors also presented large-scale data on the features of starling flocks, including shape, movement, density and structure, and showed that all these features are emergent properties.

Sumpter (2006) argues that collective behaviour is best understood by identifying the behavioural algorithms that the individuals follow, whilst also understanding how and what information is transmitted between the animals. Again though, the emphasis is on the individual and the behaviour of the individual, with the collective behaviour being an emergent property.

Communication for effective transmission of information therefore is often necessary for organising collective groups. One example of this is in pigtail macaques (Maestripieri, 1996), which exhibit a wide range of physical gestures in order to communicate. Examples include bared-teeth and lip-smacking to indicate submissiveness, nonthrusting mounts to indicate dominance, and puckering, which was found to be the most frequently used gesture, which acted to summon another macaque. Even frogs have been shown to exhibit a wide range of visual communication signals, such as toe flagging, body jerking and limb lifting (Caldart, Iop, and Cechin, 2014).

Finally, it's worth noting one study in relation to *following* behaviours in animals, and baby chicks are a useful example here. It was observed early on that baby chicks will form a memory of the first moving object they're exposed to, and will follow the object for some time after (Lorenz, 1937). Although the chicks respond to visual and auditory cues from the mother hen, they will also respond to a non-auditory moving visual object. This process is better known as visual imprinting (Maekawa et al., 2006).

2.2 Collective Behaviours in Robotics

Cazenille, Bredeche, and Halloy (2016) make note that there are different levels of description available for modelling collective behaviour, from the macroscopic level, which usually deals with the global state of the system and is typically described using ordinary differential equations, down to the microscopic level, which deals with individual's states and are often represented using Finite State Machines (FSM).

Collective behaviour in robotics tends to be analysed in terms of models and algorithms, and run in simulation environments, rather than on physical robots. One example of this is Calvao and Brigatti (2014), who develop an algorithm that aims to achieve cohesion within swarms, using a method in which agents seek an ideal distance between itself and its neighbours. They then applied the algorithm to the "selfish herd problem", with some success (as studied by King et al. (2012) in sheep herding).

A useful outcome in the use of collective robotics is in self-organisation tasks with biological agents. An example of this is given by Halloy et al. (2007), in which robots were integrated with living cockroaches and were able to affect the group decisions which allowed the robots/cockroaches to organise equally across two shelters. The authors offer a set of differential equations to describe the dynamics of the robots and cockroaches.

Isaeva (2012) relates the self-organization of biological systems to that of any other type of network capable of self-organizing, and notes five points important for a network to self-organize – feedback; stability; flexibility; modularity; and hierarchy. Note that hierarchy could give the impression that agents in the group must not be homogeneous, but this doesn't necessarily have to be the case – hierarchy could be a consequence of external factors such as agent positions, and could change at any moment.

Trianni and Dorigo (2006) developed self-organization of robots and compared varying robot communication strategies to achieve the self-organization. These included no direct communication, hand-crafted signalling and evolutionary communication. Results showed that the evolutionary approach was the most efficient for the robots to achieve their self-organization activities, supporting the view that evolutionary approaches are superior to conventional design methods.

Self-organisation is popular in the field of multi-agent systems (MAS), as discussed in great detail by Di Marzo Serugendo, Gleizes, and Karageorgos (2005). The authors apply a definition to the term "self-organization":

> Self-organization is . . . the mechanism or the process enabling a system to change its organization without explicit external command during its execution time.

In effect, the above is saying that the self-organization process of MAS should be entirely automated. Additionally, a self-organizing system can be implemented using decentralized control and is dynamic (system evolves in time).

Picard and Gleizes (2003) extend the theory of MAS to Adaptive Multi-Agent Systems (AMAS), and apply it to a robot transportation problem. They develop a non-cooperative situation detection module (NCS detection module), which seeks to prevent agents from performing tasks that are non-cooperative. Non-cooperative tasks must be determined by the user and for each level of the system (robot level, state level and activity level).

Communication is ubiquitous and highly useful in collective agents, whether biological or artificial, and can be divided into three classes: indirect communication (stigmergy), direct communication and direct interactions (Trianni and Dorigo, 2006).

There is a line of study found in literature related to adaptive communication, or the ability for agents to evolve communication strategies (Marocco, Cangelosi, and Nolfi, 2003; Nolfi, 2005; Althnian and Agah, 2016). The usefulness in the evolutionary approach is that the agents develop the minimum amount of communication necessary for them to achieve their goal, without the need for the user to invoke extraneous communication information or to restrict the agents to a less efficient communication strategy.

Communication is not always necessary to achieve collective behaviours, and non-communication strategies can sometimes be more ideal. Kube and Zhang (1993) take inspiration from the social behaviour of insects in order to develop non-communicative collective robotics. They note that although agents with the ability to communicate should be more effective, scaling up the number of agents becomes a problem with communicative approaches, and non-communicative approaches simplifies this. Again, emergent behaviour is a clear theme. In later work Kube and Zhang (1996) apply their ideas to the challenge of having multiple robots move a box. One robot alone does not have enough power to move the box, hence they must cooperate to achieve the goal. Though the algorithms implemented in each robot makes no mention of working with other robots, their behaviour emerges in a cooperative manner and they successfully accomplish the task collectively.

Rodríguez, Gómez, and Diaconescu (2015) implement a foraging strategy on a multi-bot system that has each agent contain their own personal internal maps. Information between the agents is shared via *Trophallaxis*, a technique in which agents exchange information between each other when both occupy adjacent locations. This approach is found in nature, and generates a trophallactic cascade scheme. The study of trophallaxis is primarily interested in how food/nutrients or pheromones are transferred between social insects (Suárez and Thorne, 2000), but one could easily extend this phenomenon to robotics, with the transfer of information, as performed by Rodríguez, Gómez, and Diaconescu (2015), or even transfer of energy. Ngo and Schiøler (2008) design such an energy trophallactic system, and argue that true autonomous machines should not just be behaviourally autonomous, but energetically autonomous too. Hence their robot design allows agents to autonomously seek energy sources but also transfer energy between each other. In fact, their design involves the exchange of physical batteries between the bots – a necessarily precise and energetic task, but one that overcomes other difficulties involved with wireless charging such as efficiency loss and charging times (Wang and Wei, 2015).

The above used a foraging strategy in which the agents each held individual internal maps of the environment. Burgard et al. (2000) use collaborative robots to increase the efficiency of exploring unknown environments by using a global map. Occupancy grid maps are utilised, and the following formula is used for when the robots build up an internal map of the environment,

$$\frac{1}{P(occ_{x,y})} = 1 + \prod_{i=1}^{n} \frac{1 - P(occ^{i}_{x,y})}{P(occ^{i}_{x,y})} \tag{2.1}$$

The above describes the probability of some space with coordinates $< x, y >$ in the global map being occupied by an object/wall, that is $P(occ_{x,y})$, in terms of the individual robot's probability of the $< x, y >$ coordinate being occupied by an object, that is $P(occ^{i}_{x,y})$, whilst n represents the total number of robots. Hence, the robots are collaborating to build a more accurate global map for an unknown environment. The authors then introduce a cost function for each robot exploring a frontier cell (a point that has not been explored), hence allowing the robots each to explore areas with the least amount of cost. This cost function is based on *value iteration* – a dynamic programming algorithm (see (Bellman, 1957)).

Other applications for collective robotics is in machine learning techniques. One issue in reinforcement learning (RL) algorithms is that of finding the optimal policy. Yahya et al. (2016) use the experiences of multiple robots in order to have them collectively learn an optimal policy for a robotic manipulator task (opening a door). The RL algorithm is implemented on a global neural network.

Mérmoud (2012) introduces the idea of Smart Minimal Particles (SMPs). Smart in that the agents are given an internal state (such as energy level), which is affected due to interactions with other agents and the environment. Minimal in that they have only basic amount of sensing, actuation, computation and communication, as well as not having the capability to hold internal representations of their environment. And particles due to the agents taking physical forms that consume a portion of physical space. Examples of SMPs are wide ranging, and they do include multi-robot systems (Kernbach, 2012).

Landgraf et al. (2010) built a honeybee robot in order to better understand the dancing communication system that occurs between honeybees (direct communication). Hence, by having full control over certain mechanisms in the robot (such as wings, body etc.) the authors could analyse exactly how certain signals impact the information sent and received by the honeybees.

2.3 Collective Behaviours – Conclusion

Of all the papers reviewed above, what is clear is that collective behaviour is on the whole an emergent property. This suggests that the rules of the collective are governed by often simpler rules at the level of the individual. The simplistic behaviour traits laid out by Ballerini et al. (2008) – that agents move in the same direction as their neighbours, keep close to their neighbours, and avoid collisions – are realistic traits that could be implemented in the MiRo and could form the basic rule set necessary to accomplish the objective of MiRo-MiRo *following* behaviour. The literature and experimental results on visual imprinting in baby chicks adds further justification as to why the development of a *following* strategy is not only interesting, but is relatable to phenomenon seen in the animal kingdom.

The literature on collective robotics is intrinsically linked with work on MAS and self-organisation, and the link between collective robotics and collective animal behaviours is clear (the term *Animats* is used specifically to describe the transference of animal behaviours and characteristics into robotic/artificial imitations – see Section 5.1.2 for a further discussion on how this relates to behaviour-based robotics, and also refer to the conference series From Animals to Animats (1990–2016) for more). Not only has research on biological behaviours helped in development of robotics, the integration of robotics into biological settings has proven useful for research into collective behaviours found in nature.

Fine and Shell (2013) point out that the studies completed in literature on collective flocking behaviours are "reported without explicitly detailing the sensing capabilities, limiting assumptions, and/or computation capabilities of the individual flock members." Robotics should allow a deeper investigation of these issues, considering robots allow easier manipulation and control over sensing and computation capabilities. The selfish herd problem as well as group density effects are phenomena that could potentially be explored further in a robotics environment using the MiRo.

Chapter 3

Vision – Biological and Artificial

This chapter aims to provide an overview of the aspects and characteristics of vision from within biology, and then to introduce computer/artificial vision techniques that can be found in the computer vision literature. These will include a review of object detection, motion tracking, and depth estimation strategies, both from within biology and in computer vision.

3.1 Biological Vision – A Review

Recall the process of visual imprinting in baby chicks mentioned briefly in Chapter 2; visual imprinting is intrinsically linked with invariant object recognition, with Wood and Wood (2015) arguing that newborn chicks are ideal model systems for studying the emergence of invariant object recognition, precisely because of their imprinting behaviour. This imprinting is first processed in the visual Wulst (VW), which is analogous to the mammalian visual cortex, before eventually being stored in the Intermediate Medial Hyperstriatum Ventrale (IMHV), or informally the domestic chick's forebrain (Horn, McCabe, and Cipolla-Neto, 1983; Nakamori et al., 2013). Zebrafish have been shown to imprint their kin via visual cues as well as olfactory cues (Hinz et al., 2013), thus there are a number of strategies for imprinting that take place in nature. Interestingly, unlike the chicks, zebrafish respond only to the imprinting cues of their own kin, and don't respond to nonkin.

However, much of the research performed on biological object recognition has been conducted on humans. (Biederman, 1987) proposes that human object recognition is done through the detection of component parts – that is simple volumetric entities such as cubes, cylinders, arcs, generalised cones – and their relations to one another. This is better known as recognition-by-components theory (RBC). Such an approach is similar to other proposals on object detection via parts/modules (Brooks, 1981; Tversky and Hemenway, 1984)

Further experimental work conducted by Biederman and Ju (1988) produced results showing humans can readily identify objects of full colour and detail (photos) as rapidly and error-free as they could recognise the same objects but as simple line drawings without texture or colour. This shows that edge-based features of objects, or more generally *local features* (Uchida, 2016), are more dominant in object detection than surface features. This is not to say surface features aren't important in object recognition, as Tanaka, Weiskopf, and Williams (2001) notes that surface features can be extremely useful when edge information is limited, such as when objects are occluded. In fact, experimental work conducted by Wood (2014) showed that baby chicks bind colour with shapes during the imprinting process, thus indicating colour does at least play a role in enabling the chicks to recognize the imprinted object.

Wolfe (2010) describes how visual search tasks are performed in concentrated spaces, or, that when humans are searching for an item, their visual salience is guided towards objects similar to the one they're searching for, at the expense of other possibly salient objects. A well known example of this phenomenon is the *invisible gorilla* experiment (Simons and Chabris, 1999).

But *following* requires more than just object recognition, it also requires object motion tracking. Smeets and Brenner (1994) propose a "motion detector" ability by humans that is triggered by relative motion of objects. This claim is further backed by Orban (1992), who demonstrated that motion is processed in the middle temporal area (MT) of the visual cortex (or V5).[1]

Rushton and Warren (2005) go further in producing experimental results indicating that humans can discern the speed of motion of an object from the relative motion of the retinal eye movement and movement of the object. Experimental work conducted by Wit et al. (2011) demonstrated that humans do not actually have a specialised role

[1]For the interested reader, Orban also mentions how the other areas of the visual cortex interpret scenes, starting with V1 as being a clearing-house, sending messages to other areas for further processing, such as V4 for colour, form in V3/V4 and stereo in V3 (Orban, 1992).

for tracking humans, but are able to track most efficiently grouped, coherently moving objects in an object-based attention mechanism (Scholl, 2001). This agrees with the RBC theory presented by Biederman (1987), in which humans find it easier to track the coherent movement of segmented objects.

Finally, Watson and Yellott (2012) studied the effects of pupil size on the function of vision, applying a mathematical model to describe the relationship between pupil diameter and luminance. Such adaptation of pupil size effects the ability of the eye to compute field depth, retinal illuminance and contrast sensitivity.

3.2 Computer Vision – A Review

This review is intended to introduce the reader to some of the computer vision techniques available from within the literature. Some of these techniques will be applied in MiRo, of which a fuller account can be found in Chapter 4.

3.2.1 Object Detection

Thresholding

The simplest detection method is perhaps binary thresholding, offering the ability to segment an object from its background, and can often simplify the task of further image analysis (Weszka, 1977).

Take for instance an 8-bit image of arbitrary resolution. Setting upper and lower pixel intensity thresholds, one could apply a threshold operation to each pixel, P_i, in the image and map the resulting thresholded image to a new binary image space, A:

```
for all pixels in image do:
  if $P_i$ > threshLower and $P_i$ < threshUpper:
    $A_i$ = 1
  else:
    $A_i$ = 0
  $R_i$ = 255*$A_i$
```

Notice that the binary image space, A, is then mapped into a third image space, R, making the resulting thresholded image easier to view.

Thresholding can work well under conditions of controlled lighting. Assuming this is the case, the problem of determining a suitable threshold is still unresolved. For practical purposes, this often requires experimentation, and adjustments to the thresholds are done "by eye" (Davies, 2012, p. 85).

A more interrogative approach is to sum the number of pixels at a certain intensity level (i.e. to build a histogram), and to use some minima as the threshold values, known as the mode method (Weszka, 1977). Although typically this is done on grey levels of pixels, one could easily expand it to cases of RGB pixels, thresholding each colour channel in turn.

The histogram approach was used successfully by Otsu (1979) on grey-level images to choose a suitable threshold level. By treating the histogram as a probability density function, it can then be used to predict the probability that a certain pixel of some intensity belongs to either the object in question, or to the background. The advantage of Otsu's method is that there does not need to be any *a priori* knowledge in its application.

Sauvola and Pietikäinen (2000) were able to use adaptive binarization (discussed in further detail in Section 4.2.4) to set a threshold that was a function of the mean and standard deviation of image intensities. Such adaptation is useful for overcoming illuminance changes across images (recall the eye's ability to adapt pupil size to achieve similar effects (Watson and Yellott, 2012)).

Feature detectors/descriptors

Another popular technique for object recognition is through local features. Uchida (2016) produced an in depth survey of local feature detectors, of which there are numerous techniques available, and include but are not limited to: Harris, Hessian, SIFT, Harris-Laplace, Hessian-Laplace, SURF, Harris-Affine, Hessian-Affine (Davies, 2012). They differ in their invariance, for example, Harris has rotational invariance, SIFT has scalable invariance, and Harris-Affine has affine invariance, where invariance is the robustness of the detector to those particular transformations (Uchida, 2016).

Feature extraction is the first stage in object recognition using local features. It is an image processing technique that extracts "interesting" areas of an image. What is deemed "interesting" can be arbitrary and dependent on the application, but generally this includes edges, corners and high-density regions (blobs). Homogeneous or overly repeated patterns in images are usually considered uninteresting. Feature extraction is already a widely researched area, and an in depth discussion of the various edge, corner and interest point detection techniques are given in (Davies, 2012, p.111-184).

Upon extracting interesting features from an image, detecting an object then requires a *feature descriptor* operation, which builds multi-dimensional feature vectors from those detected feature points/regions. One can than match these feature vectors to a query feature vector in order to determine the similarity between an object in an image and a query object (Uchida, 2016).

SIFT (Scale Invariant Feature Transform), introduced originally in (Lowe, 2004), is a commonly used feature detector/descriptor algorithm, and as such is often a baseline detector that gives good comparisons for other detectors. Due to its significance, the form of the algorithm will be introduced here. Lowe presents the algorithm in four major steps:

1. *Scale-space extrema detection*: The difference-of-Gaussian function (DoG) [2] is used to discover interest points that are invariant to scale and/or orientation.

2. *Key-point localization*: At each interest point, location and scale is found by fitting it to a detailed model, and key-points selected according to their stability measures.

3. *Orientation assignment*: Local gradient directions are used to appoint orientations to each of the key-point locations. Then, invariance in scale, orientation and location is realised for each feature, and further processing can occur with these features.

4. *Key-point descriptor*: Local gradients at some selected scale are found around regions at each key-point, and transformed to accommodate local shape distortions and variations in illumination.

Further feature matching is subsequently conducted by first extracting features from a set of reference images. Then, by comparing the reference feature vectors to new images using a fast nearest-neighbor algorithm, object detection can be accomplished.

Unfortunately, SIFT has shown to be comparatively slow versus newer algorithms. One such example is the SURF (Speeded-up Robust Features) operator (Bay, Tuytelaars, and Van Gool, 2006), shown to increase detection speed in various ways. Preceding SURF was work conducted by Viola and Jones (2001). Their approach was to

[2] See (Lindeberg, 1994) for justification for using the DoG operator rather than the Laplacian.

use an integral representation of the images, a machine learning algorithm based on AdaBoost, and a "cascade" scheme which enables the background of an image to be disregarded allowing more computation time on important regions. The integral image representation is a key feature of the SURF operator.

SURF and Viola/Jones's (henceforth referred to as Cascading, or the Cascade operator) algorithms both proved to be fast enough to be used with the MiRo. SURF, using a Fast-Hessian detector, had a computation time of 120ms (Pentium IV, 3GHz). Cascading could succesfully detect faces at a frame rate of 15fps, or a computation time of $<$ 67ms (Intel Pentium III, 700 MHz). MiRo's P3 processor for reference clocks at 1GHz, with standard camera frame rates of 4-8fps.

3.2.2 Motion Tracking

The task of then tracking an object falls primarily into the theory of optical flow, which at a basic level involves the computation of the change of pixels with equal intensities (i.e. local invariant features) across the image and across time (Davies, 2012). A popular addition is to incorporate a Kalman filter; not only can the Kalman filter predict future velocities and positions of the local features, but it can smooth noisy velocities that may be caused by robot disturbances (Wang et al., 2014).

Optical flow can be described mathematically as follows: if some point at (x,y,t) changes position by (dx,dy) within time dt, then the intensity at $(x+dx,y+dy,t+dt)$ should equal the intensity at (x,y,t), assuming there is no intensity changes for the same points in the image. I.e.

$$I(x+dx,y+dy,t+dt) = I(x,y,t) \tag{3.1}$$

Taking the Taylor expansion of the intensity function, and ignoring the second and higher order terms, gives,

$$I(x+dx,y+dy,t+dt) = I(x,y,t) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt \tag{3.2}$$

where $I = I(x,y,t)$. Therefore, from (3.1) and (3.2) the following holds,

$$\frac{\partial I}{\partial t} = -\left(\frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt}\right) \tag{3.3}$$

which can be written succinctly as,

$$\frac{\partial I}{\partial t} = -\nabla I \cdot \mathbf{v} \tag{3.4}$$

where,

$$\mathbf{v} = \left(\frac{dx}{dt}, \frac{dy}{dt}\right) \tag{3.5}$$

But this produces two problems: edges parallel to the direction of motion will be unrecognized, as (3.4) will equate to zero; and regions with constant intensity will result in $\nabla I = 0$ so that (3.4) will again equate to zero.

Additionally, Equation (3.4) is actually ill-posed, as there are two unknowns in $\mathbf{v}$ with only one formula. One solution was offered by Horn and Schunck (1981), by using the gradient constraint given from (3.3) and then minimizing through an iterative approach a global smoothness term. However, this approach is computationally expensive and hence most likely not suited for the MiRo.

A more efficient approach was proposed by Wei et al. (2011), in which a self-adaptive window is used, saving on computation cost, and Otsu's method for choosing a grey-level threshold (see above).

3.2.3 Vision Depth Determination

Building an accurate 3D representation of a scene with a single optical system is a nontrivial problem to solve, due to the perspective projection that depth imposes (in which parallel lines no longer appear parallel and midpoints don't appear to be at their midpoints) (Davies, 2012).

Nature has predominantly solved this task through the use of dual optical systems, whilst in computer vision scenarios, stereo vision techniques have been developed that can solve the particular problem of depth analysis (Scharstein, 1999). To introduce the reader to the simplicity of depth analysis with stereo vision, an example taken from (Davies, 2012) will be considered. Two lenses with parallel axes sets are given as shown in Figure 3.1.

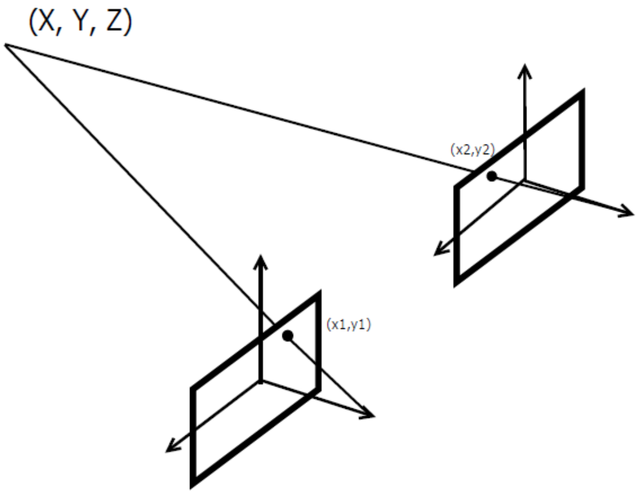


Figure 3.1: Stereo vision with two lenses, both having parallel axes systems.

Here the same point in the scene, given with co-ordinates (X, Y, Z), is viewed in two lenses at positions (x1, y1) and (x2, y2). It is simple to show that the depth, *Z*, can be computed as,

$$Z = \frac{bf}{x1 - x2} \tag{3.6}$$

where *b* is the baseline distance between the lenses and *f* is the focal length of the lenses. Clearly, if the parameters of both lenses are known, then Equation 3.1 becomes trivially easy to solve.

What becomes nontrivial however is determining whether two points in the images really do equate to the same point in the scene. Hence the need for robust local feature detectors, and usually utilising more than one point in the scene. To tackle this issue, there are two techniques available to solve this *correspondence problem* – one is that of "light striping" (Jokinen and Haggrén, 1998), and another method is using "epipolar lines" (Ishikawa and Geiger, 1998).

There can, however, be the case in which the lens axes are not parallel to one another, and a closer inspection into how camera geometries affect the image projections can be found in (Xu and Zhang, 1996). Not only does this work describe detailed camera modelling techniques, it also contains useful methods for projection computation using the single pinhole camera model. Then, using simple properties such as the space an object consumes in an image, depth and distance of an object can be predicted. This falls into the category of *perspective projection modelling*, a methodology used widely for 3D image reconstruction (Fang and Lee, 2012), but also used in motion estimation tasks (Eichenseer, Batz, and Kaup, 2016).

