Department of Electronic Engineering

MEng Project Report

2016/17

Student Name: Alistair Jewers

Project Title: Augmented Reality Debugging System for Robot Swarms

Supervisors: Dr. Alan Millard, Dr. Shuhei Miyashita

Department of Electronic Engineering
University of York
Heslington
York
Y010 5DD

UNIVERSITY OF YORK

MASTERS THESIS

Augmented Reality Debugging System for Robot Swarms

Author:
Alistair JEWERS

Supervisor:
Dr. Alan MILLARD

A thesis submitted in fulfilment of the requirements for the degree of Master of Engineering

in the

Department of Electronic Engineering

May 17, 2017

Declaration of Authorship

I, Alistair JEWERS, declare that this thesis titled, "Augmented Reality Debugging System for Robot Swarms" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for an undergraduate degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Alistair Jewers

May 17, 2017

Abstract

Debugging presents a challenge when developing swarm robotic systems, due to difficulties in retrieving internal information from multiple robots, and a lack of appropriate tools. This project addressed this challenge by creating a system for monitoring a robot swarm, and visualising internal data using graphical, augmented-reality techniques. This report describes the design, implementation and testing of the system, which comprises a software application, robot-side API and data transfer format. The system was evaluated by a number of potential users in a test scenario, and was deemed likely to be useful in a real debugging situation. The report concludes that this project was successful in creating a swarm robotics debugging tool, providing a foundation for a broader swarm robotics research and development platform.

Acknowledgements

With thanks to my supervisor, Alan Millard, without whose help this would not have been possible, and to James Hilder, without whom the technical difficulties alone would have been insurmountable.

Contents

Declaration of Authorship **i**

Abstract **ii**

Acknowledgements **iii**

1 Introduction **1**
1.1 Background . . . 1
1.2 Project Context . . . 2
1.3 Project Concept . . . 4
1.4 Aim and Objectives . . . 5
1.5 Functional Specification . . . 6
1.6 Report Structure . . . 7
1.7 York Robotics Laboratory . . . 8
1.8 Ethics . . . 8

2 Literature Review **10**
2.1 Overview . . . 10
2.2 Swarm Intelligence and Swarm Robotics . . . 10
2.3 Human Swarm Interaction . . . 13
2.4 Robotics Debugging . . . 17
2.5 Augmented Reality and Single-Robot Systems . . . 20
2.6 Augmented Reality and Multiple-Robot Systems . . . 23
2.7 Summary . . . 25

3 Problem Analysis **27**
3.1 Initial User Survey . . . 27
3.1.1 Questions and Response Data . . . 27
3.1.2 Analysis . . . 30
3.1.3 Issues and Shortcomings . . . 31
3.2 Hardware - E-Puck Robot Platform . . . 31
3.2.1 Actuators . . . 32
3.2.2 Sensors . . . 32
3.2.3 Processors and Code Architecture . . . 33
3.3 Hardware - Video Tracking System . . . 35
3.3.1 Camera . . . 35

3.3.2 ArUco Tracking System . . . 37
3.4 Software Development Tools . . . 39
3.4.1 *Git* Version Control System . . . 39
3.4.2 *Qt Creator* Integrated Development Environment . . . 41
3.5 Summary . . . 42

4 Project Plan **43**
4.1 Work Breakdown . . . 43
4.2 Timing and Plan . . . 46
4.3 Risk Analysis and Mitigation . . . 46
4.4 Application of Agile Methodologies . . . 48
4.5 Summary . . . 49

5 Design **50**
5.1 Software Architecture Design . . . 50
5.1.1 Data Model . . . 53
5.1.2 Visualiser . . . 54
5.2 User Interface Design . . . 55
5.2.1 Data Visualisation Designs . . . 58
5.3 Summary . . . 60

6 Implementation **62**
6.1 Overview . . . 62
6.2 Application Framework Selection . . . 63
6.2.1 The *Qt* Framework . . . 64
6.3 Application Structure . . . 65
6.4 Source Code . . . 67
6.5 Data Model . . . 67
6.6 Video Feed and Tracking System . . . 69
6.6.1 The *OpenCV* Image Processing Library . . . 70
6.7 Networking . . . 70
6.8 Data Transfer Format . . . 73
6.8.1 Constraints . . . 75
6.9 User Interface . . . 75
6.10 Visualiser . . . 79
6.10.1 Data Visualisations . . . 80
6.11 Robot Side API . . . 84
6.12 Summary . . . 85

7 Testing **86**
7.1 Continuous Integration Testing . . . 86
7.2 Manual User Interface Testing . . . 87
7.2.1 Method . . . 87

7.2.2 Results . . . 89
7.2.3 Fixes Implemented . . . 89
7.3 Data Model and Back End Unit Testing . . . 90
7.3.1 Results . . . 94
7.3.2 Issues with this Approach . . . 94
7.4 Verification and Validation Testing . . . 95
7.4.1 Results . . . 98
7.4.2 Analysis . . . 99
7.5 Summary . . . 100

8 Evaluation **101**
8.1 User Evaluation Sessions . . . 101
8.1.1 Participants . . . 102
8.1.2 Set Up . . . 102
8.1.3 Session Sequence . . . 103
8.1.4 Observations . . . 105
8.1.5 Questionnaire . . . 106
8.1.6 Analysis . . . 110
8.2 Comparison with Project Aim and Objectives . . . 111
8.3 System Limitations . . . 114
8.4 Project Exectuion . . . 115

9 Conclusions and Future Work **117**
9.1 Conclusion . . . 117
9.2 Future Work . . . 118
9.2.1 Additional Debugging Features . . . 118
9.2.2 Towards a Full Swarm Robotics Research and Development Platform . . . 119
9.2.3 Expansion of Supported Hardware . . . 120
9.2.4 Integration with Dedicated Augmented Reality Hardware . . . 121
9.2.5 Integration with Tablet Application . . . 122
9.3 Summary . . . 122

A Ethics Documents **123**
A.1 Ethics Form . . . 123
A.2 User Evaluation Session Consent Form . . . 129

B Gantt Chart **131**

C Source Code Files **133**

D Robot Side API **136**

E User Guide **138**

E.1 Overview 138
E.2 Incorporating the *SwarmDebug* API 139
E.3 Using the *SwarmDebug* Application 141
E.4 Compiling the Application from Source 141

F Test Results **143**
F.1 Manual UI Testing Results 143

Bibliography **167**

List of Figures

1.1 Debugging Information Abstraction Diagram . . . 3
1.2 Proposed System Architecture . . . 5

2.1 Army Ants Bridging a Gap [5]. . . . 12
2.2 Self Organised Feedback [11]. . . . 16
2.3 Swarm Craft GUI [13]. . . . 18
2.4 *NUbugger* GUI [16]. . . . 19
2.5 AR Furniture Visualisations [17] [18]. . . . 20
2.6 Microsoft HoloLens [19] 21
2.7 Sonar data visualisation. Collet and MacDonald [14] 23
2.8 Spatially situated data overlay. Garrido et al. [24] 25

3.1 Initial Survey Question 3 Responses 28
3.2 The e-puck Robot 32
3.3 E-puck IR Sensor Layout 33
3.4 E-puck Code Architecture 34
3.5 Tracking Camera Arrangement 36
3.6 Robot Arena and Tracking Camera 36
3.7 JAI-Go Camera 37
3.8 ArUco Markers 38
3.9 ArUco Marker Mounted on e-puck 39

5.1 Software Architecture Diagram 52
5.2 Data Model Diagram 53
5.3 Visualiser Render Process Design 54
5.4 UI Layout 55
5.5 UI Example 57
5.6 Data Panel Designs 58
5.7 Visualiser Settings Tab Design 59
5.8 Visualiser Overlay Designs 61

6.1 Application User Interface 63
6.2 Application Structure 66
6.3 Networking Code Flow Diagram 72
6.4 Data Format 73
6.5 Visualiser Panel 76

6.6 Robot List Panel 77
6.7 Data Panel 78
6.8 Visualiser Data Flow 80
6.9 Data Visualisations 83
6.10 Visualiser Settings Tab 84

7.1 Testing Window 94

8.1 User Evaluation Question 5 Responses 108

E.1 The *SwarmDebug* Application 138

List of Tables

3.1 Initial Survey Question 1 Responses 27
3.2 Initial Survey Question 2 Responses 28
3.3 Initial Survey Question 5 Responses 29

4.1 Development tasks. 44
4.2 Testing tasks. 45
4.3 Other tasks. 46

6.1 Robot Data Contents 68
6.2 Data Format 74

7.1 User Interface Elements for Testing 88
7.2 Data Model Test Cases 93
7.3 Verification Testing Results 99

8.1 User Evaluation Question 1 Responses 106
8.2 User Evaluation Question 2 Responses 107
8.3 User Evaluation Question 3 Responses 107
8.4 User Evaluation Question 4 Responses 107
8.5 User Evaluation Question 6 Responses 108
8.6 User Evaluation Question 7 Responses 108
8.7 User Evaluation Question 8 Responses 109
8.8 User Evaluation Question 9 Responses 109
8.9 User Evaluation Question 10 Responses 109