Without computing depth, information about potential collision of a moving camera can be estimated using the focus of expansion (FoE) of points in an image. This method exploits the principle that as an object is approached, its appearance in the image increases in size, but that the increase is about a centre point, termed the FoE. Some early work conducted by Negahdaripour and Horn (1989) used the FoE to recover motion of the camera relative to a still scene, with later work by Gil-Jiménez et al. (2016) showing an improved algorithm that could perform this operation in real-time.

3.2.4 Deep Learning in Computer Vision

This section serves primarily to highlight the growing importance of deep learning techniques for computer vision, and seems appropriate since deep learning is inspired in part by biological systems. At a basic description, deep learning is a class of machine learning that analyses large datasets and extracts the intricate structures that exist in the data (LeCun, Bengio, and Hinton, 2015). In contrast to the techniques discussed above that require careful design of algorithms for feature extraction, deep learning can use each layer to detect features "of its own accord". An example of this process could be in images: the first layer could detect edges and their orientations, the second layer could detect the particular arrangements of edges, whilst the third layer could detect particular objects as a result of the arrangements, and so on (LeCun, Bengio, and Hinton, 2015).

Deep learning is used with great success in many detection/recognition vision tasks through convolutional neural networks (CNN) (Sermanet et al., 2013; Girshick et al., 2013; Simonyan and Zisserman, 2014), a technique inspired by the biological visual cortex. Some CNNs can even outperform human recognition rates (Ciresan et al., 2012).

Chapter 4

Building a "MiRo Detector"

Two vision algorithms were then developed through using various techniques from the literature discussed in Chapter 3 for use in MiRo in order to develop a "MiRo detector".

Before introducing each of the two algorithms, this Chapter begins with a short section on thresholding within the simulator environment. The reason for developing a vision algorithm exclusively for the simulator is that it becomes highly useful to use when testing controller strategies, and hence is the reason for inclusion. The description for this is given in Section 4.1.

The first algorithm, *Algorithm 1*, uses predominantly pixel intensity thresholding techniques and a histogram approach for determining the ideal threshold limits. It then introduces a novel method of MiRo detection that builds on thresholding and the histograms, by using the histograms of the greyscale images as inputs to a perceptron neural network. The development for Algorithm 1 is given in Section 4.2.

The second algorithm, *Algorithm 2*, is an application of SURF which, as discussed in Chapter 3, is an extension of SIFT and Viola/Jones's integral image technique. But the SURF algorithm is expanded further here, by applying it to a Bayesian inference framework in order to estimate a posterior probability of TMiRo existing in an image. The development for Algorithm 2 is given in Section 4.3.

Each algorithm is introduced in turn, with a detailed description of the technical aspects of each, followed by an example of their implementation on images of MiRo. The two algorithms are then analysed and compared to determine the computation times and detection accuracies for each. The Python source code is given in Appendices A.1 and A.3 for Algorithm 1 and 2 respectively.

4.1 Thresholding for use in the Simulator

In order to detect TMiRo in a simulated environment, the simpler detection method of thresholding was first implemented. This section serves as an introduction to using thresholding for object detection, but does not form part of the final vision algorithms – as previously stated, its purpose is for use within the Gazebo simulator only.

The detector was therefore developed using simulated images of MiRo taken from the Gazebo simulator. To ensure adequate segmentation of MiRo from the background, MiRo's body is given the colour blue in order to distinguish it from its surroundings. Six thresholds then need to be set: an upper and lower limit for each of the three colour channels. Initially, this is done through an educated guess, and altered slightly until MiRo is successfully segmented from the background. The thresholds are found at [R, G, B]:

```
threshLower = [0, 0, 100]
threshUpper = [10, 10, 255]
```

Running the threshold algorithm from Section 3.2.1 three times for each colour channel gives the result shown in Figure 4.1b. The average number of pixels successfully thresholded over the left and right camera streams was 3,052. Plotting the histograms for the blue colour channel of Figure 4.1a is given in Figure 4.2. Notice that there is a large spike at a blue intensity of approximately 180, suggesting most of the blue pixel values exist here. However, there are some mini spikes at 100 and 255 (ignoring the spikes near 0). Therefore, sensible thresholds for the blue colour channel could be between 90 and 255. Doing the same for the green and red colour channels can be used to threshold 'out' rather than 'in' the background. Sensible upper thresholds were found to be 80 for red and 50 for green.

Hence, the new thresholds using the mode method is given as:

```
threshLower = [0, 0, 90]
threshUpper = [80, 50, 255]
```

Using these thresholds for the thresholding algorithm gives the results shown in Figure 4.1c. It is immediately obvious in Figure 4.1c that a larger proportion of MiRo's body has been thresholded when compared against Figure 4.1b. Similarly, the average number of pixels successfully thresholded is now 3,573 – a 17% increase upon the original threshold operation.

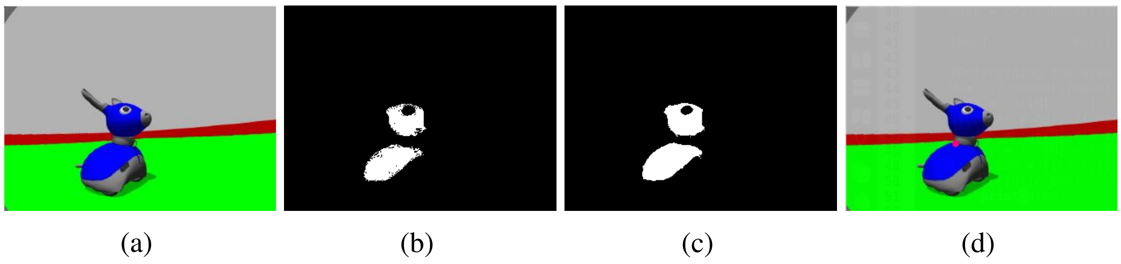


Figure 4.1: (a) A single image taken from MiRo's left camera stream in the Gazebo simulator. (b) Running the threshold algorithm with the original threshold limits set for each colour channel and outputting into a greyscale image space. (c) Running the threshold algorithm with the updated threshold limits found using the mode method for each colour channel and outputting into a greyscale image space. Using the updated threshold limits using the mode method gives a 17% increase in the number of pixels successfully thresholded. (d) Showing the position of the centroid.

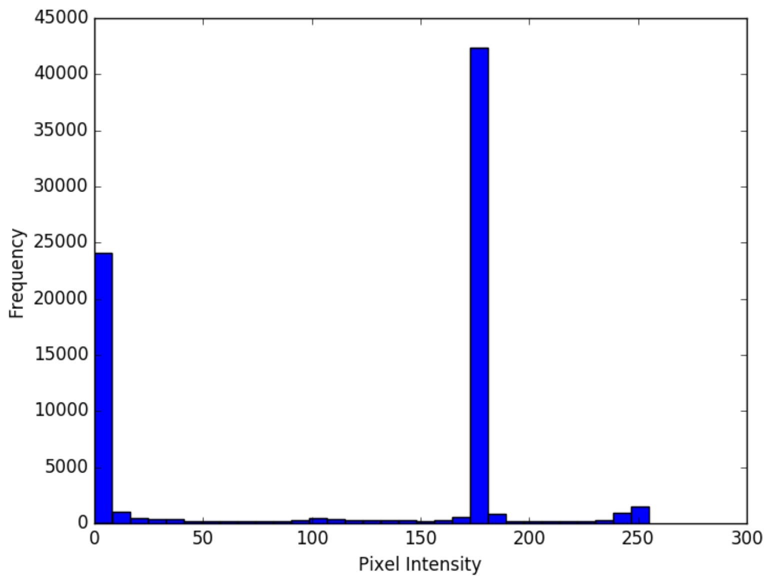


Figure 4.2: Histogram of the blue channel intensity values for Figure 4.1a. Notice the large spike at approx. 180, indicating most of MiRo's blue body has a blue intensity value of ≈180. The same can be done for the red and green colour channels to determine thresholds to filter out the unwanted background.

Finally, following successful completion of the thresholding operation, the centroid position, (C_x, C_y), is found via the first order moments:

$$C_x = \frac{M_{10}}{M_{00}}, \quad C_y = \frac{M_{01}}{M_{00}} \tag{4.1}$$

where, $M_{ij} = \sum_x \sum_y x^i y^j I(x,y)$

Given each thresholded pixel is set to a greyscale intensity of 255, the total number of successfully thresholded pixels can be found by taking $M_{00}/255$, where M_{00} is simply the total sum of the intensity values for the thresholded area.

4.2 Algorithm 1 – Adaptive Thresholding, Histograms and the Perceptron

4.2.1 Real-world Setting

Note that the above threshold approach is useful for testing the controller (Chapter 5) in a highly controlled environment, as one does not need to worry about false-negative detections. In a real-world setting however this cannot be guaranteed, and so the following algorithm developed here is intended to be used in real-world settings. The algorithm employs adaptive thresholding, histograms and a perceptron ANN for more robust detection.

Detection in a real-world setting is certainly nontrivial. There can of course be many objects that are of the same or similar colour as MiRo, and thus with a singular threshold could give many false-positive detections. Further, lighting effects, shadows and MiRo's LED's can give false-negative results through missed detection.

A single view of TMiRo under a more controlled scenario as shown in Figure 4.3 proves relatively straightforward to threshold. Notice that the image has been converted into greyscale before applying the threshold. This was done using the following rule for each pixel, $[I]$, with 8-bit RGB colour channels (OpenCV, 2015a):

$$Grey[I] = 0.299R[I] + 0.587G[I] + 0.114B[I] \tag{4.2}$$

The pixel intensity threshold limits were then chosen as 180 for the lower limit and 255 for the upper limit. Whilst in Figure 4.3 the thresholding is largely successful under the semi-controlled environment, when applying the same thresholding limits to the images in Figure 4.4 there appears too many false-positive detection zones as seen in Figure 4.5.

To overcome this issue, a more robust MiRo detector is needed – the following sections aim to build on the above development in order to increase its robustness.

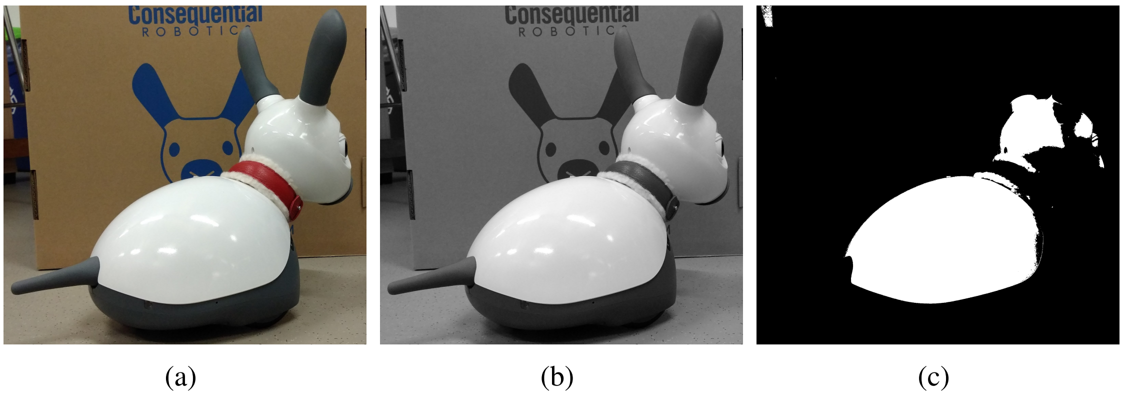


Figure 4.3: (a) Original image of TMiRo. (b) A greyscale image of (a). (c) Applying thresholding limits of 180 and 255 successfully detects TMiRo, although not perfectly. Notice specifically the shadow effect on MiRo's face that prevents detection.

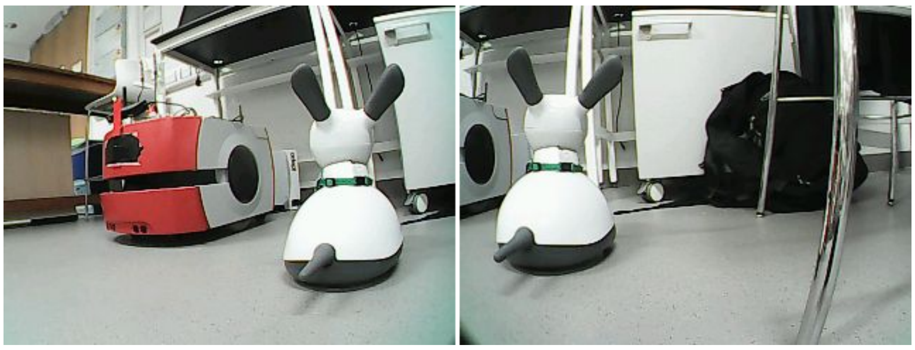


Figure 4.4: Stereo images of a TMiRo as viewed from MiRo

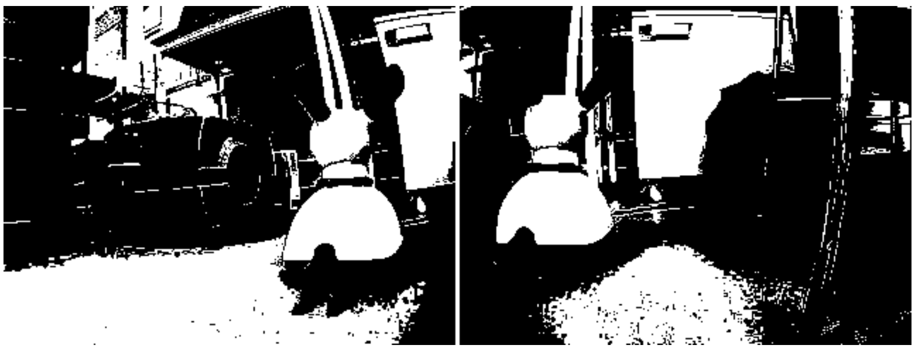


Figure 4.5: Thresholding of the stereo images of Figure 4.4 using the threshold limits of 180 and 255 as in Figure 4.3. Notice there are far too many false-positive detection zones, which is not desirable for extracting TMiRo from the scene.

4.2.2 Correlations in Histograms

The inspiration for this approach came from the work conducted by Otsu (1979), in which the histograms of greyscale images were used as probability density functions which could be used to predict the likelihood of a pixel belonging to a particular object. It was thought that perhaps through acquiring the greyscale histograms of many images of MiRo, there should be some consistencies or correlations in the histograms – for instance, MiRo's body is mostly white, but also has a grey coloured underbelly and ears – and that these consistencies could be exploited in some way.

To test this hypothesis, 48 images of MiRo were taken using MiRo's own cameras for best accuracy, and under varying scales, orientations, environments and lighting conditions (MiRo was cropped in each image to extract it from the scene). These would later form the positive training images for use in the perceptron (Section 4.2.3). Each image was then converted to greyscale using Equation 4.2, and the histograms for each image were computed (256 bins used for each of the 8-bit pixel intensities ranging from 0-255) and then normalised. See Figure 4.6 for the histograms of all 48 images. Notice the cluster of spikes at intensity ranges 120-170, and 180-255. It is speculated that these should be the image background in the first instance, and MiRo's white body/head in the second.

Taking the correlation matrix for the dataset, given as,

$$\mathbf{c} = \mathbf{X_p}^\top \mathbf{X_p} \tag{4.3}$$

where $\mathbf{X_p}$ is the positive image dataset with dimensions 48x256 (the normalised histograms for each training image). It then becomes clearer that there does indeed exist some correlations in the data, as shown in Figure 4.7. The correlation matrix highlights other areas of correlation that are not clear from the histograms in Figure 4.6, such as those at the intensities close to 63 – a sensible assumption being that this could be MiRo's grey ears/underbelly.

Subsequently, it was deemed justified that these correlations could be exploited in some way. The next section discusses how this was achieved using the perceptron artificial neural network.

4.2.3 Perceptron

Rosenblatt's perceptron (Rosenblatt, 1962) is one of the earliest artificial neural networks, and certainly one of the simplest. Given the objective for the MiRo detector is to simply detect only MiRo, one can term the problem as having to design a robust binary detector – either an object is a MiRo or it isn't. The perceptron is therefore chosen as it is well suited as a binary classifier. The perceptron algorithm for training

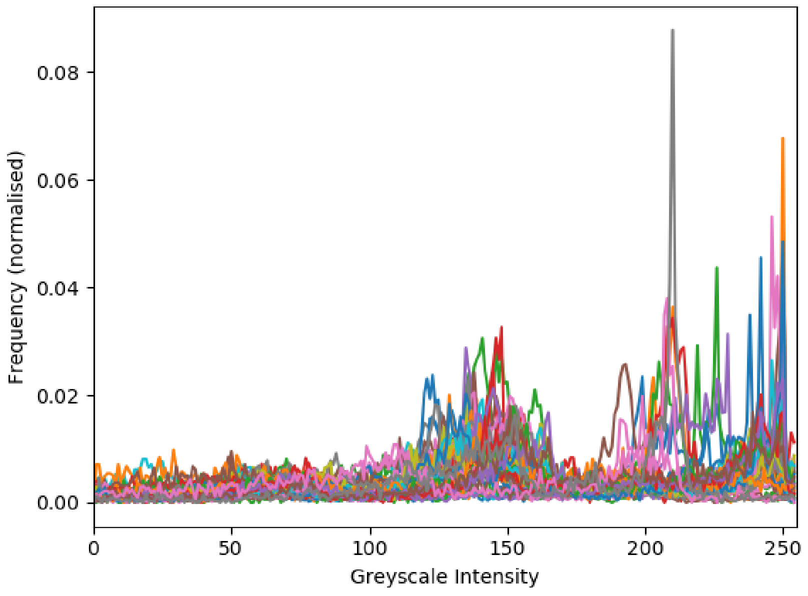


Figure 4.6: Normalised histogram of the pixel intensity values for all 48 MiRo training images. Notice the two spike clusters, one approximately between intensities 120 to 170, the other between 180 and 255. It is speculated that these regions are the background and MiRo's body/head, respectively.

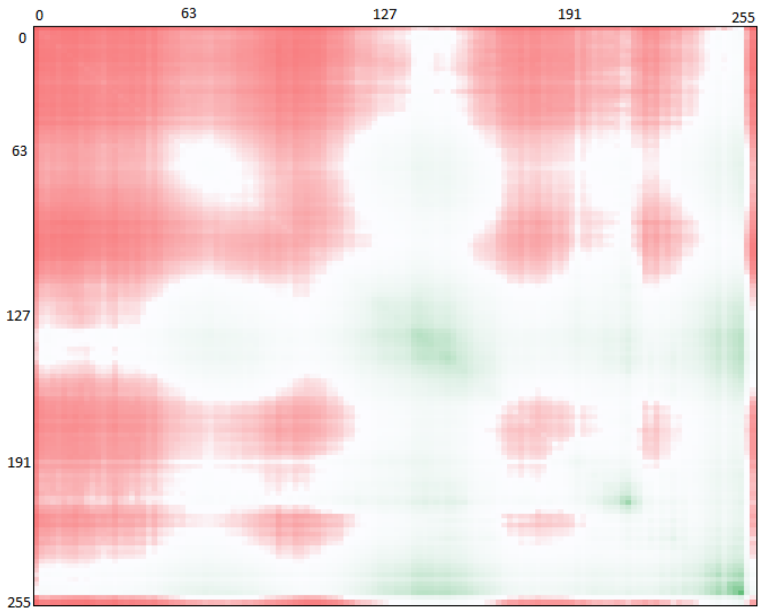


Figure 4.7: Correlation matrix for the positive training images in the form of a heat map. Red indicates low correlation, whilst green indicates highest correlations. Notice the large correlations around the midpoint and the lower right corner. These intensities are close to 136 and 250 respectively, which agrees with the histogram plots above. Further areas of correlation are highlighted here, such as at intensities close to 63, which could justifiably be argued to be MiRo's grey ears/underbelly.

Perceptron algorithm

1. Load the training histogram data, $(\mathbf{x}_1, d_1)$, ... , $(\mathbf{x}_n, d_n)$, where $\mathbf{x}_i$ is a 256x1 vector of the greyscale histogram data, and d_i = 1,0 for positive and negative images respectively.
2. Initialise the weight vector, $\mathbf{w}$, as random floating point decimals ranging from 0 to 1: $w_i = rand(1)$
3. Normalise the weights: $w_i \rightarrow \frac{w_i}{\sqrt{\mathbf{w}^{\mathrm{T}}\mathbf{w}}}$
4. Train the network an arbitrary number of times:
   (a) Choose at random an image histogram, $\mathbf{x}_i$
   (b) Compute the output as: $y = \mathbf{x}_i^T \mathbf{w}$
   (c) Update the weights as: $\mathbf{w} \rightarrow \mathbf{w} + \eta(d_i - y)\mathbf{x}_i$
5. The classifier is computed as: $h(\mathbf{x}) = \begin{cases} 1, & \text{if } y \geq \beta \\ 0, & \text{otherwise} \end{cases}$

Figure 4.8: The perceptron algorithm for training a binary classifier. Notice that the inputs to the neural network are the histogram data for the training images.

the binary classifier is shown in Figure 4.8. The full python script for the algorithm implementation is given in Appendix A.1.

Training the Perceptron ANN

In total there were 48 positive training images and 786 negative training images – 50 of the negatives were captured from the lab environment in which MiRo mostly operates, the rest were random images taken from the author's personal photo library. In order to properly train and then validate, 4 images were removed from the positive training set and 4 from the negatives, which were later used for validation purposes.

All appropriate formulas required for training the perceptron are given in Figure 4.8. Note that usually a bias is included in the perceptron algorithm, and then the decision boundary is set to 0. In this instance, there is no bias, but rather the decision boundary is set to a non-zero value β.

Optimisation

The learning rate was first optimised to give the quickest rate of convergence. η was varied from 0.01 to 100 in powers of 10, and the rate of convergence for each plotted vs number of iterations – see Figure 4.9. A learning rate of $\eta = 10$ was found to be the most acceptable value, as it gave quickest convergence without exploding the weights.

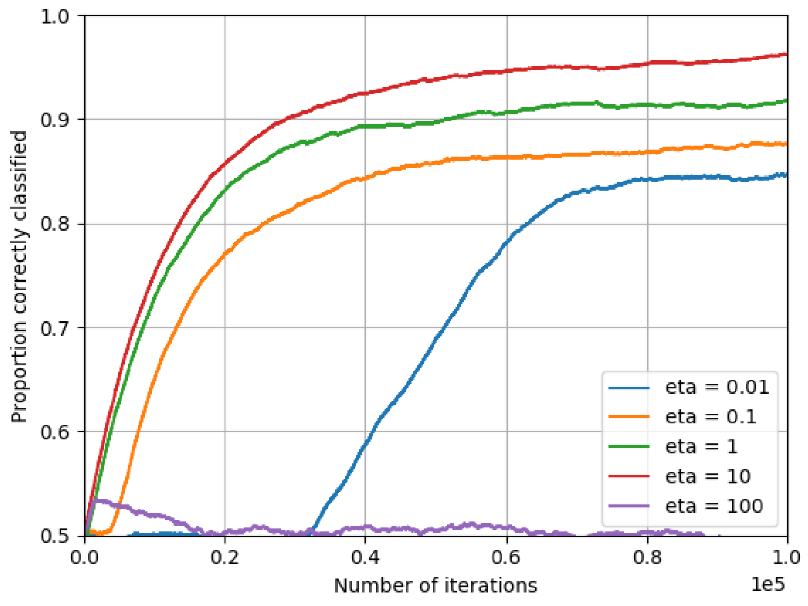


Figure 4.9: The results of varying η and its effects on convergence of the perceptron. $\beta = 0.5$. It was deemed that $\eta = 10$ gave the most efficient learning. One order of magnitude above this results in weights that explode as seen with $\eta = 100$.