C.1 Application Code Files 133
C.2 Robot-side Code Files 135

D.1 Robot API 136

List of Abbreviations

YRL	**Y**ork **R**obotics **L**aboratory
SI	**S**warm **I**ntelligence
HRI	**H**uman **R**obot **I**nteraction
HSI	**H**uman **S**warm **I**nteraction
GUI	**G**raphical **U**ser **I**nterface
AR	**A**ugmented **R**eality
HMD	**H**ead **M**ounted **D**isplay
IR	**I**nfra **R**ed
VCS	**V**ersion **C**ontrol **S**ystem
MVP	**M**inimum **V**iable **P**roduct
FIFO	**F**irst **I**n **F**irst **O**ut
IP	**I**nternet **P**rotocol
TCP	**T**ransmission **C**ontrol **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol

For my Grandfather, who would have read this with great fascination and interest.

Chapter 1

Introduction

1.1 Background

Recent years have seen rapid development in robotics technology due to the constantly increasing availability of computing power, reductions in the cost of hardware such as digital sensors and actuators, and developments in the application of artificial intelligence to robot control. This has led to robots being used to perform increasingly complex tasks and solve ever more difficult problems. Many new areas of robotics research have emerged as a result, as researchers strive to find new and better ways to apply this technology, entering into problem domains once thought to be impossible for robots. Whole new robotics paradigms have been created as the standard model of a single, complex, expensive robot has been questioned, opening the door for cooperative robots, multi-robot systems, and more specifically swarm robotics.

Studies into the self-organising behaviour of social insect colonies, and the development of mathematical models based on these behaviours, led to the development of a field of research referred to as *Swarm Intelligence* (SI). The aim of this field is to study how large numbers of individual agents are able to solve problems collectively, with each agent using only local information, and without any centralised control. Swarm Robotics developed from a desire to apply these concepts in practice to real world problem solving. Dorigo et al. describe swarm robotics as '*the study of how to design groups of robots that operate without relying on any external infrastructure or on any form of centralised control ...* [where] *the collective behaviour of the robots results from local interactions between the robots and between the robots and the environment*[1]'. Swarm robotics has since emerged as a promising area of research for solving problems which would be infeasibly difficult or expensive for a conventional robotics approach.

1.2 Project Context

Developing and debugging robotics behaviours has always been a challenging task. Whilst traditional software is run in a purely digital environment with a tightly controlled set of inputs and outputs to and from the physical world, robots must interact constantly with the physical world in order to satisfy their intended purpose. Robots are therefore subject to a much wider array of inputs and outputs, and to a huge number of changing variables within their environment at any given time. Often these variables and inputs are continuous in nature, rather than the discreet inputs more commonly used by traditional computers. This makes detecting, reproducing and correcting specific bugs in a robot's behaviour significantly harder than in traditional software. A large number of continuous inputs leads to a theoretically infinite set of possible input configurations, and can therefore make reproducing the exact conditions under which a bug occurred extremely difficult if not impossible.

Another of the main difficulties comes from the layers of abstraction between the real world, the robot, and the human developer. There is a potential disconnect between the robot's interpretation of the world and the reality of the world itself. Inaccuracies in this interpretation can be caused by any number of issues, including sensor hardware problems as well as software bugs. This can cause erroneous behaviour that might be wrongly attributed to a bug in the robot's behavioural code or decision making, rather than its perception. This issue can be compounded by the fact that the human operator's knowledge of the robot's interpretation of the world might also be inaccurate or incomplete. Figure 1.1 shows these different layers of information abstraction when dealing with a robotic system. The arrow highlighted in red shows where many of these abstraction related debugging difficulties occur. Retrieving human readable information from a robot in a timely manner whilst it is running is often non-trivial, and what the robot sees and what the human operator thinks the robot sees may differ significantly.

This problem is made more complex when working with multi-robot systems, and especially swarm robotics. Introducing multiple robots multiplies the number of potential variables and increases the amount of information required to describe the system. Hence both the number of points where a bug may be occurring, and the amount of information the operator needs in order to locate it are also increased. The decentralised nature of swarm robotics systems further exacerbates this problem through the lack of a single, central control point where information for the whole system can be retrieved.