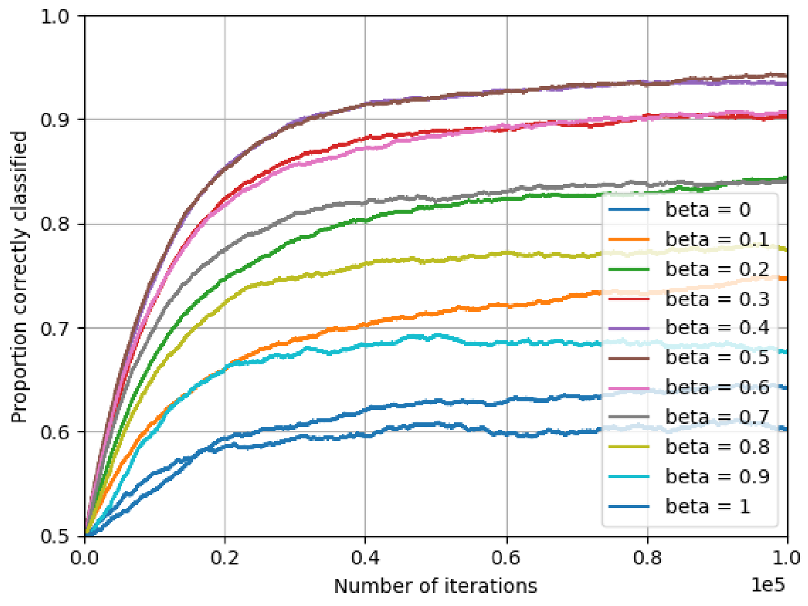


Figure 4.10: The results of varying β and its effects on classifying. $\eta = 10$.
The optimal value for β was found to be 0.5 as it gave the highest rate of correctly classified images.

It was then necessary to select a suitable decision boundary, β, that would give the most accurate classification. β was varied from 0 to 1 in increments of 0.1. The results are as given in Figure 4.10. Setting $\beta = 0.5$ gave the largest proportion of correctly classified images. Thus, the optimum values for the algorithm can be summarised as being $\eta = 10$ and $\beta = 0.5$. Note that the 'proportion correctly classified', plotted against the number of training iterations, is computed using the exponential smoothing function given as,

$$e_t = \alpha H_t + (1-\alpha)\, e_{t-1} \tag{4.4}$$

with $e_{-1} = 0.5$, $\alpha = 10^{-4}$ for large smoothing, and $H_t = 1,0$ if the classifier correctly or incorrectly classified, respectively.

Validation and Summary

In order to validate the efficacy of the algorithm, it was necessary to remove some images from the training set and use them for validation subsequent to the training phase. 4 images were removed at random from the positive images and 4 also from the negative. Running the algorithm 100 times (i.e. removing at random the validation sets on each run, and resetting the weights for each run), the algorithm managed to achieve an average rate of 90.1% correct classification with a standard deviation of +/- 9.2%; 5.4% were incorrectly classified as false-negatives, whilst 4.5% were incorrectly classified as false-positives.

Though not perfect, having rates that are better than chance adds valuable information that can be used alongside other vision operations. Furthermore, these results are for images in which MiRo was placed in uncontrolled and variable environments, lighting conditions, scales and orientations. As such, the results are more than satisfactory.

It is important to note here that the images of MiRo were cropped for the purposes of training. Thus, in order for the network to work effectively with MiRo, it is necessary to build another algorithm that can efficiently locate regions of interest (ROI) that can then be used with the network developed above. The development of a ROI algorithm is given in the next section.

4.2.4 Region of Interest (ROI)

To save on computation and to aid in extracting meaningful histograms, it is desired that an image be reduced to one or a few regions of interest that has some chance of being TMiRo, before further processing is applied to the image.

Now it shall be assumed that in any given scene, if TMiRo exists within the scene, it should be one of the lightest regions in the image. Rather than using a global threshold value, it could be more beneficial to use an adaptive threshold, based on the mean and standard deviation of the pixel intensities in the scene. A technique utilising this approach is known as *adaptive binarization*, and was successfully applied by Sauvola and Pietikäinen (2000) to overcome image noise, illumination and resolution changes. They used the following formula for determining a threshold value,

$$t(x,y) = m(x,y) \cdot \left[1 + k\left(\frac{\sigma(x,y)}{R} - 1\right)\right] \tag{4.5}$$

where $m(x,y)$ is the mean for a local pixel centred in a given window, $\sigma(x,y)$ is the standard deviation for the same pixel in the window, R = 128 for an 8-bit greyscale image, and k ranges from 0.3 to 0.5.

The problem with the approach above, as shown by Najafi and Salehi (2016), is that computing the mean and standard deviation for every pixel in an image is very computationally expensive. Najafi's approach is to use a stochastic implementation of Sauvola's algorithm in order to speed up computation. Another problem with the above is that it places the constraint of $t(x,y) \leq m(x,y)$ for $k \geq 0$. Rather, for this application it is desired that the threshold should be above the mean: $t(x,y) \geq m(x,y)$ given the assumption about MiRo's light body.

The approach proposed here then is to set a threshold for an entire image to save on computation, and to modify Equation 4.5 to give a threshold value above the mean. As such, a threshold is set for each pixel using the following,

$$t(x,y) = m(img) \cdot \left[1 - k\left(\frac{\sigma(img)}{R} - 1\right)\right] \tag{4.6}$$

which is almost identical to (4.5) except that the threshold for each pixel is given by the mean and standard deviation of the whole image, rather than for windows centred on the pixel, and, more crucially, the $+$ve sign is swapped for a $-$ve sign to satisfy the condition $t(x,y) \geq m(x,y)$ for $k \geq 0$. Note that if $t(x,y) > 254$, then the threshold is set to $t(x,y) = 254$.

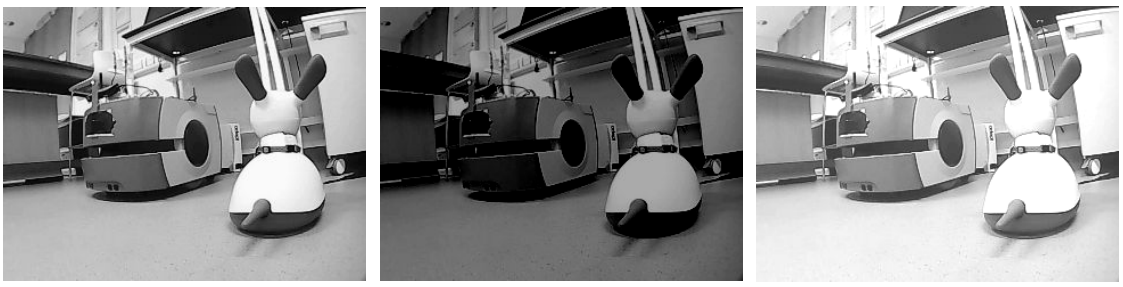
Figures 4.11a and 4.12a displays the result of applying the threshold given by Equation 4.6 with $k = 2$ on the left image in Figure 4.4 (converted to greyscale using Equation 4.2). But further, to test its capability under extreme lighting conditions, Figure

4.11b has all pixel intensities reduced by 80, with its thresholded image shown in Figure 4.12b, whilst Figure 4.11c has all pixels increased by 50 with its thresholded image shown in 4.12c, both to simulate reduced lighting levels and increased lighting levels, respectively. Notice the thresholding is consistent across all three light levels, with only small differences. When using the threshold values used in Figure 4.5, the reduced lighting levels of Figure 4.11b returns a blank image. Therefore, the adaptive thresholding technique applied above proves to be capable of handling varying lighting conditions. Finally, all three thresholded images perform more adequately than shown in Figure 4.5, with less false-positive regions.

The final step is simply to mark those regions of high density as the regions of interest using OpenCV's contours function (see OpenCV (2015b) documentation). Performing the above on Figure 4.4 gives satisfactory results as shown in Figure 4.13.

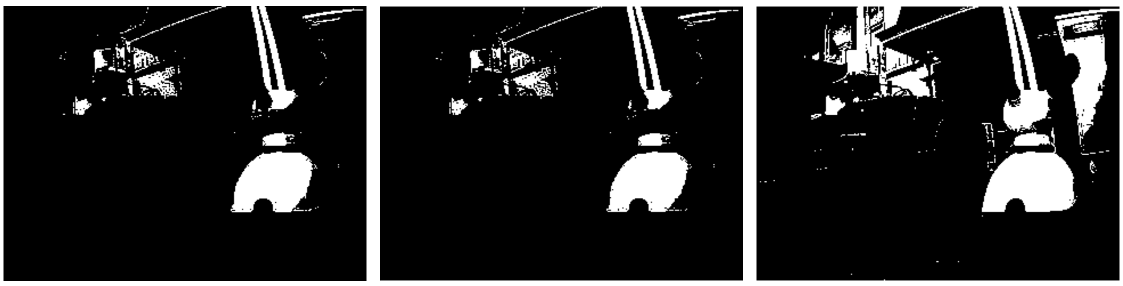
Finally, note that the ROI in Figure 4.13 doesn't take into account the whole of MiRo's body. Therefore, a $(1.05w \times 1.8w)$ window size[1] is taken, with the new window's centroid is moved to the top of the original ROI. Then the window size is increased by 10% in 4 steps. The classifier can be computed for each window increase, and stopped when either MiRo is detected or 4 window size increase steps have passed.

[1]Here, w is the width of the ROI. So a ROI with dimensions 80×75 would be increased to 84×144.



(a) (b) (c)

Figure 4.11: (a) Original image of TMiRo converted to greyscale. (b) Simulated reduced lighting of (a) by reducing all pixel intensities by 80. (c) Simulated increased lighting of (a) by increasing all pixel intensities by 50.



(a) (b) (c)

Figure 4.12: (a), (b) and (c) are all the thresholded images of their equivalent greyscale images above, using the modification of Sauvoli's formula, (4.6), for determining the thresholds. Notice there are less false-positive regions in all images compared to Figure 4.5, and that there is some consistency across each of the images, displaying a capability to cope with varying lighting.

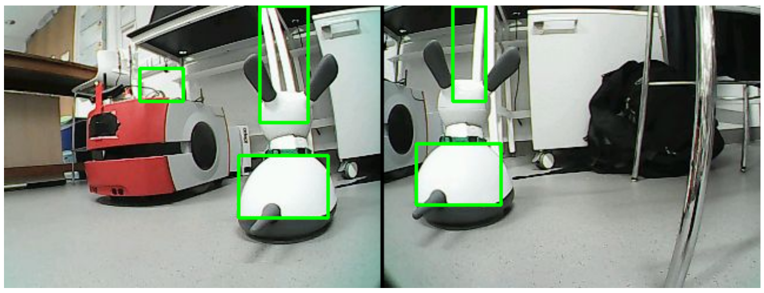


Figure 4.13: The results of running the region of interest algorithm on the stereo images in Figure 4.4. Following this will be the task of applying the histograms for each ROI

4.3 Algorithm 2 – SURF with Bayesian Inference

Speeded-Up Robust Features (SURF), developed by Bay, Tuytelaars, and Van Gool (2006), is a detector-descriptor algorithm. Its detector uses the Hessian matrix on an integral image (see below) which saves significantly on computation time, and therefore is termed the 'Fast-Hessian' detector by the authors. The descriptor describes the distribution of Haar-wavelet responses for a given interest point region. Finally, using the sign of the Laplacian, the authors presented a novel indexing step in order to increase matching speed of features and robustness of the descriptor (Bay, Tuytelaars, and Van Gool, 2006). However, feature matching shall be exchanged for a Bayesian estimator in this application for estimating posterior probabilities on TMiRo existing in an image.

4.3.1 SURF Detector

Integral Images

The integral image approach was first introduced by Viola and Jones (2001) in their boosted cascading scheme. The integral image at some location, (x,y), is computed as a sum of the pixels to the left and above point (x,y), and is given by,

$$ii(x,y) = \sum_{x'=0}^{x'\leq x} \sum_{y'=0}^{y'\leq y} I(x',y') \tag{4.7}$$

Once the integral image is computed, it is simple to compute the integrals at any image point by performing only 4 additions, thus being computationally fast (see Davies (2012, p.174-176)).

Hessian Matrix

Let a pixel within a window of an image have the location $X = (x,y)$ and the image space be at some scale σ. Then the Hessian matrix for this location is given as,

$$H(X,\sigma) = \begin{bmatrix} L_{xx}(X,\sigma) & L_{xy}(X,\sigma) \\ L_{yx}(X,\sigma) & L_{yy}(X,\sigma) \end{bmatrix} \tag{4.8}$$

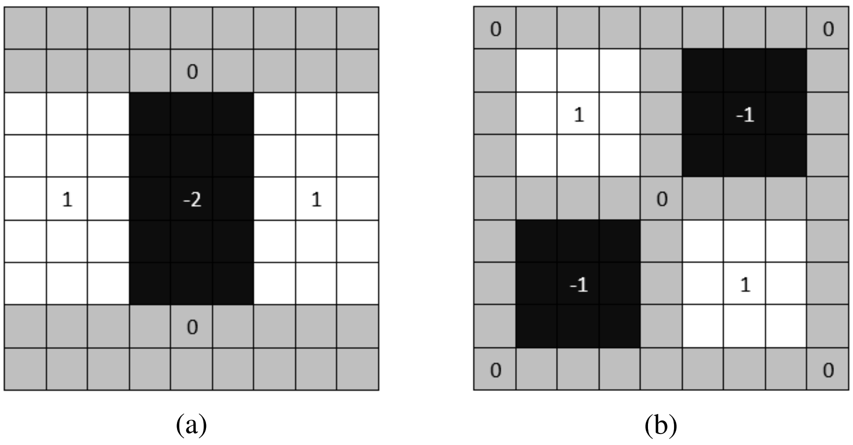


Figure 4.14: The approximation of the Gaussian second order derivative in the *x*-direction, D_{xx} (a), and *xy*-direction, D_{xy} (b), for a 9x9 box filter. To generalise, the same can be scaled for *n*x*n* = *N* box filter sizes.

where $L_{xy}(X,\sigma)$ is the convolution of the second order derivative of the Gaussian, $G(\sigma)$, with the pixel intensity, $I(X)$, given as (similarly too for $L_{xx}(X,\sigma)$, etc.),

$$L_{xy}(X,\sigma) = \frac{\partial^2}{\partial x \partial y} G(\sigma) * I(X) \tag{4.9}$$

with $G(\sigma)$ being the Gaussian function,

$$G(\sigma) = \frac{1}{2\pi\sigma^2} \texttt{exp} \left(-\frac{x^2+y^2}{2\sigma^2} \right) \tag{4.10}$$

The Gaussian second order derivatives can be approximated further by letting $L_{xx}(X,\sigma) \approx D_{xx}(X,\sigma)$ and noting that,

$$D_{xx}(X,\sigma) = \sum_{i=1}^{N} g_i * a_i \tag{4.11}$$

where N is the number of pixels in a window, g is the approximated Gaussian weight and a is the summation of pixel intensities in the region. Figure 4.14 shows the approximated second order Gaussian partial derivatives for *xx* and *xy*. g is given by the weights, ranging from -2 to 1, and a is the value of the summed intensities for each of the shaded regions. Note how the integral image method described above is used to full effect in the approximation of the Hessian, where summed regions of the image, i.e. a, can be found efficiently using the integral image method.

The approximated Hessian determinant can then be given as,

$$det\left(H_{approx}\right) = D_{xx}D_{yy} - \left(wD_{xy}\right)^2 \tag{4.12}$$

in which w is approximated using the following,

$$w \approx \frac{||L_{xy}(\sigma)||_F \, ||D_{xx}(n)||_F}{||L_{xx}(\sigma)||_F \, ||D_{xy}(n)||_F} \tag{4.13}$$

where n is the size of the box filter used in the approximation of the second order Gaussian, and $||\cdot||_F$ is the Frobenius norm operator. Letting $\sigma = 1.2$ and $n = 9$, then (4.13) results in $w \approx 0.9$, giving for (4.12)[2],

$$det\,(H_{approx}) = D_{xx}D_{yy} - (0.9D_{xy})^2 \tag{4.14}$$

The weight term is necessary to balance the given weights in Figure 4.14 with the determinant of the actual Hessian matrix in Equation 4.8.

The final point to mention is that of scaling. In Bay's paper, the scale, denoted by σ, is set to $\sigma = 1.2$ for a 9x9 box size. As the box size increases to 15x15, 21x21, 27x27, ..., nxn etc., then the scale factor increases according to $\sigma = \frac{n}{9} \times 1.2$. I.e., for a box size 27x27, $\sigma = 3 \times 1.2 = 3.6 = s$. (4.13) then remains constant throughout. $\sigma = s$ is set since s is used later in determining window sizes and Gaussian parameters in the SURF descriptor.

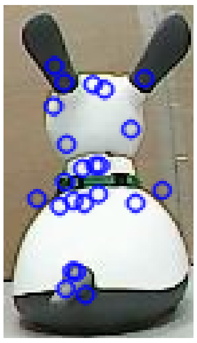


Figure 4.15: Interest points from the SURF detector

Figure 4.15 displays the results of running the Fast-Hessian detector with a threshold of 400 for the Hessian determinant. There are 23 interest points detected in total, particularly in regions such as the tail, collar and ears.

4.3.2 SURF Descriptor

Haar-wavelet responses are used in the descriptor algorithm of SURF both for assigning orientation to achieve rotation invariance, and for building the resulting descriptor. However, many applications don't require rotational invariance in situations where the camera remains horizontal (Bay, Tuytelaars, and Van Gool, 2006). For simplicity then, it is assumed MiRo's head will not rotate about the yaw and pitch axis, and thus the orientation assignment section of the SURF descriptor is excluded here. In effect, the implementation of SURF in this context is actually the implementation of Upright SURF

[2]For completeness, an example for calculating (4.13) is given in Appendix A.2.

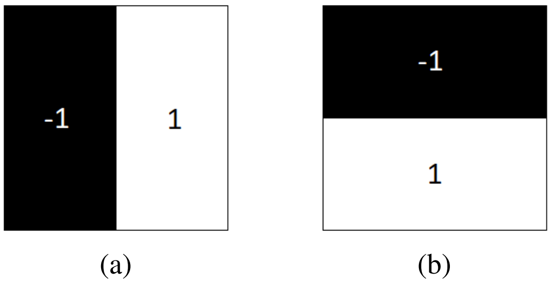


Figure 4.16: Haar-wavelet responses in the x-direction (a) and y-direction (b). Again, using the integral approach speeds up the summation of the areas in the box, and in taking the differences of the summations.

(U-SURF).

Wavelet transforms show to be in some cases more useful in capturing important information from an image than the Fourier transform, as wavelet transforms offer temporal information as well as frequency responses. A detailed discussion on the theory and methods behind wavelet transformations with particular attention to digital image processing can be found in (Gonzalez and Woods, 2010). The Haar-wavelet types used in SURF are simple differences between the sum of pixel intensities in a boxed region – see Figure 4.16. Its simplicity, as with the Gaussian approximations in Figure 4.14, make the algorithm computationally inexpensive.

The task is first to build a boxed window of size 20s around the centre of an interest point taken from the detection stage described above. The window is then split into 16 equally sized subregions. For each subregion, the Haar-wavelet responses are computed at 25 equally spaced regions. The Haar-wavelet response in the x direction is given as d_x, and in y direction d_y. Summing each Haar-wavelet response for each of the 25 regions in the original subregion then gives the initial feature descriptor for the subregion,

$$v_i = \left[\sum d_x \quad \sum d_y \quad \sum |d_x| \quad \sum |d_y|\right]_i^T \tag{4.15}$$

Notice in (4.15) that the feature vector also includes the absolute sums of the responses too.[3]

Performing the above for all 16 subregions gives the overall feature vector[4], having

[3]Consider a region that has a striped vertical pattern. $\sum d_x$ would equal 0 in such a region. But $\sum |d_x|$ would produce a positive value. Hence, the absolute values offer valuable information that the standard summations miss. See (Bay, Tuytelaars, and Van Gool, 2006).

[4]Note that feature vector and feature descriptor terms are used interchangeably in this section.

size 64×1, for that interest point,

$$\mathbf{v} = \begin{bmatrix} v_1^T & \cdots & v_i^T & \cdots & v_{16}^T \end{bmatrix}^T \tag{4.16}$$

which can then be computed for each interest point in an image.

This then completes the description of the SURF algorithm. However, having the feature descriptors is not final in object recognition; a further necessity is to be able to use those feature descriptors in additional processes, such as feature matching (as performed by Bay, Tuytelaars, and Van Gool (2006)). But here, we opt instead for the application of Bayes' theorem in predicting the probability of finding a set of features belonging to MiRo.

4.3.3 An Application of Bayes' Theorem in "MiRo-Detection"

Given an interest point in an image, the probability of that interest point belonging to a certain class of object, H_i, given the feature descriptor associated with the interest point, $\mathbf{v}$, can be given as,

$$p(H_i|\mathbf{v}) = \frac{p(\mathbf{v}|H_i)p(H_i)}{\sum_j p(\mathbf{v}|H_j)p(H_j)} \tag{4.17}$$

H_i in this application takes on two values only (i.e. a binary classifier), where H_0 represents a non-MiRo class, and H_{Miro} represents a MiRo class. Further, it is sensible only to consider the probability of a feature descriptor belonging to a MiRo, then (4.17) can be re-written as,

$$p(H_{Miro}|\mathbf{v}) = \frac{p(\mathbf{v}|H_{Miro})p(H_{Miro})}{p(\mathbf{v}|H_0)p(H_0) + p(\mathbf{v}|H_{Miro})p(H_{Miro})} \tag{4.18}$$

Noticing that $p(H_0) = 1 - p(H_{Miro})$, and substituting into (4.18),

$$p(H_{Miro}|\mathbf{v}) = \frac{p(\mathbf{v}|H_{Miro})p(H_{Miro})}{p(\mathbf{v}|H_0)(1 - p(H_{Miro})) + p(\mathbf{v}|H_{Miro})p(H_{Miro})} \tag{4.19}$$

gives the final form of the posterior probability for a single feature vector, dependent only on the likelihoods and the prior probabilities. The challenge is then in estimating these values.

Estimating the likelihoods – $p(\mathbf{v}|H_{Miro})$ and $p(\mathbf{v}|H_0)$

Commencing first with the estimation of the likelihood $p(\mathbf{v}|H_{Miro})$. For this, we turn once again to the 48 positive training images used in the perceptron and proceed by extracting the feature descriptors from all 48 training images. A common approach taken in feature matching is to use the Mahalanobis distance for measuring the similarity between two feature vectors (Baumberg, 2000). Setting the covariance matrix in the definition for the Mahalanobis distance as an identity matrix results simply in it computing the Euclidean distance. $p(\mathbf{v}|H_{Miro})$ could then estimated by comparing the Euclidean distance between the feature in question, $\mathbf{v}$, and each of the stored feature vectors extracted from the stored positive training images. The number of training images containing a feature vector with high similarity (where similarity is measured by setting a Euclidean distance threshold) to the query feature vector as a ratio to the total number of images is then the estimated value of $p(\mathbf{v}|H_{Miro})$. To be more succinct, the following set of operations are performed:

1. The Euclidean distance between query feature vector, $\mathbf{v}$, and each stored feature vector, $\mathbf{v}_k$, in a training image K is computed as

$$d(\mathbf{v},\mathbf{v}_k) = \sqrt{(\mathbf{v}-\mathbf{v}_k)^T(\mathbf{v}-\mathbf{v}_k)} \tag{4.20}$$

2. The smallest $d(\mathbf{v},\mathbf{v}_k)$ for image K is then stored as $d(\mathbf{v},\mathbf{v}_k)_{min}$

3. If $d(\mathbf{v},\mathbf{v}_k)_{min} < thresh$, then: $count \rightarrow count + 1$

4. Steps 1-3 are repeated for all stored training images

5. $p(\mathbf{v}|H_{Miro})$ is then estimated as,

$$p(\mathbf{v}|H_{Miro}) \approx \frac{count}{no.training\,images} \tag{4.21}$$

Note that the above 5-step procedure can be used also for estimating $p(\mathbf{v}|H_0)$, simply by substituting the stored feature vectors of the positive training images in step 1 for the feature vectors of the negative training images, with one important difference: due to the negative images having exceedingly higher resolution than the positive images, they also contain a much larger number of feature vectors (exceeding 1000 feature vectors vs less than 20 for a typical positive image of MiRo). As such, the possibilities of running into one feature vector that is similar to a query feature vector is much higher. To avoid this, only the first 50 feature vectors for a negative image is therefore used.[5]

Further, for speedier implementation, steps 2 and 3 can be performed simultaneously by computing the Euclidean distance before comparing against the threshold value for each feature in turn, and stopping if a feature passes the threshold.

Setting the Euclidean distance threshold – *thresh*

It is necessary then to select the optimal value for *thresh* in order to classify most accurately. For this, taking the same 48 positive training images and using 45 for training and 3 for validation, the SURF feature descriptors for the 45 training images are computed and stored. The 5-step procedure outlined above is then performed for each validation image. The goal is to maximise the proportion of feature descriptors having $p(\mathbf{v}|H_{Miro}) \geq 0.5$ – a feature descriptor that satisfies this condition is defined as being correctly classified.[6]

But further, all 786 negative training images will be taken and the above 5-step procedure is performed for all these images. The goal here is to minimise the number of feature descriptors with $p(\mathbf{v}|H_{Miro}) \geq 0.5$ (or maximise the number of features having $p(\mathbf{v}|H_{Miro}) < 0.5$).

For the positive images, the validation images are selected randomly and the process is repeated 10 times for each value of *thresh*, where *thresh* is incremented from 0 to 1 in steps of 0.1 (so that the process is repeated 330 times in total), whereas the process is only performed once for each negative image. Figure 4.17 displays the proportions of correctly classified features for the positive and negative training images. Having a threshold of close to 0.46 seems to give the greatest proportion of classified images both for the positive and negative sets, both giving a rate of approximately 0.6 correctly classified features.

[5]Consider also that upon actual implementation, the ROI filter would most likely be run first anyway, reducing the image resolution.

[6]One should take note though that "correctly classified" is a slight misnomer, as there is no actual classification taking place – it is set only so that an optimal value for the threshold can be found.

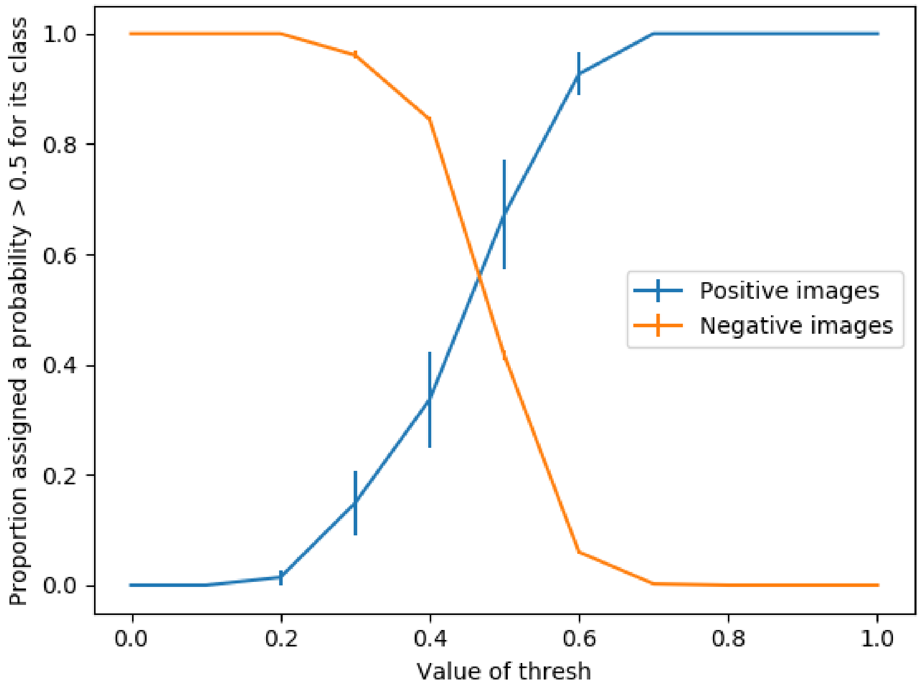


Figure 4.17: Plot for number of correctly classified feature vectors vs threshold value when using Bayes' with SURF. Plotted also are the error bars indicating one standard deviation. The negative images have small error bars due to the large number of images used vs positive images. At $thresh \approx 0.46$, the proportion of images given a likelihood of 0.5+ is close to 0.6 for both positive and negative images (the crossover point), which indicates setting $thresh = 0.46$ to be a justifiable value.

Incorporating a series of feature vectors, V

Note the above is computing the probability given one feature vector. Rather, it is more desirable to obtain the probability of MiRo given all the feature vectors obtained in the image, which shall be defined as $p(H_{Miro}|\mathbf{V})$. Then, if there are N independent feature vectors in an image the following relation can be used,

$$p(\mathbf{V}|H_{Miro}) = p(\mathbf{v}_1|H_{Miro})p(\mathbf{v}_2|H_{Miro})...p(\mathbf{v}_N|H_{Miro}) = \prod_{i=1}^{N} p(\mathbf{v}_i|H_{Miro}) \tag{4.22}$$

which is replaced into Equation 4.19 to give the final form of the probability of an image (or ROI) containing a MiRo given the feature descriptors in the image,

$$p(H_{Miro}|\mathbf{V}) = \frac{\prod_i p(\mathbf{v}_i|H_{Miro})p(H_{Miro})}{\prod_i p(\mathbf{v}_i|H_0)\,(1-p(H_{Miro}))+\prod_i p(\mathbf{v}_i|H_{Miro})p(H_{Miro})} \tag{4.23}$$

The final requirement is then to estimate the prior probability, $p(H_{Miro})$.

Estimating the prior – $p(H_{Miro})$

The prior probability offers interesting possibilities for shaping the behaviour of MiRo: for instance, if we assume that MiRo is always guaranteed to be in a ROI and set $p(H_{Miro}) = 1$, then MiRo will follow any salient object that passes the ROI filter, as (4.23) equates to 1. Setting $p(H_{Miro}) = 0$ then has the opposite effect, and MiRo will never follow any salient object it encounters. Thus, as the prior probability is increased, MiRo becomes more willing to follow objects that pass its ROI filter.

For now though, the following assumption is going to be made for setting an initial value to the prior: assume that there will always be at least one TMiRo in MiRo's proximity, and that if MiRo were to turn round 360^o, a TMiRo would certainly be somewhere within its field of view. Each of MiRo's cameras covers a viewing angle of 90^o, and hence the prior is set to $p(H_{Miro}) = 90/360 = 0.25$.

Finalising the Procedure

Finally, performing the Bayesian procedure with the above parameters on the MiRo in Figure 4.15 gives a posterior probability of 0.999998, almost certainty that MiRo is contained in the image (performed by removing that image from the training set so as not to influence the result). Compare this with the negative images in which only 7/700, or 1%, had probabilities over 0.1, and only 4/700, or 0.6%, gave probabilities above 0.9, and the significance of a probability of 0.999998 therefore becomes clear. This completes the development on the SURF with Bayesian estimation algorithm. A full analysis of the algorithm with a comparison against the first algorithm is given in the next section. The full Python source code for this algorithm is given in Appendix A.3.

4.4 Analysis

The SURF algorithm with Bayes' estimation was compared against the histogram with perceptron approach. The criteria for comparison were detection time and detection accuracy. Detection time is split into two separate timings: the first is the time taken to process an image in order to ready it for the classifying stage. In the first algorithm this includes extracting the histogram, whilst for SURF this includes computing the feature descriptors. The second time is completed for the timing of the classifier. Again, in the

Table 4.1: Comparison of the histogram/perceptron algorithm vs SURF with Bayes' inference. All times should be taken relative to one another.

Algorithm	Total time (ms) (processing + classifying)	Correctly classified (%) (false-negatives / false-positives)
Histogram/perceptron	0.271 (0.146 + 0.125)	90.1 (5.4 / 4.5)
SURF with Bayes	84.729 (0.729 + 84.0)	87.8 (11.4 / 0.8)

first algorithm, this includes the computation time for the perceptron neural network (a simple linear sum, no learning is conducted here), whereas for SURF this means running the Bayesian estimator.

In each case, the same image was used for validating, by removing it from the training set. Therefore, it is assumed that the ROI filter is run for each instance already and is not included as part of the analysis. The results are given in Table 4.1.

For the reader's interest, all positive training images are displayed in Appendix A.4. One may notice the diversity of TMiRo orientations, scales, backgrounds and environments that were necessary to properly assess the robustness of each algorithm.

4.5 Discussion

Starting with the results in the analysis above, it is clear which of the two algorithms offers the most time efficient solution, with the histogram/perceptron performing 313 times faster than SURF. However, there is a noticeable difference in the proportion of time spent for each in the processing and classifying stages. In the histogram approach, the time spent between the processing stage and classifying was almost equal, having a processing:classifying ratio of 1:0.86. For SURF with Bayes however, this ratio was 1:9,767; or, alternatively, 99% of the total time was spent classifying. This 'bottleneck' in the classifying stage can be explained simply due to the need for the algorithm to compute a Euclidean distance between each feature in the query image, say 20+ features for a typical MiRo image, and all other feature vectors in the training images. With 48 positive and 786 negative training images, each with say another 30 features each on average, this equates to a total of 500,000+, 64-vector length dot product computations each also needing to be square-rooted. Compare this with the perceptron approach, which requires only a single 256-vector length dot product computation, and the difference is obvious. And this is in spite of applying the Naive Bayes' classifier above (see

Davies (2012, pp.678-679)) – a method employed to reduce the amount of computation necessary. Hence, for the Bayes' approach, not only is it necessary to store all training feature vectors thus consuming a lot of memory, but also the computations are laboriously long. The advantage of the perceptron is that learning can be done off-line, so the only data that needs storing on MiRo are the network weights, and the computations are an efficient, single dot product.

Perhaps then it would be beneficial to employ the simpler strategy adopted by the authors of SURF (Bay, Tuytelaars, and Van Gool, 2006), using simple feature matching. Whether this would indeed be a faster strategy is as yet unknown, but what can be said is that the accuracy would likely suffer with feature matching, and feature matching alone is not equivalent to object recognition. Bay achieved an average of 82.6% recognition rates with standard SURF feature matching, whereas the Bayes' estimator could achieve 87.8% 'MiRo estimation' – of more use than simply matching features, and with slightly higher recognition rates.

In terms of accuracies of MiRo recognition for both the algorithms, notice again that the histogram approach slightly outperforms SURF. There are also noticeable differences in the proportion of false-negatives and false-positives for each. In the histogram/perceptron algorithm, of the incorrectly classified images, 55% were false-negatives whilst 45% were false-positives. Of the false-negatives, the images were largely of the same type: they contained images of MiRo positioned far away from the camera, in an overly cluttered scene, or were positioned side ways (see Figure 4.18).

Meanwhile, of the misclassified[7] images with SURF almost all, 93%, were false-negatives. Of these, there appeared a clear pattern, for 70% of the false-negatives were images in which TMiRo was positioned sideways (this is despite there being only 23% of the total number of training images having sideways views). The other 30% were of TMiRo being positioned from behind. Interestingly, for all front facing images of TMiRo the Bayes' estimator worked with 100% accuracy, which could be explained due to the efficient ability of the SURF algorithm to detect and describe facial features – Viola and Jones (2001) work was initially developed for facial detection applications, from which Bay, Tuytelaars, and Van Gool (2006) took the integral image approach for use in SURF.

[7]Classified is declared as any significant positive probability, in this case $p(H_{Miro}|\mathbf{V}) \geq 0.1$.

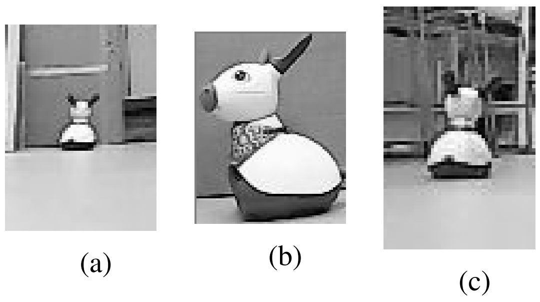


Figure 4.18: Typical examples of false-negatives in the histogram/perceptron algorithm. (a) when TMiRo was positioned far away. (b) when TMiRo was positioned sideways. (c) when TMiRo was positioned far away and was in a cluttered scene.

Sideways views of TMiRo therefore presented a challenge for both of the algorithms above, and perhaps further work, whether an additional classifier for sideways views or a separate algorithm altogether, is left open for additional consideration. It may be that interesting collective phenomena emerge from this lapse in the algorithms when implemented for a group of MiRos. This again is left open for further consideration.

To conclude this section on the development of MiRo's vision algorithms, recall from the project aims that one of the primary criteria for the controller was the ability to not just detect TMiRo, but to track TMiRo too. By determining at each camera frame rate the position of TMiRo in the scene, then tracking is to a minor extent completed. However, a more rigorous approach to tracking may be desired, in that TMiRo's velocity be computed as well as position.

This should be considered as further work, and the initial steps have already been documented in Section 3.2.2 with motion tracking. By applying a binary threshold, such as the one developed in the ROI approach (Section 4.2.4), Equation 3.4 can be used as an input to an advanced controller (compared to the controller in Chapter 5).

It is worth raising here the potential issues that could be encountered when determining motion of features of image points, and a critical one could be that of noise. For exact positioning, the impact of noise is more negligible, so long as the noise isn't overwhelmingly large. For motion tracking however, consider the effects of noise on a thresholded centroid. The centroid could rapidly move position depending on the amount of pixels thresholded in the ROI algorithm for instance, and certainly physical disturbances to the robot would cause large fluctuations in centroid velocities. It would therefore be wise to employ some smoothing algorithms – a common time-varying filter being the Kalman filter for instance (Wang et al., 2014).

Chapter 5

Control Law

5.1 Following Control Laws – Review

Developing a controller for a mobile robot can take on a number of forms, but fall usually into two main categories: one is the high precision kinematic modelling approach in a feedback/state-space system – a classical control approach. The other is a biologically inspired behaviour-based approach; an example being simplistic reactive controllers in which actuator excitations are some monotonic function of sensor intensities readings. This review aims to give an overview of the application of each in a mobile robotics setting, highlighting the various differences and noting the advantages and disadvantages for each approach.

5.1.1 Kinematic Modelling

Kinematic modelling in mobile robotics is usually performed even in behaviour-based control mechanisms, for instance when determining the necessary voltage input to a motor in order to achieve a desired angular velocity. Furthermore, having a kinematic model of the MiRo allows a 'closely-optimal' solution to be found for a path planning problem. As such, any behaviour-based controller developed later can then be compared against the kinematic solution in order to analyse its effectiveness. Developing a kinematic model for the MiRo therefore has a number of uses.[1]

[1]The dynamic model is arguably just as important. But in an attempt not to extend this section more than is necessary, one can refer to (Wahde, 2016, p.21-28) for an in-depth analysis of the differentially driven mobile robot's dynamics.

The MiRo is a differentially driven robot with one caster wheel for support as shown in Figure 5.1. The *forward kinematics* are solved for in order to give MiRo's new x, y and θ position in a global co-ordinate frame, and is found from the forward velocity, V, and angular velocity, ω, of MiRo's body given as a function of the angular velocities of MiRo's wheels,

$$V = \frac{(\omega_L + \omega_R) r}{2} \tag{5.1}$$

and,

$$\omega = \frac{(\omega_R - \omega_L) r}{D} \tag{5.2}$$

where ω_L and ω_R are the left and right wheel angular velocities (rad/s) respectively, r (m) is the wheel radius (same for each), and D (m) is the track distance between the two wheels. Velocities in x and y as a function of θ for the differentially driven robot in a global coordinate reference frame can then be given as,

$$V_x(t) = V \cos \theta(t) \tag{5.3}$$

$$V_y(t) = V \sin \theta(t) \tag{5.4}$$

Then using $\omega_L r = v_L$ and $\omega_R r = v_R$, and integrating (5.3) and (5.4) to give position, and (5.2) to give rotation,

$$[\Delta x]_{t_0}^{t_1} = \int_{t_0}^{t_1} V_x(t) dt = \int_{t_0}^{t_1} \frac{v_L(t) + v_R(t)}{2} \cos \theta(t) dt \tag{5.5}$$

$$[\Delta y]_{t_0}^{t_1} = \int_{t_0}^{t_1} V_y(t) dt = \int_{t_0}^{t_1} \frac{v_L(t) + v_R(t)}{2} \sin \theta(t) dt \tag{5.6}$$

$$[\Delta \theta]_{t_0}^{t_1} = \int_{t_0}^{t_1} \omega(t) dt = \int_{t_0}^{t_1} \frac{v_R(t) - v_L(t)}{D} dt \tag{5.7}$$

Notice that this system is coupled, as (5.5) and (5.6) are dependent on (5.7), and that it is nonlinear due to the trigonometric terms, thus having a classical state-space system without linearisation is not possible. These results are due to the constraints that the wheels impose on the movement of the robot – see above that the number of degrees of freedom (DOF) the robot has is 3 (in x, y and θ), yet the number of differential degrees of freedom (DDOF) is only 2, as it can only manipulate its forward/reverse velocity, and its rotational velocity. As such, if DOF $>$ DDOF, then the robot is said to have *nonholomonic kinematic constraints* (Siegwart and Nourbakhsh, 2004, p.75-77).

But the interest is in creating a controller that can position MiRo in any pose desired of it, and particularly that the controller be in a closed-loop feedback set-up for increased optimality (though not necessarily optimal). A number of solutions are offered within literature, see for instance (Campion, Bastin, and D'Andrea-Novel, 1996; Wit and Sordalen, 1993). But the one presented here is similar to Siegwart and Nourbakhsh (2004, p.82-88), with the only difference here being that the robot's local co-ordinate frame is not considered.

In Figure 5.1, the goal is to have MiRo move from its initial position at $[x_0, y_0, \theta_0]$, to a goal configuration $[x_G, y_G, \theta_G]$. The 'error' is equal to $\mathbf{e}(t) = [\Delta x, \Delta y, \Delta\theta]$, and thus the goal is achieved if $\mathbf{e} = 0$. Then the aim is to design a controller, $\mathbf{K}$, that drives the error towards zero: $\lim\limits_{t\to\infty} \mathbf{e}(t) = 0$.

MiRo's kinematic model is as defined in Figure 5.1, with ρ being the distance between MiRo's wheels centre and the goal position; α is the angle between MiRo's forward velocity vector and ρ; and β is the angle between the vector that is in the direction of ρ, and the goal position – these define the polar coordinates. Δx, Δy and θ are transformed into the polar coordinates by,

$$\rho = \sqrt{\Delta x^2 + \Delta y^2} \tag{5.8}$$

$$\alpha = -\theta + \arctan\frac{\Delta y}{\Delta x} \tag{5.9}$$

$$\beta = -\theta - \alpha \tag{5.10}$$

which can then be expressed in terms of their derivatives in matrix form, in which the linear and rotational velocities of the robot are considered the system's inputs (Ferreira et al., 2008),

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -\cos\alpha & 0 \\ \frac{\sin\alpha}{\rho} & -1 \\ -\frac{\sin\alpha}{\rho} & 0 \end{bmatrix} \begin{bmatrix} V \\ \omega \end{bmatrix} \tag{5.11}$$

If a linear control law of the form,

$$\begin{bmatrix} V \\ \omega \end{bmatrix} = \mathbf{K} \begin{bmatrix} \rho \\ \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} k_\rho & 0 & 0 \\ 0 & k_\alpha & k_\beta \end{bmatrix} \begin{bmatrix} \rho \\ \alpha \\ \beta \end{bmatrix} \tag{5.12}$$

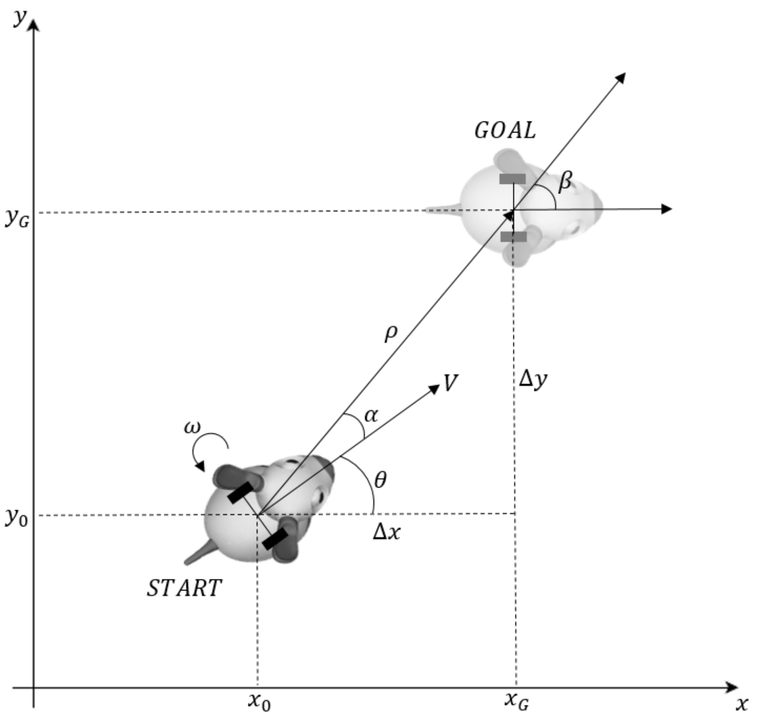


Figure 5.1: The kinematic model of the MiRo. The aim is to develop a controller that can take MiRo from the START position to the GOAL configuration, by driving the error terms Δx, Δy and $\Delta\theta$ (in this instance $\Delta\theta = \theta$) to zero. MiRo's inputs are V and ω, which are determined by Equation 5.12 for this control law.

is assumed, then substituting (5.12) into (5.11) gives the final form of the closed-loop controlled system,

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -k_\rho \rho \cos\alpha \\ k_\rho \sin\alpha - k_\alpha \alpha - k_\beta \beta \\ -k_\rho \sin\alpha \end{bmatrix} \tag{5.13}$$

with the criterion that,

$$k_\rho > 0; \quad k_\beta < 0; \quad k_\alpha - k_\rho > 0 \tag{5.14}$$

to retain stability, determined from the eigenvalues of the controlled dynamics matrix for the linearised system (see (Siegwart and Nourbakhsh, 2004, p.86-89) for full proof).

The above classical control method gives good performance and has proven effective for many robotic tasks. But this approach can be arduous, and makes assumptions that cannot generally be made, such as perfect knowledge of distance/orientation and non-slip conditions. Furthermore, it does not consider the dynamics of the robot which can be yet another arduous task to develop. As such, a behaviour-based approach will next be introduced, which does not require as complex mathematics as given above, simplifying greatly the task of controller development.

5.1.2 Behaviour-based Control

MiRo does not have any global positioning system (GPS), or built in internal maps, and as such MiRo's ability to follow a TMiRo can only be accomplished through its own local sensors. The restrictions go further for this project though, in that the controller should use only information from the cameras. Thus, unless a robust distance and orientation estimator can be implemented, the classical controller method above would not be a suitable option.[2] This point and similar other issues in classical control is expressed by Brooks (1999), perhaps one of the originators of the behaviour-based approach to robotics control. There are a number of advantages with a behaviour-based robot over the classically-controlled robot. Firstly, a behaviour-based system that is biologically inspired can be robust, repeatable and adaptive, in the same way biological systems are (Matarić, 1998). Secondly, algorithms are often simpler to develop whilst retaining complex looking behaviours, as is exemplified with reactive controllers or the Braitenberg vehicle (Braitenberg, 1984). Thirdly, behaviours are of a higher-level description than the lower level processes, enabling more general levels of control (Matarić, 2001). Fourthly, whilst being used to develop controllers for robotic systems, behaviour-based robotics also aims to help in the understanding of biological systems too (Matarić, 1998). And lastly, applying the behaviour-based approach remains true to the principle of MiRo being biomimetic, given that behaviour-based control has a closer resemblance to biological control strategies.

Behaviour-based robotics shares similar characteristics to finite-state machines, in that the robot exhibits high-level states (such as 'collision avoidance', 'follow mate', etc.), but is noticeably different in that behaviours, rather than states, have some interaction amongst each other so that multiple behaviours can be active simultaneously (Matarić, 1998). This creates interesting dynamics that are exploited to create intelligent and stable robotic systems (Brooks, 1991).

This review on behaviour-based control is divided into two sections: reactive controllers and adaptive behaviour. The principles of each are briefly introduced, and then notable applications of each from within the literature are mentioned.

[2]Despite human's ability to see with dual vision, distance is still not perfectly inferred as exemplified with the "moon illusion" (Weidner et al., 2014). Further, orientation of a "target human" can actually affect the perceived distance of the target human (Ejung et al., 2016).