Traditional software debugging tools are predominantly text based, and often require that program execution is paused in order to examine internal values. These tools are therefore often insufficient for robotics applications, as the quantities of data produced by a robot's sensors, and used to make its decisions, can be too large

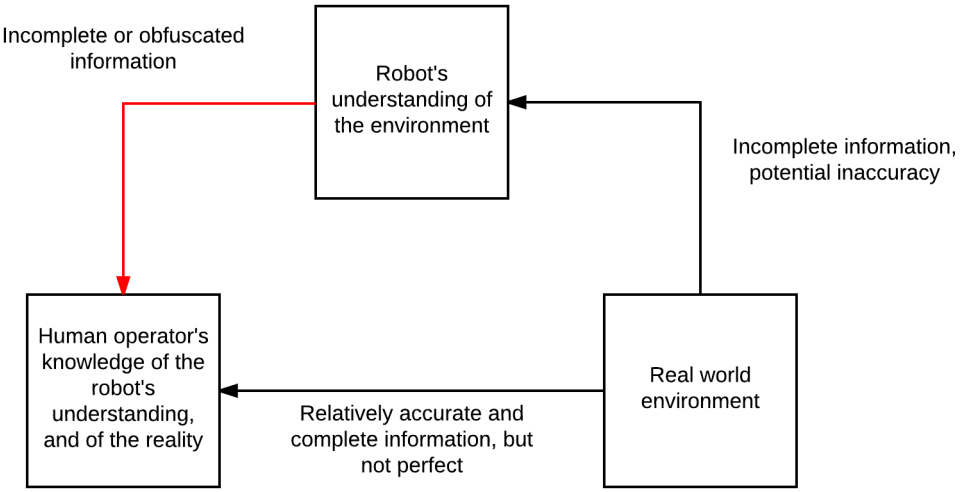


FIGURE 1.1: Layers of information abstraction in robotics debugging.

and vary too rapidly for any sensible textual representation. Furthermore pausing a robot's execution can be undesirable, especially if it is operating in an active real world environment. Once again swarm robotics and multi-robot systems compound this problem by increasing the amount of data involved, making textual representations less likely to be sufficient, and by involving multiple data sources, which traditional debugging tools are rarely designed to handle. New tools and techniques are therefore needed to retrieve data from these robotic systems and present it in a manner that allows humans to understand and process it effectively.

Virtual and augmented reality technologies have seen significant progress in the last few years, and are beginning to emerge into a large number of real-world applications. Augmented reality in particular offers a promising new method for interacting with robots, and may help to overcome some of the limitations of traditional debugging tools. An augmented reality system works by capturing a view of a real world environment through a camera, and augments it with computer generated graphical elements. These graphical elements or 'virtual objects' are usually positioned within the space such that they either appear to be real, physical objects, or relate to other physical objects within the space. Augmented reality systems can therefore create 'hybrid-' or 'mixed-reality' environments, where users are able to interact with real world objects and virtual, computer generated ones simultaneously.

This shows particular promise when combined with robotics, as a mixed-reality space is one that can be shared and understood by both humans and robots. The virtual elements of an augmented reality environment are simply representations of digital data, and this data can be readily understood by a digital system such as a robot. By creating tools which effectively utilise augmented reality techniques

and mixed-reality environments in conjunction with robots, it should be possible to broaden the human-robot communication channel, introducing novel ways for humans and robots to communicate and share information, with implications for all aspects of human-robot interaction. Considering debugging specifically, it is easy to imagine how converting a robot's data such as sensor readings to graphical representations, correctly positioned to reflect their spatial significance, could improve a human operator's ability to identify problems with a robot's perception and behaviour.

1.3 Project Concept

This project focuses on trying to mitigate some of the problems discussed in the previous section by improving a human operator's access to a swarm system's internal information, thus improving the timeliness with which bugs in the system can be identified, located and fixed. This means designing and implementing a system capable of collecting information from multiple sources and presenting it all in one place, in a human readable manner, in real time. The information sources to be used include the individual robots themselves, as well as a live video feed of the robots and their environment. The project also attempts to incorporate ideas and techniques related to augmented reality, in order to represent data retrieved from a robotic system in ways that can be more intuitively understood by a human operator than purely textual and numerical representations.