Reactive Controllers

The Braitenberg vehicle originated in a series of thought experiments conducted by the psychologist Valentino Braitenberg (1984). In his work, Braitenberg introduced a number of vehicle types, each an extension of a previously developed Braitenberg vehicle. The principle is rather simple, and involves nothing more than direct connections between external sensing equipment and a vehicle's wheels. One sensor is connected to one wheel, of which the sensors can sense anything: light, temperature, chemicals, etc., and the connections can be excitatory (larger sensor reading gives larger motor speeds), inhibitory (larger sensor reading gives smaller motor speeds), linear, nonlinear, and can include logic gates. With such controllers, one can imagine a wide range of complex behaviours that the robot could exhibit.

The issue with Braitenberg's vehicles in reality is that, as Ronald Arkin mentions, they are “inflexible, custom machines and are not reprogrammable” Arkin (1998, p.11). This is mostly true, but they can be programmable in a limited sense, in that the input/output parameters coupling the sensors to the actuators could be updated, or "reprogrammed" by changing a physical component (say resistance values in an amplifier). But without a microprocessor, that is the limit to the adaptability of the Braitenberg vehicle.

Hogg, Martin, and Resnick (1991) soon developed 12 Braitenberg type vehicles, made from specially modified Lego bricks. One interesting development of theirs is an *Attractive* and *Repulsive* pair of vehicles. Using lights and light sensors, the *Repulsive* vehicle is attracted towards the *Attractive* vehicle. However, once the light sensor on *Attractive* is switched to ‘on’ due to *Repulsive* being too close, *Attractive* then dashes away until out of range of *Repulsive*. *Repulsive* later catches up with *Attractive*, and the process repeats. Hence simple interactions can be achieved using nothing more than lights, light sensors and thresholding devices.

A natural progression of the Braitenberg vehicle is towards reactive controllers in general and the extension of reactive controllers particularly for mobile robots. Examples include Zhu et al. (2005) who combine reactive navigation control with a fuzzy controller, or Tigli et al. (1993) who use reactive controllers in a MAS with a centralised global control element to perform high-level reasoning. Both aim to take advantage of the simplicity of the reactive controller, whilst also attempting to introduce more sophisticated planning or intelligence.

The problem of the non-adaptive nature of the Braitenberg vehicle was confronted by Santamaría and Ram (1996), who were able to apply a learning approach to the controllers via case-based reasoning and the application of reinforcement learning in a robot navigation task. Thus, reactive controllers have proven to be suitable for adaptive learning in order to cope with changing environments.

Adaptive Behaviour

Adaptive behaviour is the ability for an agent to adapt, or learn, over a period of time in order to better its behaviours to achieve a specific outcome (Dorigo and Colombetti, 1998). Almost all learning approaches tend to fall in any one of the categories of supervised, reinforcement, or unsupervised learning (Alpaydin, 2010).

Adaptive behaviour can further be applied for a number of robotic control approaches, as it involves nothing more than an iterative updating of parameters, weights or controller gains. For instance, in the classical control approach, the theory of *Adaptive Control* exists for the purpose of learning and adapting controller parameters in a classical control setting (Narendra and Annaswamy, 1989).

Adaptive behaviour is widely used in neural networks, as demonstrated in Section 4.2.3, in which a neural network could be trained to "learn" to correctly classify MiRo in an image. The credit often goes to Hebb (1949) for the introduction of a rule for synaptic plasticity, and is certainly the basis on which much of the work on synaptic plasticity is done today (Dayan and Abbot, 2001). And as discussed in the review of collective behaviours, adaptive learning can be used in social robotics, using multi-bot systems to learn a global optimal policy. A detailed discussion of the application of learning in a behaviour-based multi-bot system is given by Matarić (2001),[3] in which mobile robots demonstrated the ability to learn from past behaviour choices, and optimise their behaviour selections by avoiding the execution of those inefficient behaviour sequences. A reinforcement learning technique was implemented along with something known as *shaping*.

It was Dorigo and Colombetti (1998) who first introduced the term *shaping* in a

[3]Matarić also argues that the behaviour-based control is separated from reactive control. But often the two can be used in conjunction, as demonstrated by He, Ren, and Kan (2010) in a behaviour-based approach where each behaviour contained reactive controllers.

robotics setting, taking it from the field of psychology, in which behaviours can be gradually altered over time to converge on a desired behavioural outcome. Their approach was based upon reinforcement learning mechanisms, although they did not document explicitly the algorithms used. They also argued that the work conducted at that time by Arkin (1998), Brooks (1999) and others on behaviour-based robotics had not yet fully realised the potentials of combining robotics with the behavioural sciences – mainly due to the incapability in deciding what in a robot should be explicitly designed, and what should be learned by the robot.

Reinforcement learning is clearly a promising approach to take, allowing the robot to learn gradually overtime an optimal policy that will maximise its rewards without having any input from an outside observer. As such there is no shortage of reinforcement learning techniques applied in mobile robot applications (Bruske, Ahrns, and Sommer, 1997; Fagg, Lotspeich, and Bekey, 1994; Altuntas et al., 2016).[4]

5.2 MiRo Control Law

A controller was developed based upon the techniques of reactive control discussed above, and implemented with good effect within the Gazebo simulator. This section documents the development, optimisation and analysis of the controller.

5.2.1 Development

Basic Following

The controller was intended to be designed to have low complexity, but that achieves the basic aim of *following* behaviour. This was accomplished through the inspiration of the Braitenberg vehicle and reactive controllers, with TMiRo's position in each camera coupled directly with the wheel speeds using some nonlinear laws and logic gates.

Firstly, for ease of use within the Gazebo simulator, TMiRo is detected using the thresholding technique described in Section 4.1. Then by extracting the centroids from each camera stream, an estimated centroid position in x, C_x^{est}, for the TMiRo in the

[4]See Appendix A.6 for a proposal of a MiRo controller which applies reinforcement learning in an ANN.

Table 5.1: Logic table for the B parameter. A value of 1 for L_{det} or R_{det} indicates that TMiRo has been detected in the left or right camera respectively, whilst 0 indicates TMiRo has not been detected. If TMiRo is not detected in either camera, then TMiRo *following* behaviour is not exhibited.

L_{det}	R_{det}	B
1	1	1
1	0	1
0	1	-1
0	0	N/A

scene could be computed as a function of the centroids from each camera:

$$C_x^{est} = (C_x^l - B\frac{w}{2}) + (C_x^r - \frac{w}{2}) = C_x^l + C_x^r - \frac{w}{2}(B+1) \tag{5.15}$$

where C_x^{est} is the estimated x component of the centroid computed as functions of C_x^l and C_x^r, which are the respective x components of the centroids in the left and right camera streams.[5] w is the camera width (320 for MiRo's 320x240 camera resolution), so that the $(-\frac{w}{2})$ terms act to centre the centroid around the midpoint of the image. The B term is equal 1 if TMiRo is detected in either both cameras or only the left camera, and equals -1 if TMiRo is only detected in the right camera or neither of the cameras. Note that if TMiRo is not detected in a camera stream, it's corresponding C_x is equal 0.

If TMiRo is detected in both camera streams, then there is no issue and $B = 1$. If however TMiRo is detected only in the left camera, it is guaranteed that TMiRo's position in C_x^{est} should be negative (MiRo is obviously to the left of the scene's centre), but again, $B = 1$ ensures this is true since C_x^l is guaranteed to be < 0. Similarly, if MiRo only appears in the right camera stream, setting $B = -1$ ensures MiRo's C_x^{est} position is positive since C_x^r is guaranteed to be > 0.

Just to illustrate the above more clearly, consider the following: as a point is moved across the scene, from left to right, its C_x^{est} value would range from $C_x^{est} = -320$ ($C_x^l = 0$, $C_x^r = 0$, $B = 1$ and hence (5.15) equates to -320) to $C_x^{est} = 320$ ($C_x^l = 0$, $C_x^r = 320$, $B = -1$ and hence (5.15) equates to 320). The MiRo can be said to be looking directly ahead at the point when $C_x^{est} = 0$, and thus from Equation 5.15, $C_x^l + C_x^r = w$, as expected. The logic table for determining B is given in Table 5.1.

Letting now $C_x^{est} = x$ to make its use as a variable more convenient, once the com-

[5] C_x, C_y in either camera stream has its axis origin at the bottom left of the image. So if the centroid was in the centre position in a 320x240 image, it would be given as $(C_x, C_y) = (160, 120)$.

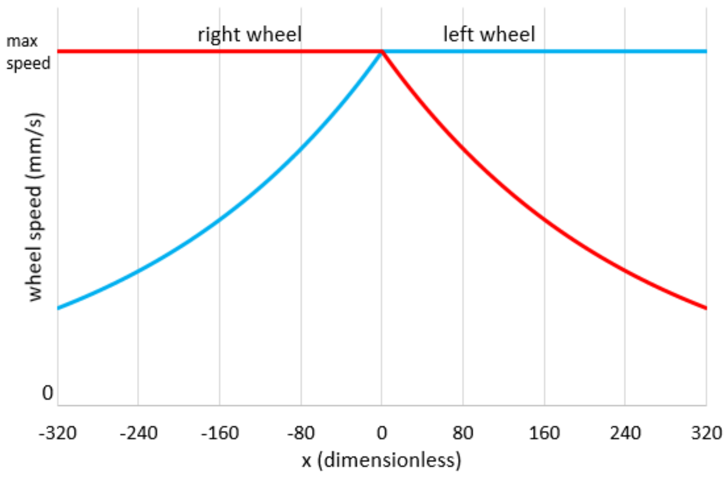


Figure 5.2: Plot of Equation 5.16, with wheel speeds as a function of the combined x value from Equation 5.15. γ is set at 0.004, which ensures that the difference between the left and right wheel does not change too dramatically, which would otherwise cause large overshoots.

bined x position for TMiRo is determined, MiRo's left and right wheel speeds are computed as a function of x in the following way, remembering x ranges from -320 to 320:

$$
\begin{aligned}
s_l(x) &= \begin{cases} s_{max}e^{\gamma x}, & \text{if } x \leq 0 \\ s_{max}, & \text{otherwise} \end{cases} \\
s_r(x) &= \begin{cases} s_{max}, & \text{if } x \leq 0 \\ s_{max}e^{-\gamma x}, & \text{otherwise} \end{cases}
\end{aligned} \tag{5.16}
$$

where γ is a tuning parameter that determines the rate of decay of the wheel speed from the maximum wheel speed, s_{max}. This causes MiRo to move straight ahead if $x = 0$, turns left with decreasing radius as x becomes more negative, and turns right with decreasing radius as x becomes more positive. There is a situation where x is undefined for when TMiRo is not detected in either camera. Under such circumstances, MiRo reverts to another state, such as performing a random walk, going into demo mode, or exhibiting any other pre-programmed behaviour. See Figure 5.15 for the plot of Equation 5.16 when $\gamma = 0.004$.

The methodology applied above works well for simple following, and having a tuning parameter allows further the option of optimising if necessary. However, the controller itself still lacked the ability to stop when it got too close to TMiRo in order to avoid colliding.

Collision Avoidance

Again, techniques similar to those proposed in the Braitenberg vehicle are applied, namely that of a logic gate having some dynamics. The aim was to have MiRo's velocity slow as it closely approached TMiRo, and then to stop if it got too close. The solution to this was to multiply the wheel speeds in Equation 5.16 by a minor modification of the logistic function to give the final wheel speeds as follows,

$$S_{l,r} = H(n)s_{l,r} \tag{5.17}$$

in which $H(n)$ is the modified logistic function, which gives outputs ranging from 1-0, given as,

$$H(n) = \frac{1}{1+e^{k(n-n_0)}} \tag{5.18}$$

where n is equal to the number of successfully thresholded pixels in the image. One may get a sense of the solution already if it is said that n_0 is set to the number of thresholded pixels that is deemed as TMiRo becoming 'too close'. k then determines how quickly MiRo alters its speed – too large and the result is MiRo dropping almost instantaneously from max speed to zero speed, which clearly is not a desirable outcome when MiRo's dynamics are considered; too small and MiRo doesn't change its speed quick enough to avoid collision. This coupled with the controller described above then completes the development process for the final controller.

5.2.2 Optimisation

It was then necessary to optimise the controller, by tuning the parameter γ in Equation 5.16 and the parameters k and n_0 in Equation 5.18.

Optimising γ

For assessing how varying γ influences MiRo's movements, MiRo was positioned at an angle of 45^{o} and a distance of 1.5m relative to a reference target – a blue cube (see Figure 5.3 for the experimental setup). The controller was then implemented at this starting position for a total of 10 trials, each with varying values of γ, ranging from 0.001 to 0.01.

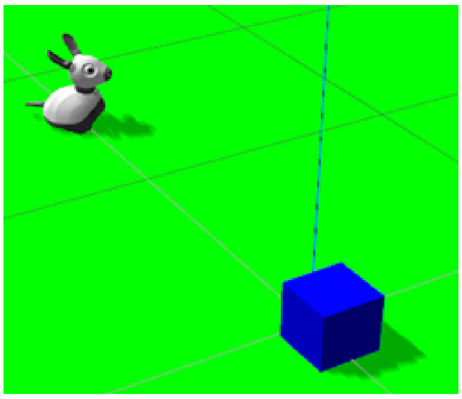


Figure 5.3: MiRo's starting position in the experimental setup for optimising γ. MiRo is positioned 1.5m away and at an angle of 45° from the target (blue box).

Then, the centroid position, C_x^{est}, is used to assess the controller's effectiveness. The desired value for C_x^{est} is 0, since at this position MiRo will be facing the target directly, and as such its forward velocity will be maximum (Equation 5.16). The criteria measured for finding most optimal γ were then determined as the following, where the final settling value should be $C_x^{est} = 0$:

- *Rise time*, T_r: Time taken to reach 90% of final settling value
- *Overshoot*, OS: Maximum overshoot above settling value
- *Settling time*, T_s: Time for C_x^{est} to settle to within +/- 20

The results of the trials are plotted in Figure 5.4, with the rise time, overshoot and settling times given in Table 5.2. It was found that a value of $\gamma = 0.004$ seems to be most appropriate, as its rise time is quick enough (within 40% of the quickest rise time), overshoot is reasonable (within 19% of the smallest OS) and has the shortest settling time.

Table 5.2: Rise time, overshoot and settling time for different values of γ. Note that the other parameters in Equation 5.18 have no effect on these results, as MiRo does not reach the object.

γ	T_r (s)	OS	T_s (s)
0.001	> 6	0	> 6
0.002	3.20	37	> 6
0.003	1.80	38	4.4
0.004	1.32	44	2.5
0.005	1.16	54	3.5
0.006	1.16	63	> 6
0.007	1.10	80	> 6
0.008	0.95	85	> 6
0.009	0.98	89	> 6
0.01	0.95	113	> 6

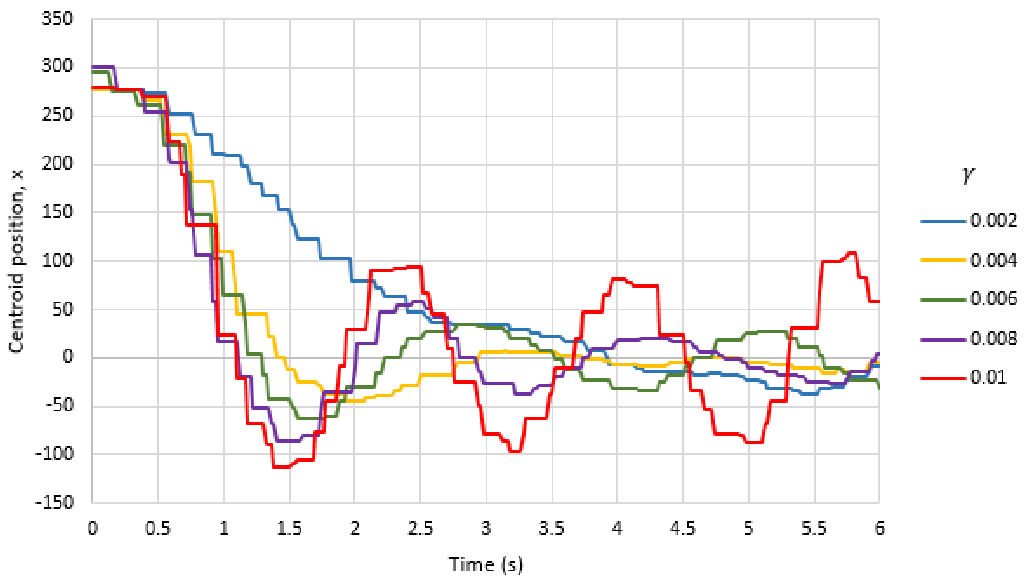


Figure 5.4: Plot of centroid position (Equation 5.15) against time when varying γ in Equation 5.16. Only results for γ in steps of 0.002 is given to avoid having too many plots. C_x^{est} begins at 300 due to MiRo's starting position in Figure 5.3, before dropping as the controller computes MiRo's wheel velocities. The stepped changes are due to the difference in camera frame rates ($\approx$5fps) and the simulation rate ($\approx$50fps)

Optimising k and n_0

Optimising for k and n_0 requires a little more consideration, and experimentation wasn't the most efficient method for the optimisation process. Instead, it is far more beneficial to look directly at plots that Equation 5.18 give for various value of k and n_0. Then one could set a desired stopping distance, which is converted into a desired 'pixel area'

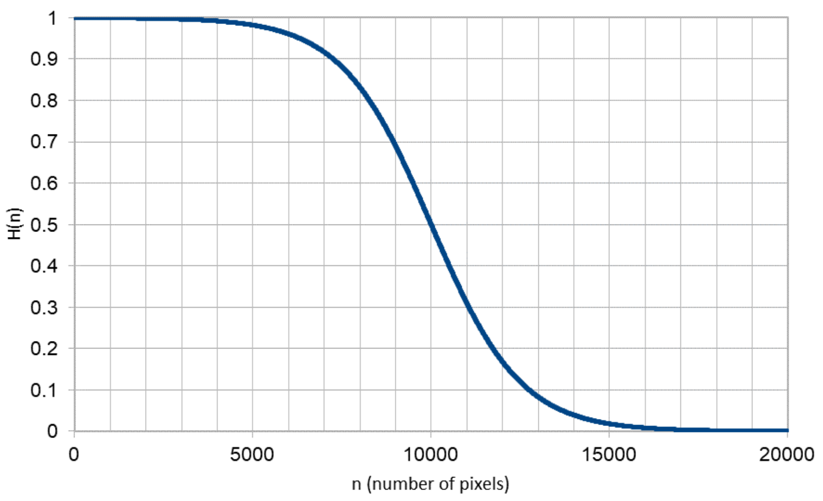


Figure 5.5: Plot of Equation 5.18 with $k = 0.0008$ and $n_0 = 10,000$. k was selected so that saturation was reasonable (i.e. reaches 0 within good time) but without a having a steep curve. n_0 was then selected as 10,000 so that H reached 0 at the desired value of $n = 15,000$.)

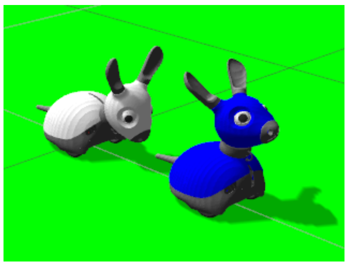


Figure 5.6: MiRo's stopping distance behind TMiRo with the optimised parameters.

(as MiRo approaches TMiRo, TMiRo's 'pixel area' increases). k is then selected to give a desired speed change – 0.0008 gave an acceptable plot that saturated within a reasonable time, and did not give too steep a curve. n_0 is then simply set so that Equation 5.18 saturates at 0 as n approached the desired value. For instance, with $k = 0.0008$, and having a desired stopping distance with a pixel value of $n = 15,000$, n_0 could be set at approximately 10,000 so that $H(n)$ reached 0 at close to $n = 15,000$. See Figure 5.5 for the final plot with $k = 0.0008$ and $n_0 = 10,000$. Applying the above parameters resulted with an effective controller that allowed MiRo to stop at reasonable distances from TMiRo without large overshoots. See Section 5.2.4 for an analysis of the final controller.

5.2.3 Losing Track of TMiRo

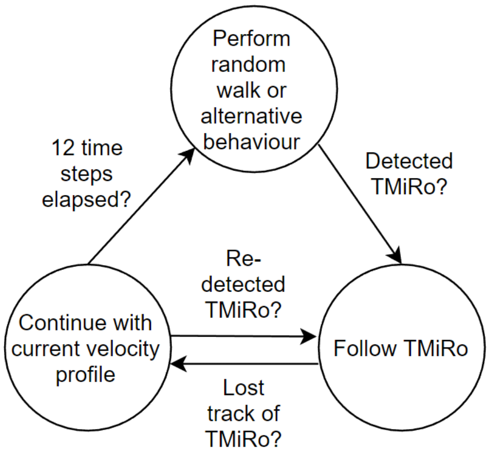
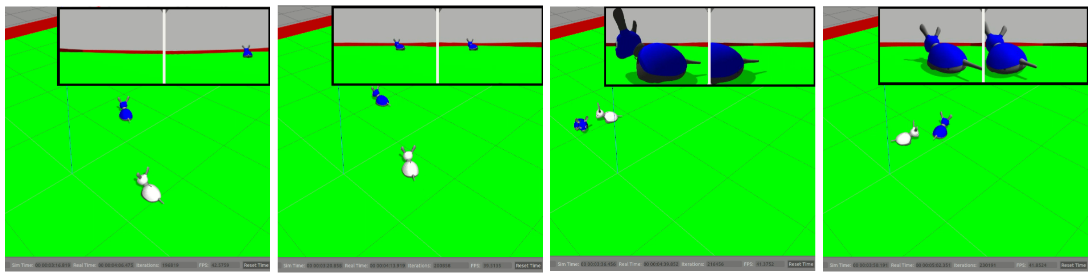


Figure 5.7: A simple finite state machine.

Although the vision algorithms had good success with the detection of TMiRo, at approximately 90% accuracy for each, the 10% failure rate becomes significant when one considers the number of time steps – at 4fps camera rate, then after 2.5 seconds MiRo may lose track of TMiRo an average of once. Rather than MiRo stopping or going into a random walk immediately after losing track of TMiRo, both of which may result in losing sight of TMiRo altogether, a more sensible strategy would be to follow a simple finite state machine type procedure as follows: Initially, when TMiRo is undetected, MiRo performs a random walk or, alternatively, exhibits another preprogrammed behaviour. Upon detecting TMiRo, MiRo commences with *following* TMiRo. If however TMiRo was to become lost, MiRo would continue with its previous velocity profile in the hope it may rediscover TMiRo's position. If it is unsuccessful in its rediscovery after 12 time steps (or 3s for a 4fps camera rate), then MiRo returns to a random walk / alternative behaviour state. This process is summarised in Figure 5.7.



(a) (b) (c) (d)

Figure 5.8: Testing of the initial controller. The two images in the top right of each image are the camera streams from MiRo. (a) Starting positions for MiRo (white) and TMiRo (blue). (b) TMiRo begins moving in a circular, counter-clockwise path. MiRo's controller is activated manually. MiRo has detected TMiRo and is moving towards it. (c) MiRo stops before hitting TMiRo's side, and waits to follow in behind. (d) MiRo is now successfully following TMiRo.

5.2.4 Analysis

The Python source code implemented for the initial controller is given in Appendix A.5. The controller performed reasonably well, allowing MiRo to successfully detect TMiRo, and then to move towards TMiRo until reaching a stopping distance determined by the collision avoidance method above. A typical example of the following behaviour can be seen in Figure 5.8. MiRo generally remained approximately half a MiRo body length behind TMiRo at all times.

Its computation time to perform the image processing is reasonable, with each run through of the script giving an average run time of 6.82ms with standard deviation of 0.09ms (statistics computed from 10 normal runs). Given MiRo has a frame rate of 8fps with the 320x240 resolution, then there is plenty of time for the image processing operations to complete: 6.82ms $<<$ 125ms (8fps = 0.008 frames/ms = 125 ms/frame). Note however that this was conducted off-board the MiRo and within the simulator on an Intel i7 4770K, 3.5GHz CPU. For comparison, MiRo's P3 processor clocks at 1GHz.

There are of course some issues with the initial controller, in that the controller cannot account for noise. This was more noticeable in experimental work, as for instance sending a forward velocity often resulted in MiRo veering slightly to one side. This problem is highlighted further in Chapter 6 and addressed in Chapter 7.

A final note on this controller is that it can, if necessary, be utilised with any number of computer vision detection methods. So long as a method to predict point positions in each camera and distance of TMiRo are available, there should be no problems with adapting the parameters of the above controller to suit.

Chapter 6

Real-World Experimentation in MiRo

In order to display the effectiveness and feasibility of the optimised controller from simulation into a real-world setting, a short experiment was performed with MiRo. A small blue item was used for MiRo to detect (allowing for control over the environment), and then MiRo was positioned some distance away and at an angle from the blue object – see Figure 6.1a for the experimental set-up. Upon implementation of the main controller, MiRo showed a clear ability to generate the necessary velocities in order to move towards the object. However, in reaching the object MiRo can be seen to run past it, as the collision avoidance mechanism had not been calibrated for use with the small object. MiRo did however remain at rest once the object was no longer in MiRo's field of view, indicating clearly MiRo's movement was due to an "attraction" to the blue object, and not due to other sources. The results for this experiment are given in Figure 6.1. Under controlled scenarios such as these, MiRo successfully moved towards the object 100% of the time with smooth trajectories and zero overshoot (barring networking issues).

This suggests the controller works well when the conditions of the environment are controlled. Possible issues in having MiRo *follow* another MiRo therefore would likely *not be due to problems with the controller, but would be a result of the vision algorithms not performing effectively*.

It may be noticeable in Figure 6.1 that MiRo doesn't quite position itself head on with the blue object, rather it is slightly to the right of it. This issue was not due to the controller, but was caused by minor unwanted wheel-speed differentials that occurred even when sending the same speeds to the wheels. Whilst minor, it is worth mentioning

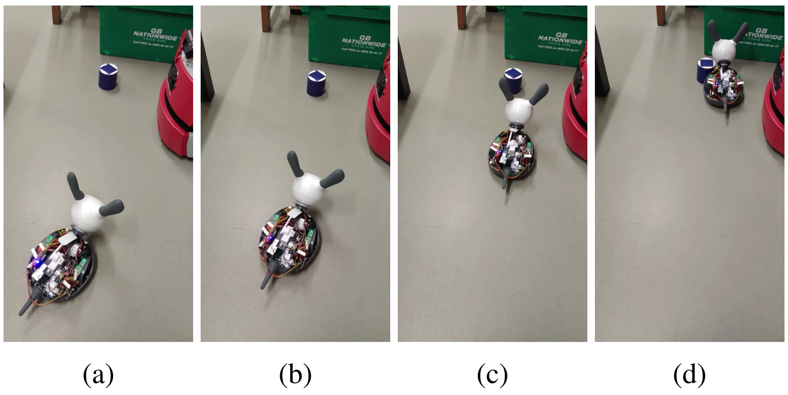


Figure 6.1: Running the controller on MiRo for following a blue object. (a) MiRo's starting position. (b) Shortly after implementing the controller, MiRo begins moving towards the blue object. (c) MiRo reaches closes in on the blue object. (d) MiRo has ran past the blue object and remains stationary as the object is no longer in its field of view.

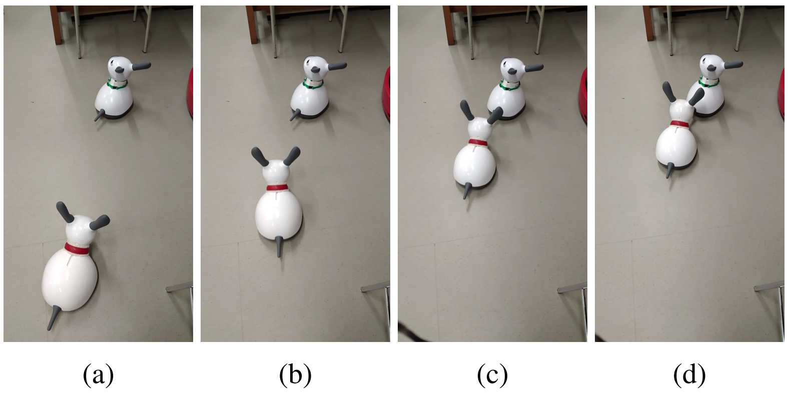


Figure 6.2: Running the controller on MiRo using the ROI threshold method. (a) MiRo's starting position. (b) Shortly after implementing the controller, MiRo begins moving towards a "distraction" on its left. (c) MiRo soon turns back towards TMiRo. (d) MiRo's collision avoidance stops MiRo close to 10cm behind TMiRo.

as these slight issues adds further emphasis for the need to develop a more adaptive control strategy.

Following this success, the controller was then tested using the ROI threshold method for attempting to follow another MiRo in the same environment, along with the collision avoidance mechanism. This worked with remarkable success as shown in Figure 6.2, with MiRo moving towards TMiRo, before stopping a short distance behind due to the collision avoidance mechanism. MiRo does temporally become "distracted" by other white objects to its left (Figure 6.2b), but overcomes these and returns to TMiRo. Utilising the two vision algorithms should improve the robustness of the detection in order to eliminate these "distractions". Again, it is important to note that testing of the vision algorithms is separate from testing of the controller – the results and discussion for testing of the vision algorithms were given in Sections 4.4 and 4.5.

Chapter 7

Discussion and Conclusion

7.1 Vision Algorithms and the Controller

Both vision algorithms are designed such that they both provide compatibility with the main controller developed above. There should therefore be no issues with any combination of vision algorithm and controller.

The two vision algorithms perform to a high standard, both achieving successful recognition rates of close to 90%. However, both algorithms offer different means of classification. Algorithm 1 for instance is a simple binary classifier, giving only a 'yes' or 'no' answer to the question of whether MiRo exists in a ROI. Algorithm 2, however, offers something rather more interesting, particularly in the way it could be used to affect the behaviour of MiRo. Because it is probabilistic, it is not necessary to present it as a binary classifier, but rather to let the computed posterior probabilities determine the 'probability' that MiRo decides to follow the detected TMiRo, or not. An example would be the following: MiRo detects a potential TMiRo in a scene, and has assigned it with a posterior probability of 0.4. Rather than MiRo deciding to avoid following the object because the probability is less than half, perhaps MiRo could decide to follow the object with a probability of 0.4, the same probability it assigned to it being a TMiRo. MiRo's behaviour would become stochastic under such a rule, and to an outside observer it would appear as if MiRo had a 'mind of its own' – in some situations it decides to follow TMiRo unrelentingly, whilst at other times MiRo would seem indifferent to TMiRo. The Bayesian estimator could therefore enable particularly interesting behaviours within MiRo, with the *prior* probability also having use as a control mechanism to affect the estimator's probabilities.[1]

7.2 Adaptive Control

An adaptive controller is left as a proposal for further work, and the initial development of such a controller is documented in Section A.6 of the Appendices. Some discussion should be made related to this proposal here though. The controller's aim is to overcome the issue mentioned above, namely the wheel speed differentials. It is hoped that MiRo could learn, given a certain centroid and its rate of change, which wheel speeds would be optimal for these set of states. A reinforcement learning mechanism is implemented with centroid positions and centroid velocities used as reward signals – the goal is to give greater rewards as the centroid and its velocity moves towards zero, which would indicate perfect *following*. Whether it works effectively is as yet unknown, but its current form still lacks perhaps one crucial element for it being an even more robust controller – it can only consider the differential speeds between MiRo and TMiRo, and cannot compute TMiRo's global velocity (relative to the environment's reference frame). This is crucial in order to distinguish whether TMiRo's movements are due to MiRo's own movements, TMiRo's movements, or both. The original controller doesn't suffer this problem, as centroid rate of change is not a parameter used in its computations.

It could be possible to incorporate this into the proposed controller however. Consider that there is some mapping that can convert wheel speeds to a relative camera speed, using MiRo's angular velocity as given by Equation 5.2,

$$\dot{c}_{map} = a(\omega_R - \omega_L) \tag{7.1}$$

where a is the parameter to map angular rotation to a useful relative image pixel rate of change. The global rate of change of an interest point, or the centroid, $\dot{c}_g$, can then be estimated by subtracting the mapped relative speed above from the relative speed. The mapped speed could be estimated through experimental work by measuring how a still object's position in an image changes with MiRo's angular velocity. The mapped speed

[1]Consider another interesting possibility, in that MiRo's current 'mood' (MiRo has the ability to exhibit 'affective states' – see (Collins, Prescott, and Mitchinson, 2015a)) affects the *prior* probability.

would then offer further useful information for determining how wheel speeds should be set. This wouldn't just be functional for the adaptive controller proposed, but for potentially other tasks too, such as velocity predictions in a filter.

7.3 A Final Discussion on Collective Behaviour

The work conducted as part of this project provides the potential for developing further types of collective behaviours, aside from *following*, to be explored in MiRo. Chapter 2's review of the literature on collective behaviours should offer a wide range of potential collective behaviour research areas, with group herding, flocking, following, communication strategies (direct, stigmergic and non-communicative), analysis at the micro-level, macro-level, and mathematical models of group behaviours all discussed.

Of most interest could be the type of flocking behaviour exhibited in selfish-herd theory as studied by King et al. (2012) in sheep and as studied by Calvao and Brigatti (2014) in robots, both discussed in Chapter 2. Particularly necessary for implementing these behaviours was a method for the robots to detect each other and to estimate their distances from one another. A solution to the first has been offered as part of the work documented in this thesis; as for the second, this has partly been achieved, yet there remains the need for a more robust depth estimator, and further work on depth estimation should therefore be carried out, for which the initial theory is already laid out in Section 3.2.3.

Aside from *following*, herding and flocking, the vision algorithms are also well suited as a MiRo face-detector. Whilst face-detectors are already rather popular within literature and in open source vision modules, applying one for MiRo in order to detect another MiRo's face would be simple (MiRo has only one standard face shape and colour), and beneficial for exploring collective behaviours that require knowledge of MiRo's orientation, as well as certain types of communication.

7.4 Looking Back at the Project Objectives

The work documented thus far has hopefully made it clear that all (with one exception to be discussed) of the basic objectives laid out in Section 1.4 have been successfully achieved. A literature review on a wide variety of collective behaviours and vision techniques has been conducted; two vision algorithms have successfully been developed to allow "MiRo-detection", with each compared using computation times and accuracy; a *following* technique has been developed and was subsequently successfully tested on MiRo.

The only objective not carried out was in incorporating MiRo's current motion detection ability with the "MiRo-detection" algorithm. Rather, a different strategy for motion tracking was implemented by simply taking a point position (centroid) and using the point to track MiRo's position between camera frames.

The advanced objective has not been conducted, but is left open for further work.

7.5 Conclusions

The main aim of this project was to have MiRo *following* another MiRo using only its vision, and this has successfully been accomplished under controlled settings. Reiterating the closing section of the discussion above, all but one of the project's objectives have been adequately accomplished, including literature reviews on collective behaviours and vision techniques, the development of two vision algorithms for "MiRo-detection" and their comparisons, and of a control strategy to allow MiRo to follow another MiRo. Finally, this has been tested with some success on the real-world MiRo.

The two vision algorithms – the first using grey-scale image histograms with a perceptron neural network and the second using the SURF algorithm with Bayesian inference – perform respectably well for detecting MiRo, with both performing close to 90% correct detections on a set of training images. The first algorithm slightly outperformed the second both in terms of detection (90.1% vs 87.8%) and in terms of relative execution time (0.271ms vs 84.729ms). An adaptive threshold has efficiently been applied for extracting a region of interest in order to speed up the vision algorithms. Yet despite this, the Bayesian estimation computations of algorithm 2 take a considerable amount of processing time, hence the reason for the overall slow execution time of algorithm 2 versus algorithm 1.

For developing a following strategy in MiRo, the controller then computes the centroid positions of the detected MiRo in each of MiRo's camera streams, to which the centroid values are mapped into wheel velocities. The parameters in this mapping have been optimised to give the most efficient *following* behaviour. This controller was applied with success on the real-world MiRo robot when using only the ROI algorithm.

The developed vision algorithms do have important implications for future research within MiRo, and are particularly useful as tools for studying more complex forms of collective behaviours. Phenomena such as flocking, herding, other physical interactions and certain forms of communication strategies could be explored now that there exists vision algorithms that can perform "MiRo-detection".

The work presented in this thesis will continue development post-completion of the author's MSc degree in collaboration with Consequential Robotics, and hence there are areas of important further development to be mentioned; these are covered in the next section.

7.5.1 Further Work

MiRo is a versatile robot, and there are many number of further research avenues that could be pursued on its platform. With regards to work conducted on this project however (that will, as mentioned above, be undertaken post completion of the project), there is one immediate focus for further work that should be considered: to fully test both of the vision algorithms in real-time *on-board* the actual MiRo. Although the development and testing was conducted off-board with images captured directly from MiRo's cameras, and therefore the algorithms should transfer comfortably for use in MiRo, it is still necessary to test how the algorithms perform when noise, image disturbances and lost frames are introduced, as well as ensuring MiRo has the computing resources (speed and memory) to run both on-board.

Other work should involve the development of a method for finding the absolute velocity of an image interest point / centroid; developing further the adaptive controller; and then combining these two developments to enable a more robust controller. There is also the necessity to estimate object depth using MiRo's dual cameras, which again should be no more difficult than performing experimentation and/or the development of a mathematical model.

Finally, the herding/flocking behaviours of the advanced objective set out at the beginning of this project should not be forgotten, and through use of the vision algorithms developed here these offer an interesting area for future research.

Chapter 8

Project Management and Self Review

8.1 Project Management

Upon completion of the Interim Report, 4 milestones were set for the forthcoming project duration, which were then inserted into the project timing plan (see the 'purple' milestones in Appendix A.7). Although progress on each milestone was intended to be undertaken upon successful completion of a previous milestone, it later became possible to undertake a later milestone before thorough completion of the preceding milestones. An example of this was how milestone 3 – developing a MiRo-MiRo following strategy – could be performed without relying on the full completion of the algorithms in milestone 1, since only a simple threshold detection was necessary.

Furthermore, milestone 2 actually became unnecessary to complete, as adequate tracking was achievable without the need to incorporate MiRo's original motion detector. Hence no work was conducted towards achieving this milestone.

This was, however, partly a reflection of how involved the vision algorithms had become, and although the original timing plan had allotted 2 weeks for full completion of the algorithms, it actually ran for most of the project period.

A final note is that the thesis itself was written in incremental stages as originally planned. As such, the basic objectives were completed two weeks ahead of the project deadline, allowing some time to consider the development of an adaptive controller strategy. Completed also were constant and regular log book entries[1].

The project timing plan Gantt chart is given in Appendix A.7 with a comparison of the original timings vs the actual. Table 8.1 summarises the statuses of the milestones.

[1]Available upon request

Table 8.1: Summary of the project's milestones

Milestone	Planned completion	Actual completion	Notes
1. Apply two or three detection algorithms to achieve "MiRo-detection"	25/06/2017	13/08/2017	Generating a robust detector took longer than planned, but had little effect on the development of the controller
2. Incorporate MiRo's current motion detection ability with the "MiRo-detection" algorithm to track MiRos	09/07/2017	N/A	Tracking was completed independently with the vision algorithms
3. Develop a MiRo-MiRo following technique and perform simulations	23/07/2017	09/07/2017	Completed ahead of schedule as a simpler vision algorithm could be used in its development
4. Test the winning algorithm on the physical robot	30/07/2017	13/08/2017	Delayed by two weeks due to the vision algorithms delay

8.2 Self Review

Learning and the application of learned materials are invariably important conditions for the completion of a successful research project, as is the ability to find potential connections between ideas and concepts that to most seem disconnected – the basis of creativity. Finding such connections are possible perhaps only through careful and fluent understandings of separate subjects. Whilst it was important that this project produced useful outcomes, it was also a creative experiment in how different scientific concepts – zoology, computational neuroscience, classical systems modelling and control, biologically inspired control, and computer vision, could be coherently united to achieve the project aim.

If there has been anything of significance learned from undertaking this project, whether it be proper time management, importance of keeping log-books, being an independent researcher, or effective reviewing of literature, it has probably been this: that in order to fully understand a particular topic, it would undoubtedly require substantial amounts of time; but also that new insights can often come from the application of a concept in one subject to the rules of another. This requires having comprehensive understandings of multiple subject areas, which no doubt would be an inescapably arduous proposition. Hence there can be no other way than for effective collaboration amongst teams and individuals. As a student with a preference for solitude, this understanding has inspired me more than anything to increase my communications with others in order to achieve the ultimate aim of knowledge, understanding and discovery.

References

Alpaydin, Ethem (2010). *Introduction to Machine Learning*. MIT Press. Chap. The Universal Features of Cells on Earth.

Althnian, Alhanoof and Arvin Agah (2016). “Evolving goal-driven multi-agent communication: what, when, and to whom”. In: *Evolutionary Intelligence* 9.4, pp. 181–202.

Altuntas, Nihal et al. (2016). “Reinforcement learning-based mobile robot navigation”. In: *Turkish Journal of Electrical Engineering & Computer Sciences* 24.3, pp. 1747–1767.

Arkin, Ronald C. (1998). *BehaviorâĂŚBased Robotics*. Massachusetts Institute of Technology.

Ballerini, Michele et al. (2008). “Empirical investigation of starling flocks: a benchmark study in collective animal behaviour”. In: *Animal Behaviour* 76.1, pp. 201 –215.

Baumberg, A. (2000). “Reliable feature matching across widely separated views”. In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662)*. Vol. 1, pp. 774–781.

Bay, Herbert, Tinne Tuytelaars, and Luc Van Gool (2006). “SURF: Speeded Up Robust Features”. In: *Computer Vision – ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I*. Ed. by Aleš Leonardis, Horst Bischof, and Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 404–417.

Bellman, Richard E. (1957). *Dynamic Programming*. Princeton University Press.

Biederman, Irving (1987). “Recognition-by-Components: A Theory of Human Image Understanding”. In: *Psychological Review* 92.2, pp. 115–147.

Biederman, Irving and Ginny Ju (1988). “Surface versus edge-based determinants of visual recognition”. In: *Cognitive Psychology* 20.1, pp. 38 –64.

Braitenberg, Valentino (1984). *Vehicles: Experiments in Synthetic Psychology*. MIT Press.

Brooks, Rodney (1981). “Symbolic reasoning among 3-D models and 2-D images”. In: *Artificial Intelligence* 17.1, pp. 285 –348.

— (1991). “Artificial Life and Real Robots”. In: *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*. Ed. by Francisco J. Varela and Paul Bourgine. MIT Press, pp. 3–10.

Brooks, Rodney (1999). *Cambrian Intelligence: The Early History of the New AI*. Massachusetts Institute of Technology.

Bruske, Jorg, Ingo Ahrns, and Gerald Sommer (1997). “An integrated architecture for learning of reactive behaviors based on dynamic cell structures”. In: *Robotics and Autonomous Systems* 22.2, pp. 87 –101.

Burgard, W. et al. (2000). “Collaborative multi-robot exploration”. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 1, 476–481 vol.1.

Caldart, Vinícius Matheus, Samanta Iop, and Sonia Zanini Cechin (2014). “Social interactions in a neotropical stream frog reveal a complex repertoire of visual signals and the use of multimodal communication”. In: *Behaviour* 151.6, pp. 719–739.

Calvao, Angelo M. and Edgardo Brigatti (2014). “The Role of Neighbours Selection on Cohesion and Order of Swarms”. In: *PLoS One* 9.5, e94221.

Campion, G., G. Bastin, and B. D’Andrea-Novel (1996). “Structural Properties and Classification of Kinematic and Dynamic Models of Wheeled Mobile Robots”. In: *IEEE Transactions on Robotics and Automation* 12, pp. 47–62.

Cazenille, L., N. Bredeche, and J. Halloy (2016). “Automated optimisation of multi-level models of collective behaviour in a mixed society of animals and robots”. In: *ArXiv e-prints*. arXiv: `1602.05830 [nlin.AO]`.

Ciresan, Dan et al. (2012). “Multi-column deep neural network for traffic sign classification”. In: *Neural Networks* 32, pp. 333 –338.

Collins, E.C., T.J. Prescott, and B. Mitchinson (2015a). “Saying It with Light: A Pilot Study of Affective Communication Using the MIRO Robot”. In: *Biomimetic and Biohybrid Systems, Living Machines 2015*. Lecture Notes in Computer Science 9222, pp. 243–255.

Collins, E.C. et al. (2015b). “MIRO: A Versatile Biomimetic Edutainment Robot”. In: *Proceedings of the 12th International Conference on Advances in Computer Entertainment Technology*, pp. 631–634.

Davies, E. R. (2012). *Computer and Machine Vision: Fourth Edition*. Elsevier Inc.

Dayan, Peter and L. F. Abbot (2001). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press. Chap. Plasticity and Learning, pp. 281–283.

Di Marzo Serugendo, Giovanna, Marie-Pierre Gleizes, and Anthony Karageorgos (2005). “Self-organization in multi-agent systems”. In: *The Knowledge Engineering Review* 20.2, 165âĂŞ189.

Dorigo, Marco and Marco Colombetti (1998). *Robot Shaping: An Experiment in Behavior Engineering*. MIT Press.

Eichenseer, A., M. Batz, and A. Kaup (2016). “Motion estimation for fisheye video sequences combining perspective projection with camera calibration information”. In: *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 4493–4497.

Ejung, Edgard et al. (2016). “The Influence of Human Body Orientation on Distance Judgments”. In: *Frontiers in Psychology* 7.

Fagg, A. H., D. Lotspeich, and G. A. Bekey (1994). “A reinforcement-learning approach to reactive control policy design for autonomous robots”. In: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. Vol. 1, pp. 39–44.

Fang, Fen and Yong Tsui Lee (2012). “3D reconstruction of polyhedral objects from single perspective projections using cubic corner”. In: *3D Research* 3.2, p. 1.

Ferreira, André et al. (2008). “An approach to avoid obstacles in mobile robot navigation: the tangential escape”. In: *Sba: Controle & Automação Sociedade Brasileira de Automatica* 19, pp. 395 –405.

FESTO (2017). *Technical Specifications*. [Online: accessed 15-March-2017]. URL: `http://www.festo.com/en/bionickangaroo`.

Fine, Benjamin T. and Dylan A. Shell (2013). “Unifying microscopic flocking motion models for virtual, robotic, and biological flock members”. In: *Autonomous Robots* 35.2, pp. 195–219.

From Animals to Animats (1990–2016). “From Animals to Animats: 1–14”. In: *International Conference on Simulation of Adaptive Behavior*.

Gautrais, Jacques et al. (2008). “Deciphering Interactions in Moving Animal Groups”. In: *PLoS Computational Biology* 8.9, e1002678.

Gil-Jiménez, Pedro et al. (2016). “Estimating the focus of expansion in a video sequence using the trajectories of interest points”. In: *Image and Vision Computing* 50, pp. 14 –26.

Girshick, Ross B. et al. (2013). “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *CoRR*.

Gonzalez, Rafael and Richard Woods (2010). *Digital Image Processing*. Pearson Education. Chap. Wavelets and Multiresolution Processing, pp. 483–546.

Gurney, Kevin et al. (2004). “Computational models of the basal ganglia: from robots to membranes”. In: *Trends in Neurosciences* 27.8, pp. 453 –459.

Halloy, J. et al. (2007). “Social Integration of Robots into Groups of Cockroaches to Control Self-Organized Choices”. In: *Science* 318.5853, pp. 1155–1158.

Hamilton, W.D. (1971). “Geometry for the selfish herd”. In: *Journal of Theoretical Biology* 31.2, pp. 295 –311.

He, Bo, Hongen Ren, and Wei Kan (2010). “Design and simulation of Behavior-Based Reactive Decision-making Control System for autonomous underwater vehicle”. In: *2010 2nd International Conference on Advanced Computer Control*. Vol. 5, pp. 647–651.

Hebb, Donald (1949). *The organization of behavior : a neuropsychological theory*. Wiley.

Hinz, Cornelia et al. (2013). “Kin recognition in zebrafish, Danio rerio, is based on imprinting on olfactory and visual stimuli”. In: *Animal Behaviour* 85.5. Including Special Section: Behavioural Plasticity and Evolution, pp. 925 –930.

Hogg, D., F. Martin, and R. Resnick (1991). “Braitenberg Creatures”. In: *MIT Epistemology and Learning Memorandum* 13.

Horn, Berthold K.P. and Brian G. Schunck (1981). “Determining optical flow”. In: *Artificial Intelligence* 17.1, pp. 185 –203.