This project attempts to create a software application and associated wireless data transaction format capable of presenting a user with a single, coherent, and highly readable interface through which they can view relevant information about a swarm and it's constituent robots in real time. This includes the use of a video based tracking system to monitor robot positions, and provide the user with a view of the robots' environment. This view can then be augmented with graphical representations of relevant elements of the retrieved robot data, such as sensor readings. The robots will communicate data to the computer running the application wirelessly. The initial target robot platform is the widely used *e-puck* robot [2], equipped with a Linux extension board and WiFi adapter. The e-puck platform is discussed in greater detail in chapter 3. The diagram in figure 1.2 gives a logical representation of the proposed system architecture, in terms of its component parts, including the e-puck robots, tracking camera, and the application's host computer. This report describes the design, implementation and testing of this system, and includes details of the steps undertaken to evaluate its effectiveness. Some portions of this report appeared previously in a similar form in an *Initial Report* document, and are included here for completeness, with minor alterations.

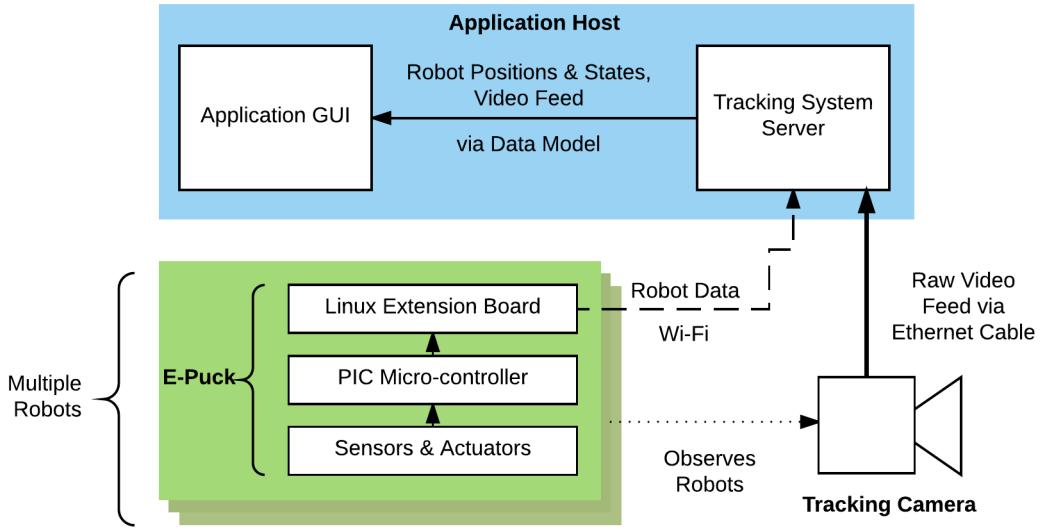


FIGURE 1.2: The proposed system architecture.

1.4 Aim and Objectives

Having gained an understanding of the context for this project in section 1.2, and outlined the concept for the system in section 1.3, the project aim can be summarised in the following statement:

This project aims to understand the needs of a swarm robotics researcher or system developer when attempting to debug their system, and create a computer application which allows a user to monitor the state and behaviour of a robot swarm system in real time, employing augmented reality techniques where possible, and thus improving the ease and efficiency of the debugging process.

The objectives required to achieve this aim are as follows:

- Utilise existing fiducial marker based tracking technology to track the position of individual robots within a swarm over time.
- Develop code to allow multiple robots to communicate information regarding their internal state, sensor readings and decision making to a central application wirelessly via a network.
- Develop a data model that allows the central application to store information received from the robots, and update it as new information arrives.
- Develop code to ascertain higher level data related to the robots, such as recent movement history or state transition history, and add this data to the model.
- Design and implement a user interface which is easy and intuitive to use, and presents data from the data model to the user in a human readable manner.

- Develop a visual display component for the application, which presents the user with a live video feed of the robot swarm, augmented with relevant and spatially situated information relating to the robots, by fusing data obtained from the robots with data obtained from the tracking system.
- Develop the user interface in such a way as to allow the user to filter out information that is not currently relevant, and to compare and contrast information related to specific robots.
- Design and implement the system in a modular way so as to allow for relatively simple integration with other swarm robotic platforms and tracking systems in future extensions.

1.5 Functional Specification

When developing software of any kind it is common practice to define a *functional specification* prior to starting development. This specification describes the functionality required in the software in order for it to satisfy its purpose. The specification presented here is separated into core and secondary requirements. Core requirements are considered essential to the satisfactory delivery of the software. Secondary requirements are desirable but not strictly necessary, and will be satisfied where possible, given the time constraints of the project.