Horn, G., B. J. McCabe, and J. Cipolla-Neto (1983). “Imprinting in the domestic chick: The role of each side of the hyperstriatum ventrale in acquisition and retention”. In: *Experimental Brain Research* 53.1, pp. 91–98.

Hurst, Jane L. et al. (2001). “Individual recognition in mice mediated by major urinary proteins”. In: *Nature* 414, pp. 631–634.

Isaeva, V. V. (2012). “Self-organization in biological systems”. In: *Biology Bulletin* 39.2, pp. 110–118.

Ishikawa, Hiroshi and Davi Geiger (1998). “Occlusions, discontinuities, and epipolar lines in stereo”. In: *Computer Vision — ECCV’98: 5th European Conference on Computer Vision Freiburg, Germany, June, 2–6, 1998 Proceedings, Volume I*. Ed. by Hans Burkhardt and Bernd Neumann. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 232–248.

Jokinen, Olli and Henrik Haggrén (1998). “Statistical analysis of two 3-D registration and modeling strategies”. In: *{ISPRS} Journal of Photogrammetry and Remote Sensing* 53.6, pp. 320 –341.

Kernbach, Ed S. (2012). *Handbook of Collective Robotics: Fundamentals and Challenges*. Pan Stanford Publishing.

King, Andrew J. et al. (2012). “Selfish-herd behaviour of sheep under threat”. In: *Current Biology* 22.14, R561 –R562.

Kube, C. R. and Hong Zhang (1996). “The use of perceptual cues in multi-robot box-pushing”. In: *Proceedings of IEEE International Conference on Robotics and Automation*. Vol. 3, 2085–2090 vol.3.

Kube, C. Ronald and Hong Zhang (1993). “Collective Robotics: From Social Insects to Robots”. In: *Adaptive Behavior* 2.2, pp. 189–218.

Landgraf, T. et al. (2010). “A biomimetic honeybee robot for the analysis of the honeybee dance communication system”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3097–3102.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). “Deep Learning”. In: *Nature* 521, pp. 436–444.

Lindeberg, Tony (1994). “Scale-space theory: A basic tool for analysing structures at different scales”. In: *Journal of Applied Statistics* 21.2, pp. 225–270.

Lorenz, Konrad Z. (1937). “The Companion in the Bird’s World”. In: *The Auk* 54.3, pp. 245–273.

Lowe, David G. (2004). “Distinctive Image Features from Scale-Invariant Keypoints”. In: *International Journal of Computer Vision* 60.2, pp. 91–110.

Maekawa, Fumihiko et al. (2006). “Imprinting modulates processing of visual information in the visual wulst of chicks”. In: *BMC Neuroscience* 7.75.

Maestripieri, Dario (1996). “Gestural Communication and Its Cognitive Implications in Pigtail Macaques (Macaca Nemestrina)”. In: *Behaviour* 133.13, pp. 997–1022.

Marocco, Davide, Angelo Cangelosi, and Stefano Nolfi (2003). “The Emergence of Communication in Evolutionary Robots”. In: *Philosophical Transactions: Mathematical, Physical and Engineering Sciences* 361.1811, pp. 2397–2421.

Matarić, Maja J (1998). “Behavior-based robotics as a tool for synthesis of artificial behavior and analysis of natural behavior”. In: *Trends in Cognitive Sciences* 2.3, pp. 82 –86.

Matarić, Maja J. (2001). “Learning in behavior-based multi-robot systems: policies, models, and other agents”. In: *Cognitive Systems Research* 2.1, pp. 81 –93.

McInroe, Benjamin et al. (2016). “Tail use improves performance on soft substrates in models of early vertebrate land locomotors”. In: *Science* 353, pp. 154–158.

Mérmoud, Gregory (2012). “Design, Modeling and Optimization of Stochastic Reactive Distributed Robotic Systems”. eng. PhD thesis. Lausanne: ENAC. DOI: `10.5075/epfl-thesis-5392`.

Mitchinson, B. and T.J. Prescott (2016). “MIRO: A robot âĂIJMammalâĂİ with a biomimetic brain-based control system”. In: *Biomimetic and Biohybrid Systems. 5th International Conference, Living Machines*, pp. 453 –459.

Najafi, M. H. and M. E. Salehi (2016). “A Fast Fault-Tolerant Architecture for Sauvola Local Image Thresholding Algorithm Using Stochastic Computing”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.2, pp. 808–812.

Nakamori, T. et al. (2013). “Imprinting modulates processing of visual information in the visual wulst of chicks”. In: *Development, Growth and Differentiation* 55.1, pp. 198–206.

Narendra, K.S. and A. M. Annaswamy (1989). *Stable Adaptive Systems*. Prentice Hall.

Negahdaripour, Shahriar and Berthold K.P Horn (1989). “A direct method for locating the focus of expansion”. In: *Computer Vision, Graphics, and Image Processing* 46.3, pp. 303 –326.

Ngo, Trung Dung and Henrik Schiøler (2008). “Truly autonomous robots: hardware design for an energy trophallactic robot”. In: *Artificial Life and Robotics* 12.1, pp. 335–345.

Nolfi, Stefano (2005). “Emergence of communication in embodied agents: co-adapting communicative and non-communicative behaviours”. In: *Connection Science* 17.3-4, pp. 231–248.

Okubo, Akira (1986). “Dynamical aspects of animal grouping: Swarms, schools, flocks, and herds”. In: *Advances in Biophysics* 22, pp. 1 –94.

OpenCV (2015a). *Color Conversions*. [Online: accessed 20-July-2017]. URL: `http://docs.opencv.org/3.1.0/de/d25/imgproc_color_conversions.html`.

— (2015b). *Structural Analysis and Shape Descriptors*. [Online: accessed 27-July-2017]. URL: `http://docs.opencv.org/3.1.0/d3/dc0/group__imgproc__shape.html#ga17ed9f5d79ae97bd4c7cf18403e1689a`.

Orban, Guy A. (1992). “The Analysis of Motion Signals and the Nature of Processing in the Primate Visual System”. In: *Artificial and Biological Vision Systems*, pp. 24–56.

Otsu, N. (1979). “A Threshold Selection Method from Gray-Level Histograms”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1, pp. 62–66.

Picard, Gauthier and Marie-Pierre Gleizes (2003). “An Agent Architecture to Design Self-Organizing Collectives: Principles and Application”. In: *Adaptive Agents and Multi-Agent Systems: Adaptation and Multi-Agent Learning*. Ed. by Eduardo Alonso, Daniel Kudenko, and Dimitar Kazakov. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 141–158.

Rescorla, R. A. and A. R. Wagner (1972). “A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement”. In: *Classical conditioning: current research and theory*. Ed. by A. H. Black and W. F. Pokasy. New York: Appleton-Century-Crofts, pp. 64–98.

Rodríguez, A., J. Gómez, and A. Diaconescu (2015). “Foraging-Inspired Self-Organisation for Terrain Exploration with Failure-Prone Agents”. In: *2015 IEEE 9th International Conference on Self-Adaptive and Self-Organizing Systems*, pp. 121–130.

Rosenblatt, Frank (1962). *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Spartan Books.

Rushton, Simon K. and Paul A. Warren (2005). “Moving observers, relative retinal motion and the detection of object movement”. In: *Current Biology* 15.14, R542 –R543.

Santamaría, Juan Carlos and Ashwin Ram (1996). “Multistrategy Learning of Adaptive Reactive Controllers”. In:

Sauvola, J. and M. Pietikäinen (2000). “Adaptive document image binarization”. In: *Pattern Recognition* 33.2, pp. 225 –236.

Scharstein, Daniel (1999). *View Synthesis Using Stereo Vision*. Springer Berlin Heidelberg.

Scholl, Brian J (2001). “Objects and attention: the state of the art”. In: *Cognition* 80.1âĂŞ2, pp. 1 –46.

Sermanet, Pierre et al. (2013). “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks”. In: *CoRR*.

Siegwart, Roland and Illah Nourbakhsh (2004). *Autonomous Mobile Robots*. Massachusetts Institute of Technology.

Simons, Daniel J and Christopher F Chabris (1999). “Gorillas in Our Midst: Sustained Inattentional Blindness for Dynamic Events”. In: *Perception* 28.9, pp. 1059–1074.

Simonyan, Karen and Andrew Zisserman (2014). “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *CoRR* abs/1409.1556.

Smeets, Jeroen B.J. and Eli Brenner (1994). “The difference between the perception of absolute and relative motion: a reaction time study”. In: *Vision Research* 34.2, pp. 191 –195.

Suárez, Mark E. and Barbara L. Thorne (2000). “Rate, Amount, and Distribution Pattern of Alimentary Fluid Transfer via Trophallaxis in Three Species of Termites (Isoptera: Rhinotermitidae, Termopsidae)”. In: *Annals of the Entomological Society of America* 93.1, p. 145.

Sumpter, D. J. T. (2006). “The Principles of Collective Animal Behaviour”. In: *Philosophical Transactions: Biological Sciences* 361.1465, pp. 5–22.

Sutton, Richard S. and Andrew G. Barto (1998). *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts: MIT Press.

Tanaka, James, Daniel Weiskopf, and Pepper Williams (2001). “The role of color in high-level vision”. In: *Trends in Cognitive Sciences* 5.5, pp. 211 –215.

Consequential Robotics Ltd, (2016). *Technical Specifications*. [Online: accessed 14-March-2017]. URL: `http://consequentialrobotics.com/technical-specifications/`.

Tigli, J. Y. et al. (1993). “Methodology and computing model for a reactive mobile robot controller”. In: *Proceedings of IEEE Systems Man and Cybernetics Conference - SMC*. Vol. 2, pp. 317–322.

Trianni, Vito and Marco Dorigo (2006). “Self-organisation and communication in groups of simulated and physical robots”. In: *Biological Cybernetics* 95.3, pp. 213–231.

Tversky, Barbara and Kathleen Hemenway (1984). “Objects, Parts, and Categories”. In: *Journal of Experimental Psychology: General* 113.2, pp. 169–193.

Uchida, Yusuke (2016). “Local Feature Detectors, Descriptors, and Image Representations: A Survey”. In: *CoRR*. University of Tokyo.

Viola, P. and M. Jones (2001). “Rapid object detection using a boosted cascade of simple features”. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Vol. 1, pp. 511–518.

Wada, K. et al. (2003). “Psychological and social effects of robot assisted activity to elderly people who stay at a health service facility for the aged”. In: *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 3996–4001.

Wahde, Mattias (2016). *Introduction to Autonomous Robots*. Chalmers University of Technology. Chap. Kinematics and Dynamics, pp. 21–28.

Wang, B. R. et al. (2014). “Design of Jitter Compensation Algorithm for Robot Vision Based on Optical Flow and Kalman Filter”. In: *The Scientific World Journal* 130806.

Wang, Zhenshi and Xuezhe Wei (2015). “Design Considerations for Wireless Charging Systems with an Analysis of Batteries”. In: *Energies* 8.10, pp. 10664–10683.

Watson, Andrew B. and John I. Yellott (2012). “A unified formula for light-adapted pupil size”. In: *Journal of Vision* 12.10, p. 12.

Wei, Shui gen et al. (2011). “Motion Detection Based on Optical Flow and Self-adaptive Threshold Segmentation”. In: *Procedia Engineering* 15, pp. 3471 –3476.

Weidner, Ralph et al. (2014). “The moon illusion and size-distance scaling–evidence for shared neural patterns”. In: *Journal of cognitive neuroscience* 26.8, pp. 1871–1882.

Weszka, Joan S. (1977). “A Survey of Threshold Selection Techniques”. In: *Computer Graphics and Image Processing* 7, pp. 259–265.

Wit, C. Canudas de and O. J. Sordalen (1993). “Exponential Stabilization of Mobile Robots with Nonholonomic Constraints”. In: *IEEE Transactions on Robotics and Automation* 37, pp. 1791–1797.

Wit, Lee H. de et al. (2011). “Investigating the Status of Biological Stimuli as Objects of Attention in Multiple Object Tracking”. In: *PLOS ONE* 6.3, pp. 1–8.

Wolfe, Jeremy M. (2010). “Visual search”. In: *Current Biology* 20.8, R346 –R349.

Wood, Justin N. (2014). “Newly Hatched Chicks Solve the Visual Binding Problem”. In: *Psychological Science* 25.7, pp. 1475–1481.

Wood, Samantha M. W. and Justin N. Wood (2015). “A chicken model for studying the emergence of invariant object recognition”. In: *Frontiers in Neural Circuits* 9.7.

Xu, Gang and Zhengyou Zhang (1996). *Epipolar Geometry in Stereo, Motion and Object Recognition: A Unified Approach*. Springer Science & Business Media.

Yahya, A. et al. (2016). “Collective Robot Reinforcement Learning with Distributed Asynchronous Guided Policy Search”. In: *ArXiv e-prints*. arXiv: `1610.00673 [cs.LG]`.

Zhu, Anmin et al. (2005). “A Neuro-fuzzy Controller for Reactive Navigation of a Behaviour-Based Mobile Robot”. In: *Advances in Neural Networks – ISNN 2005: Second International Symposium on Neural Networks, Chongqing, China, May 30 - June 1, 2005, Proceedings, Part III*. Ed. by Jun Wang, Xiao-Feng Liao, and Zhang Yi. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 259–264.

Appendices

A.1 The Perceptron Algorithm – Python Source Code

The below script is divided in to 3 main functions. The first, named image_noise_histograms, takes a greyscale image, adds Gaussian salt-and-pepper noise to the image, and returns a histogram vector according to the number of bins set (values range from 0-255). The second, named perceptron_ANN_train, trains the perceptron algorithm according to Section 4.2.3. The third, named perceptron_ANN_test, then tests the network on validation data. Finally, below the functions lies the script that retrieves the images for training, calls the histogram and perceptron training functions, and finally retrieves the number of correctly classified images.

```python
import numpy as np
from matplotlib import pyplot as plt
import cv2
import random
import os

# function for displaying an image
def imshow(img):
    cv2.imshow('Image', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def image_noise_histograms(img, std_dev, bins):
# function for adding Gaussian noise to an image and returning its
    greyscale histogram

    ### extracting the histograms for all positive images

    # add Gaussian noise to the image
    noise = cv2.randn(np.zeros((img.shape),np.int8),0,std_dev)
    img_noise = img + noise
    img_noise = np.clip(img_noise,0,255)
    img_noise = np.uint8(img_noise)

    # calculate the normalised histogram vector for the image
    hist = cv2.calcHist([img_noise], [0], None, [bins], [0,256])
    hist_norm = hist / sum(hist)

    return hist_norm

def perceptron_ANN_train(training_pos_hists, training_neg_hists, w, eta, beta):
# Training the perceptron weights given positive and negative training sets, plus initial weights

    training_pos = training_pos_hists
    training_neg = training_neg_hists

    # assign the positive and negative images with the correct classifier
    train = np.zeros((len(training_pos), 2))
    train_pos = train.astype(np.object)
    for h in range(len(training_pos)):
        train_pos[h,0] = training_pos[h]
        train_pos[h,1] = 1

    train = np.zeros((len(training_neg), 2))
    train_neg = train.astype(np.object)
    for h in range(len(training_neg)):
        train_neg[h,0] = training_neg[h]
        train_neg[h,1] = 0

    #### train the network
    noIter = 100000
    i = 0
    e = np.zeros(noIter+1)
    e[0] = 0.5
    alpha = 0.0001
    while i < noIter:
        # take an image at random
        pos_or_neg = np.round(np.random.rand())
        if pos_or_neg:
            index = np.random.randint(len(training_pos))
            img = train_pos[index]
        else:
            index = np.random.randint(len(training_neg))
            img = train_neg[index]

        # compute the output as a simple linear sum of inputs x weights
      + bias
        y = np.dot(np.transpose(img[0]),w) #+ beta

        # update the error rate with exponential smoothing
        if (y < beta) & (img[1] == 0):
            H = 1
        elif (y > beta) & (img[1] == 1):
            H = 1
        elif (y < beta) & (img[1] == 1):
            H = 0
        else:
            H = 0
        e[i+1] = alpha*H + (1 - alpha)*e[i]

    # update the weights
    w = w + eta*(img[1] - y)*img[0]

    i += 1

    return w, e

def perceptron_ANN_test(validate_pos, validate_neg, w, beta):
### test the net on the validation data
    outputs = []
    # assign the positive and negative images with the correct
      classifier
    train = np.zeros((len(validate_pos), 2))
    train_pos = train.astype(np.object)
    for h in range(len(validate_pos)):
        train_pos[h,0] = validate_pos[h]
        train_pos[h,1] = 1

    train = np.zeros((len(validate_neg), 2))
    train_neg = train.astype(np.object)
    for h in range(len(validate_neg)):
        train_neg[h,0] = validate_neg[h]
        train_neg[h,1] = 0
    correct = 0
    false_neg = 0
    false_pos = 0
    for i in range(len(validate_pos)):
        img = train_pos[i]

        # compute the output as a simple linear sum of inputs x weights
      + bias
        y = np.dot(np.transpose(img[0]),w)

        if (y < beta) & (img[1] == 0):
        correct += 1
        elif (y > beta) & (img[1] == 1):
        correct += 1
        elif (y < beta) & (img[1] == 1):
        false_neg += 1
        else:
        false_pos += 1
        i += 1
        outputs.append(y)

    for i in range(len(validate_neg)):
        img = train_neg[i]

        # compute the output as a simple linear sum of inputs x weights
      + bias
        y = np.dot(np.transpose(img[0]),w)

        if (y < beta) & (img[1] == 0):
```

```
        correct += 1
        elif (y > beta) & (img[1] == 1):
        correct += 1
        elif (y < beta) & (img[1] == 1):
        false_neg += 1
        else:
        false_pos += 1
        i += 1
        outputs.append(y)
    proportion_correct = float(correct)/(len(validate_pos) + (len(
    validate_neg)))

    return proportion_correct, correct, false_neg, false_pos, outputs

# the directory for the positive and negative images
positive_images_dir = 'training_images/positive_images'
negative_images_dir = 'training_images/negative_images'

# constants and lists
eta = 10 #learning rate
beta = 0.5 #decision boundary
bins = 256 #number of bins for histograms
hists_pos = []
hists_neg = []
correct = []
false_negs = []
false_poss = []
error = []

# begin storing the image data
print 'Storing␣image␣data...'
imgs_pos = []
imgs_neg = []
for filename in os.listdir(positive_images_dir):
    img = cv2.imread(os.path.join(positive_images_dir,filename))
    if img is not None:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        imgs_pos.append(img)
for filename in os.listdir(negative_images_dir):
    img = cv2.imread(os.path.join(negative_images_dir,filename))
    if img is not None:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        imgs_neg.append(img)
print 'Completed␣collecting␣image␣data.'

print 'Starting␣the␣training␣and␣validating...'
# start the training and validating
for Round in range(100):
    for i in range(len(imgs_pos)):
        hist = image_noise_histograms(imgs_pos[i], std_dev=10, bins=256)
        hists_pos.append(hist)
    for i in range(len(imgs_neg)):
        hist = image_noise_histograms(imgs_neg[i], std_dev=10, bins=256)
        hists_neg.append(hist)

    # extract the validation sets at random and delete them from the training sets
    val_index_pos = random.sample(range(len(hists_pos)), 4)
    val_index_neg = random.sample(range(len(hists_neg)), 4)
    validate_pos = [hists_pos[i] for i in val_index_pos]
    validate_neg = [hists_neg[i] for i in val_index_neg]
    for index in sorted(val_index_pos, reverse=True):
        del hists_pos[index]
    for index in sorted(val_index_neg, reverse=True):
        del hists_neg[index]

    # train the perceptron and get the network weights
    # initialise the weights and normalise
    w = np.random.rand((bins))
    w = (w / np.linalg.norm(w)).reshape((bins,1))
    w, e = perceptron_ANN_train(hists_pos, hists_neg, w, eta, beta)
    error.append(e)
    # test the perceptron
    proportion_correct, no_correct, false_neg, false_pos, outputs = perceptron_ANN_test(validate_pos, validate_neg, w, beta)
    correct.append(proportion_correct)
    false_negs.append(false_neg)
    false_poss.append(false_pos)
    print 'Round ',Round+1, ' of training and validating complete!'
#print outputs
print 'Proportion correct = ', correct
print 'Number of false-negatives = ', false_negs
print 'Number of false-positives = ', false_poss
```

A.2 Computing the value of w in the approximated Hessian determinant

Recall that the formula for computing the weight w was given by,

$$w \approx \frac{||L_{xy}(\sigma)||_{\mathrm{F}}\,||D_{xx}(n)||_{\mathrm{F}}}{||L_{xx}(\sigma)||_{\mathrm{F}}\,||D_{xy}(n)||_{\mathrm{F}}} \tag{A.1}$$

where n is the size of the box filter used in the approximation of the second order Gaussian, and $||\cdot||_{\mathrm{F}}$ is the Frobenius norm operator.

The Frobenius norm can be calculated by taking the square root of the Frobenius inner product, say for instance matrix $\mathbf{A}$,

$$||\mathbf{A}||_{\mathrm{F}} = \sqrt{\langle \mathbf{A}, \mathbf{A} \rangle_{\mathrm{F}}} \tag{A.2}$$

where the Frobenius inner product is given by,

$$\langle \mathbf{A}, \mathbf{A} \rangle_{\mathrm{F}} = \mathtt{tr}\left(\mathbf{A}^T \mathbf{A}\right) \tag{A.3}$$

We now define the matrices D_{xx} and D_{xy} by converting Figure 4.14 into matrix form,

$$D_{xx} = \begin{bmatrix} 0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0\\ 1&1&1&-2&-2&-2&1&1&1\\ 1&1&1&-2&-2&-2&1&1&1\\ 1&1&1&-2&-2&-2&1&1&1\\ 1&1&1&-2&-2&-2&1&1&1\\ 1&1&1&-2&-2&-2&1&1&1\\ 0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0 \end{bmatrix}, \quad D_{xy} = \begin{bmatrix} 0&0&0&0&0&0&0&0&0\\ 0&1&1&1&0&-1&-1&-1&0\\ 0&1&1&1&0&-1&-1&-1&0\\ 0&1&1&1&0&-1&-1&-1&0\\ 0&0&0&0&0&0&0&0&0\\ 0&-1&-1&-1&0&1&1&1&0\\ 0&-1&-1&-1&0&1&1&1&0\\ 0&-1&-1&-1&0&1&1&1&0\\ 0&0&0&0&0&0&0&0&0 \end{bmatrix} \tag{A.4}$$

Applying (A.2) to (A.4) gives,

$$\begin{aligned} ||D_{xx}||_{\mathrm{F}} &= \sqrt{\langle D_{xx}, D_{xx} \rangle_{\mathrm{F}}} = 9.4868 \\ ||D_{xy}||_{\mathrm{F}} &= \sqrt{\langle D_{xy}, D_{xy} \rangle_{\mathrm{F}}} = 6.0 \end{aligned} \tag{A.5}$$

We next discretise a 9x9 box with the centre space having location $(x_{centre}, y_{centre}) = (0,0)$, and ranging from $(-4,-4)$ to $(4,4)$, as in (A.6)

$$\begin{bmatrix}
(-4,4) & (-3,4) & (-2,4) & (-1,4) & (0,4) & (1,4) & (2,4) & (3,4) & (4,4) \\
(-4,3) & (-3,3) & (-2,3) & (-1,3) & (0,3) & (1,3) & (2,3) & (3,3) & (4,3) \\
(-4,2) & (-3,2) & (-2,2) & (-1,2) & (0,2) & (1,2) & (2,2) & (3,2) & (4,2) \\
(-4,1) & (-3,1) & (-2,1) & (-1,1) & (0,1) & (1,1) & (2,1) & (3,1) & (4,1) \\
(-4,0) & (-3,0) & (-2,0) & (-1,0) & (0,0) & (1,0) & (2,0) & (3,0) & (4,0) \\
(-4,-1) & (-3,-1) & (-2,-1) & (-1,-1) & (0,-1) & (1,-1) & (2,-1) & (3,-1) & (4,-1) \\
(-4,-2) & (-3,-2) & (-2,-2) & (-1,-2) & (0,-2) & (1,-2) & (2,-2) & (3,-2) & (4,-2) \\
(-4,-3) & (-3,-3) & (-2,-3) & (-1,-3) & (0,-3) & (1,-3) & (2,-3) & (3,-3) & (4,-3) \\
(-4,-4) & (-3,-4) & (-2,-4) & (-1,-4) & (0,-4) & (1,-4) & (2,-4) & (3,-4) & (4,-4)
\end{bmatrix} \tag{A.6}$$

Take next the Gaussian function from Equation 4.10, for which the second order derivatives are given by,

$$\begin{aligned}
\frac{\partial^2}{\partial x^2} G(\sigma) &= \frac{x^2 - \sigma^2}{2\pi\sigma^6} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \\
\frac{\partial^2}{\partial x \partial y} G(\sigma) &= \frac{xy}{2\pi\sigma^6} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)
\end{aligned} \tag{A.7}$$

and applying (A.7) to the (x,y) positions in (A.6) gives,

$$L_{xx} = \begin{bmatrix}
0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\
0.000 & 0.001 & 0.001 & -0.001 & -0.003 & -0.001 & 0.001 & 0.001 & 0.000 \\
0.001 & 0.004 & 0.008 & -0.004 & -0.019 & -0.004 & 0.008 & 0.004 & 0.001 \\
0.002 & 0.013 & 0.024 & -0.012 & -0.054 & -0.012 & 0.024 & 0.013 & 0.002 \\
0.003 & 0.018 & 0.034 & -0.017 & -0.077 & -0.017 & 0.034 & 0.018 & 0.003 \\
0.002 & 0.013 & 0.024 & -0.012 & -0.054 & -0.012 & 0.024 & 0.013 & 0.002 \\
0.001 & 0.004 & 0.008 & -0.004 & -0.019 & -0.004 & 0.008 & 0.004 & 0.001 \\
0.000 & 0.001 & 0.001 & -0.001 & -0.003 & -0.001 & 0.001 & 0.001 & 0.000 \\
0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000
\end{bmatrix},$$

$$L_{xy} = \begin{bmatrix} 0.000 & 0.000 & 0.000 & 0.001 & 0.000 & -0.001 & 0.000 & 0.000 & 0.000 \\ 0.000 & 0.001 & 0.004 & 0.005 & 0.000 & -0.005 & -0.004 & -0.001 & 0.000 \\ 0.000 & 0.004 & 0.013 & 0.019 & 0.000 & -0.019 & -0.013 & -0.004 & 0.000 \\ 0.001 & 0.005 & 0.019 & 0.027 & 0.000 & -0.027 & -0.019 & -0.005 & -0.001 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ -0.001 & -0.005 & -0.019 & -0.027 & 0.000 & 0.027 & 0.019 & 0.005 & 0.001 \\ 0.000 & -0.004 & -0.013 & -0.019 & 0.000 & 0.019 & 0.013 & 0.004 & 0.000 \\ 0.000 & -0.001 & -0.004 & -0.005 & 0.000 & 0.005 & 0.004 & 0.001 & 0.000 \\ 0.000 & 0.000 & 0.000 & -0.001 & 0.000 & 0.001 & 0.000 & 0.000 & 0.000 \end{bmatrix} \tag{A.8}$$

where I in the convolution is equal unity.

Again, applying (A.2) to (A.8) gives,

$$\begin{aligned} ||L_{xx}||_{\mathrm{F}} &= \sqrt{\langle L_{xx}, L_{xx}\rangle_{\mathrm{F}}} = 0.1414 \\ ||L_{xy}||_{\mathrm{F}} &= \sqrt{\langle L_{xy}, L_{xy}\rangle_{\mathrm{F}}} = 0.0816 \end{aligned} \tag{A.9}$$

to which the weight w can now be computed,

$$w \approx \frac{||L_{xy}(\sigma)||_{\mathrm{F}}\ ||D_{xx}(n)||_{\mathrm{F}}}{||L_{xx}(\sigma)||_{\mathrm{F}}\ ||D_{xy}(n)||_{\mathrm{F}}} = \frac{0.0816 \times 9.4868}{0.1414 \times 6.0} = 0.9125.. \approx 0.9 \tag{A.10}$$

A.3 SURF with Bayesian Estimation – Python Source Code

The following Python code extracts interest points and their descriptors by implementing the SURF algorithm using OpenCV's SURF module. It then proceeds to compute the posterior probability of an image being a MiRo given the feature vectors, according to the procedure laid out in Section 4.3.3.

```
import cv2
import os
import numpy as np
import random

surf = cv2.SURF(400,extended=False)

def imshow(img):
    cv2.imshow('image',img)
    cv2.waitKey(0)
    cv2.destroyAllWindows

positive_images_dir = 'C:/Users/Matt Work/Dropbox/MSc Individual
    Project/ANN_training/training_images/positive_images/'
negative_images_dir = 'C:/Users/Matt Work/Dropbox/MSc Individual
    Project/ANN_training/training_images/negative_images/'
# begin storing the image data
print 'Storing image data...'
imgs_pos = []
imgs_neg = []
for filename in os.listdir(positive_images_dir):
    img = cv2.imread(os.path.join(positive_images_dir,filename))
    if img is not None:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        imgs_pos.append(img)
for filename in os.listdir(negative_images_dir):
    img = cv2.imread(os.path.join(negative_images_dir,filename))
    if img is not None:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        imgs_neg.append(img)
```

```
print 'Completed collecting image data.'

# for each negative image, compute the surf feature vectors
f_descriptors_neg = []
for i in range(len(imgs_neg)):
    k,des = surf.detectAndCompute(imgs_neg[i],None)
    f_descriptors_neg.append(des)

runs = 48
probabilities = []
for run in range(runs):
    # for each positive image, compute the surf feature vectors
    f_descriptors = []
    t = []
    for i in range(len(imgs_pos)):
        k,des = surf.detectAndCompute(imgs_pos[i],None)
        f_descriptors.append(des)
    # extract the positive validation images
    val_index_pos = random.sample(range(len(imgs_pos)), 3)
    val_index_pos = random.randint(0,47)
    val_index_pos = run
    if isinstance(val_index_pos, int):
        f_descriptor_validate = f_descriptors[val_index_pos]
        del f_descriptors[val_index_pos]
    else:
        f_descriptors_validate = [f_descriptors[i] for i in
     val_index_pos]
        for index in sorted(val_index_pos, reverse=True):
            del f_descriptors[index]

#    # for using with tests on negative images
#    index_neg = random.randint(0,700)
#    index_neg = run
#    f_descriptor_validate = f_descriptors_neg[index_neg]

    if f_descriptor_validate is None:
        continue

    # finding the likelihoods for the image and storing into two lists
```

```
probs_MiRo = []
probs_H0 = []
thresh = 0.46
features = f_descriptor_validate # extract the feature descriptors for the image
for f in range(len(features)): # run through each feature descriptor in the image
  # computing first p(v|MiRo)
  count_MiRo = 0
  for img_train in range(len(f_descriptors)): # run through each of the positive training images
    features_train = f_descriptors[img_train] # extract the feature descriptors for the image
    for f_train in range(len(features_train)): #run through each of the features in each training image
      d = np.sqrt(np.dot(np.transpose((features[f,:]) - features_train[f_train,:]),(features[f,:]) - features_train[f_train,:]))
      if d < thresh:
      count_MiRo += 1
      break
  # computing now p(v|H0)
  count_H0 = 0
  negs_removed = 0 #some negative images have no feature descriptors, and need removing from the probability estimation
  for img_train in range(len(f_descriptors_neg)): # run through each of the negative training images
    features_train = f_descriptors_neg[img_train] # extract the feature descriptors for the image
    if not features_train is None:
      if len(features_train) < 50:
        for f_train in range(len(features_train)): #run through each of the features in each training image
        d = np.sqrt(np.dot(np.transpose((features[f,:]) - features_train[f_train,:]),(features[f,:]) - features_train[f_train,:]))
        if d < thresh:
          count_H0 += 1
          break
```

```
                else:
                    for f_train in range(50): #run through each of the features in each training image
                        d = np.sqrt(np.dot(np.transpose((features[f,:]) - features_train[f_train,:]),(features[f,:]) - features_train[f_train,:]))
                        if d < thresh:
                            count_H0 += 1
                            break
            else:
                negs_removed += 1
        p_v_given_MiRo = float(count_MiRo) / float(len(f_descriptors)) + 0.001
        p_v_given_H0 = float(count_H0) / float(len(f_descriptors_neg) - negs_removed) + 0.001
        probs_MiRo.append(p_v_given_MiRo)
        probs_H0.append(p_v_given_H0)
        if f > 20: # only estimate for the first 20 features
            break
    p_MiRo = 0.25
    p_V_given_MiRo = np.prod(np.array(probs_MiRo))
    p_V_given_H0 = np.prod(np.array(probs_H0))
    p_MiRo_given_V = (p_V_given_MiRo*p_MiRo) / (p_V_given_H0*(1-p_MiRo) + p_V_given_MiRo*p_MiRo)
    probabilities.append(p_MiRo_given_V)
```

A.4 Vision Algorithm Training Images

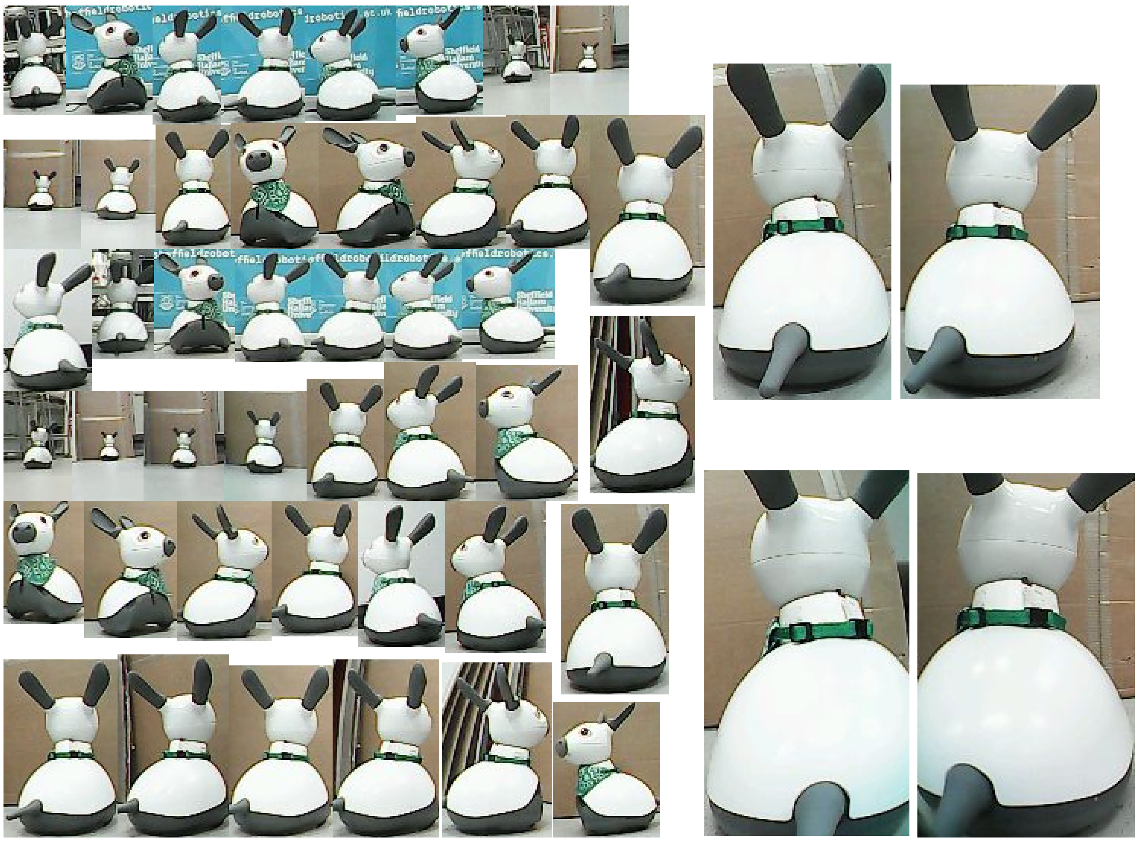


Figure A.4: All MiRo positive training images used in the vision algorithms training and validation procedures. Notice the variation in background, orientation and scale. Refer to Chapter 4 for the description of the vision algorithms.

A.5 MiRo Controller – Python Source Code

Python code changes to the functions callback_platform_sensors, imdecode, callback_caml and callback_camr for implementing the initial controller. The above functions are found in the miro_ros_client.py script in the MiRo Developer Kit.

```
def callback_platform_sensors(self, object):

    # ignore until active
    if not self.active:
        return

    # store object
    self.platform_sensors = object

    # timing
    sync_rate = 50
    period = 2 * sync_rate # two seconds per period
    z = self.count / period
    self.z_bak = z

    q = platform_control()
    max_speed = 200

    # get the thresholded values for each image
    areaL = 0
    areaR = 0
    try:
        areaL = self.area_l
    except NameError:
        pass
    try:
        areaR = self.area_r
    except NameError:
        pass

    c_posL = self.c_pos_l
    c_posR = self.c_pos_r

```

```python
    if (areaL and areaR) or areaL:
        x_l = c_posL[0] - 160
        x_r = c_posR[0] - 160
        x = x_l + x_r
        y_l = c_posL[1] - 120
        y_r = c_posR[1] - 120
        y = (y_l + y_r) / 2 #y is average of two images
    elif areaR:
        x_l = c_posL[0] + 160
        x_r = c_posR[0] - 160
        x = x_l + x_r
        y_l = c_posL[1] + 120
        y_r = c_posR[1] - 120
        y = (y_l + y_r) / 2 #y is average of two images
    else:
        x = None
        y = None

    wheel_speed_r = 0
    wheel_speed_l = 0

    # specifiy wheel speeds in mm/s
    if not x is None:
        if x == 0:
            wheel_speed_r = max_speed
            wheel_speed_l = max_speed
        elif x < 0:
            wheel_speed_r = max_speed
            wheel_speed_l = numpy.exp(0.004*x) * max_speed
        else:
            wheel_speed_r = numpy.exp(-0.004*x) * max_speed
            wheel_speed_l = max_speed

        # use the area to compute the modified logistic function
        area_avg = (areaL + areaR) / 2
        H = 1.0 / (1.0 + numpy.exp(0.0008*(area_avg - 10000.0)))
        wheel_speed_l = wheel_speed_l * H
        wheel_speed_r = wheel_speed_r * H

    # send downstream command, ignoring upstream data
    q.body_vel.angular.z = self.wheel_speed_2_body_vel_angular(wheel_speed_l, wheel_speed_r)
    q.body_vel.linear.x = self.wheel_speed_2_body_vel_linear(wheel_speed_l, wheel_speed_r)

    # publish
    self.pub_robot.publish(q)

    # count
    self.count = self.count + 1
    if self.count == 400:
    self.count = 0

def imdecode(self, fifo, frm, overlayable = False):
    im = cv2.imdecode(numpy.fromstring(frm.data, numpy.uint8),cv2.IMREAD_COLOR)
    w = im.shape[1]
    h = im.shape[0]
    N = h * w
    for i in range(0, N):
        tmp = im.data[i*3+0]
        im.data[i*3+0] = im.data[i*3+2]
        im.data[i*3+2] = tmp
    pb = GdkPixbuf.Pixbuf.new_from_data(im.data,GdkPixbuf.Colorspace.RGB,False, 8, w, h, w*3)
    # thresholding
    lower = numpy.array([0,0,100]) #defining RGB thresholds
    upper = numpy.array([10,10,255])
    mask = cv2.inRange(im, lower, upper)
    M = cv2.moments(mask)
    area = int(M['m00']/255) # number of pixels successfully filtered
    if area: # only if miro is detected
        cx = int(M['m10']/M['m00'])
        cy = int(M['m01']/M['m00'])
        c_pos = ((cx,cy)) # centroid position
    else:
        c_pos = ((0,0))

    fifo.push(pb)

    return(area, c_pos)

def callback_caml(self, frm):

    # ignore until active
    if not self.active:
        return

    # store object
    self.area_l, self.c_pos_l = self.imdecode(self.caml_fifo, frm,
    True)

def callback_camr(self, frm):

    # ignore until active
    if not self.active:
        return

    # store object
    self.area_r, self.c_pos_r = self.imdecode(self.camr_fifo, frm,
    True)
```

A.6 A Proposal for an Adaptive Controller Strategy

Section 5.2.4 mentioned the inability of the controller to adapt for MiRo's unbalanced wheel speeds. Here is proposed a potential solution to the problem, where its implementation is left open for further consideration.

The aim for this adaptive controller is to have MiRo adapt its parameters according to its behaviour outputs, thus overcoming many of the uncertainties that give rise to the issue outlined above. In addition to the centroid, c_{est}, the rate of change of the centroid, $\dot{c}_{est}$, is required in this controller.

The range of centroid values determined by Equation 5.15 are presented as 641 discreet states (from -320 to 320), which then act as the inputs to a single layered neural network. An additional set of inputs are included for the centroids rate of change, which should be another 641 states ranging from -320 to 320 (c/fps). The total number of inputs to the neural network is therefore 1282.

An input that represents a centroid position shall be given as x^c, and for a centroid rate of change x^{cc} shall represent it. Input x_i^c is 1 if the current state is equal to that input, or 0 otherwise, and similarly for x_j^{cc}.

The left and right wheel speeds are then given as a ratio of one another,

$$s_{ratio} = \frac{s_l}{s_r} \tag{A.11}$$

ranging from $s_{ratio} = \frac{0.25}{1}$ to 1, in steps of 0.05 (0.25, 0.30, ... , 1) and then descending down again but with the inverse values ($\frac{1}{0.95}$, $\frac{1}{0.90}$, ... , $\frac{1}{0.25}$) giving 31 discreet bins. These act as the outputs to the neural network. Given that there will be two inputs active at any given time, representing the centroid and centroid rate of change, the outputs are computed as a simple linear sum as follows,

$$y_k = w_{ik}x_i^c + w_{jk}x_j^{cc} \tag{A.12}$$

An output, y_k, is 1 if it has the maximum output vs all other outputs, and 0 otherwise, i.e.

$$y_k = \begin{cases} 1, & \text{if } y_k > y_l \text{ for all } l \neq k \\ 0, & \text{otherwise} \end{cases} \tag{A.13}$$

Reinforcement learning is then performed using the Rescorla-Wagner update rule (Rescorla and Wagner, 1972) which shares the same characteristics as the perceptron rule used in section 4.2.3, i.e. that a weight change occurs according to,

$$w_{ik} \to w_{ik} + \eta \delta x_k \tag{A.14}$$

where in this instance δ is the difference between the output and the reward,

$$\delta = r - y_i \tag{A.15}$$

Now it is well understood in temporal difference learning that rewards can be predicted for some future time instance (see Q-learning and the SARSA algorithm (Sutton and Barto, 1998)), but here the approach will be to offer a reward at every time instance, and to offer two rewards – one as a function of the centroid position; the second as a function of the centroid rate of change, (letting $c_x^{est} = c$),

$$r(c) = 1 - \frac{|c|}{c_{max}} \tag{A.16}$$

$$r(\dot{c}) = 1 - \frac{|\dot{c}|}{\dot{c}_{max}} \tag{A.17}$$

where $c_{max} = 320$ and $\dot{c}_{max} = 640$ and as such $\frac{1}{c_{max}}$ and $\frac{1}{\dot{c}_{max}}$ act as normalising terms, ensuring $|r| \leq 1$ in each case. Then, the above will give decreased rewards for being further away from $c = 0$ or for large values of $\dot{c}$, and will give rewards of 1 when $c = \dot{c} = 0$, and therefore represents the ideal states.

In order to compute Equations A.14 and A.15, it is imperative that the correct reward is used for the correct state. I.e., for a c state (x^c), r is given by (A.16), whilst for a $\dot{c}$ state (x^{cc}), (A.17) is used.

A state-action pair is thus given greater rewards for actions that reduce the absolute value of c and $\dot{c}$. There is one noticeable problem with the controller, and is found in Equation A.13. Choosing always the action with the largest output could prevent the controller from finding optimal solutions. To prevent this, a policy such as *softmax* could be implemented (see (Sutton and Barto, 1998, p.30-31)) to allow possibilities for further exploration of actions, before eventually converging to greedy as in (A.13).

A.7 Project Timing Plan

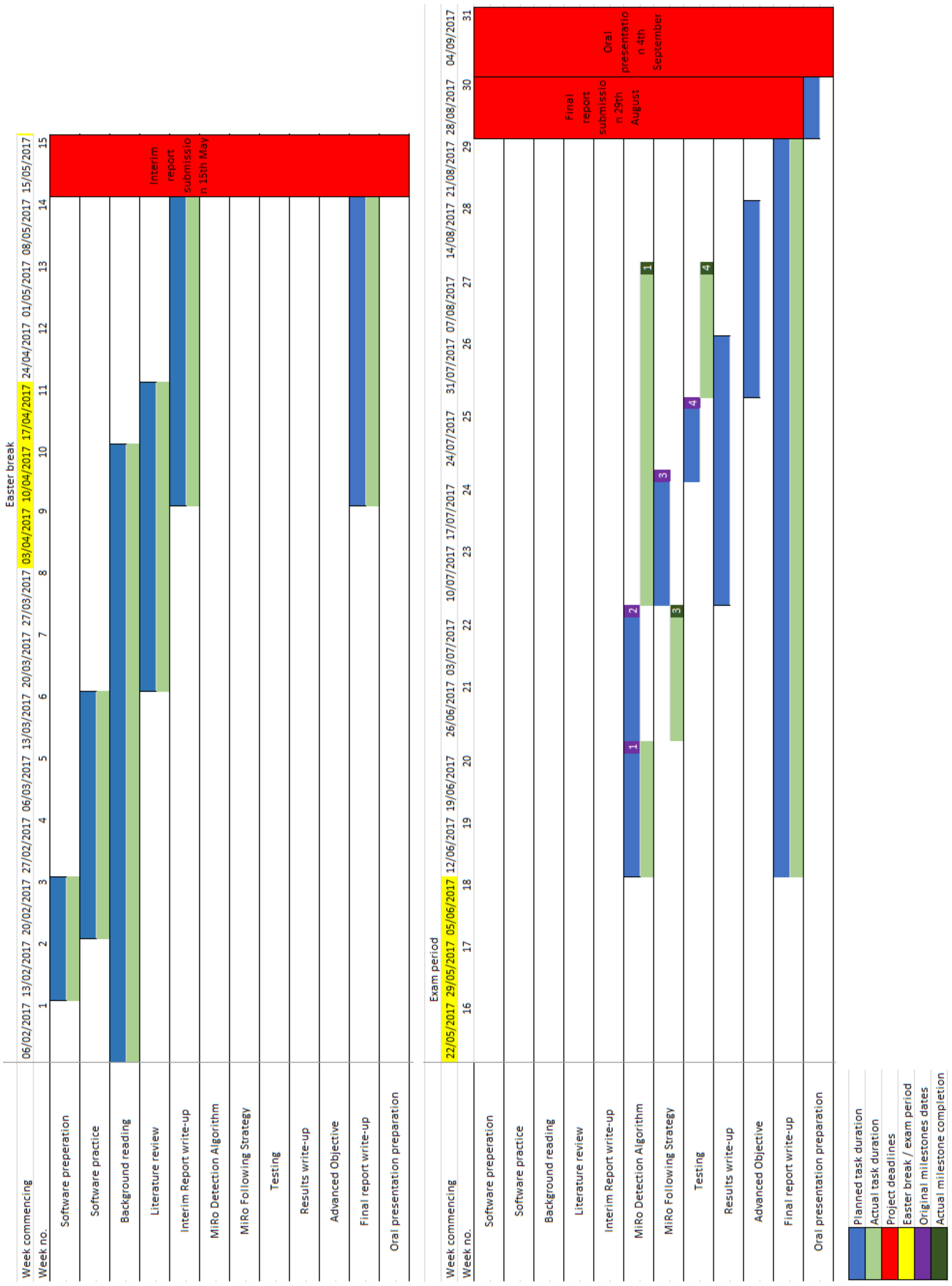


Figure A.7: The project's Gantt chart. In blue are the original project timing plans, with the green bars the actual progress. Although the project did not strictly adhere to the original timing plan, this did not affect the overall completion of the project aim and objectives.