Core Requirements:

1. Must comprise a PC application.
2. Must be capable of receiving data related to the state of multiple robots.
3. Must be capable of receiving positional data for the same set of robots.
4. Must be capable of receiving a live video feed of the robots in their environment.
5. Must collate received data and present it to the user in a combined graphical form.
6. Must present auxiliary, non-spatial data to the user in textual or other forms.
7. Must update in approximately real time.
8. Must at minimum support the e-puck robot platform.

Secondary Requirements:

1. Should use a modularised structure.

2. Should exchange data between the robot platform and the application using a platform-agnostic, extensible protocol.
3. Should provide a basis for interoperability with a number of robotics platforms.
4. Should allow the user to configure the displayed data.
5. Should employ a model-view-controller (MVC) software architecture.
6. Could provide the user with ways to configure and display custom data types.
7. Could allow the user to compare data on two or more specific individual robots.
8. Could calculate and display swarm-level meta-data and statistics.
9. Could generate log files of robot activity over a user defined period.

1.6 Report Structure

This report consists of the following chapters:

Introduction. An introduction to the project, including details of the background and context, a general overview of the system implemented, and a statement of the aims and objectives, which are then used to form a functional specification for the system. The York Robotics Laboratory is introduced, and the ethical considerations of the project are addressed.

Literature Review. A review of the existing literature relevant to the project topic, wherein individual pieces of research and writing with relevance to a specific area or areas of the project are highlighted. The place this project intends to take amongst this literature is then established.

Project Plan. An outline of the plan formed at the beginning of the project, including details of the tasks to be undertaken and the time allocated for each, as well as an assessment of the associated risks and the mitigation steps taken.

Problem Analysis. An overview of various information relevant to the problem domain, including details of the hardware infrastructure utilised in the system implementation, and the results of the initial user survey which influenced the design and implementation.

Design. A detailed description of the design of the system, including details of key design decisions and the reasons behind them. This section is divided into two main topics. First the structural design of the system is described. This is then followed by the design of the user interface.

Implementation. A full description of the system implementation, including implementation details for all of the key components, and explanations of how these components connect and interact. Particular attention is paid to the movement of data through the system. Details of the implementation of the user interface, following the design laid out in the previous section, are also given, with images for reference.

Testing. A description of the testing processes used to validate the software portions of the system. The results of these tests and the resulting fixes and changes are presented.

Evaluation. A description of the steps taken to evaluate the system itself and the project as a whole, and the findings of this evaluation. The trial sessions undertaken with potential system users are described and the results presented and analysed, leading to an assessment of the effectiveness of the system, its shortcomings, and some suggestions for possible improvements. This is followed by a more general assessment of the system in relation to the core aim and specific objectives of the project. Various limitations of the system are also summarised.

Conclusion and Future Work. A summary of the results of the project and the conclusions drawn. These are related back to the original aims and objectives, and the system is considered in terms of its local context within the York Robotics Laboratory and the wider context of the swarm robotics field. A number of suggestions for possible future work to extend and improve the system are then made.

1.7 York Robotics Laboratory

This project was carried out in conjunction with the York Robotics Laboratory (YRL). Established in 2012, the YRL is jointly run by the Department of Electronic Engineering and the Department of Computer Science at the University of York. All of the hardware infrastructure used in this project was provided by the YRL, and the primary use case for the system is in aiding the development of swarm behaviours for experimental research being conducted by members of the laboratory.

1.8 Ethics

After consideration of the University's code of practice and principles for good ethical governance, no ethical issues were identified in this project. The contents of

both the initial survey and the user evaluation sessions did not raise any significant ethical issues. All participation was voluntarily, and all participant data was stored carefully and anonymously. The ethics compliance form completed for this project, and ratified by the project supervisor, is available in appendix A.1. The consent form completed by participants in the user evaluation sessions is available in appendix A.2.

Chapter 2

Literature Review

2.1 Overview

This section presents a review of some of the literature from the field of swarm robotics, beginning with some general summaries of the field's fundamentals, followed by a number of specific pieces of research from topic areas with particular relevance to this project. These areas include human-swarm interaction, robotics debugging, and robotics-focused augmented reality research. Finally a number of papers which connect augmented reality techniques directly to swarm robotics, and are therefore well aligned with the aims of this project, are discussed and critiqued. The results of this literature survey informed the project direction significantly, and formed the basis for many of the design and implementation decisions made later on. This literature review is also presented with the aim of providing the reader with the base of knowledge required to better understand the project.

2.2 Swarm Intelligence and Swarm Robotics

An understanding of the fundamental concepts of Swarm Robotics, and to a lesser extent Swarm Intelligence, was deemed key to producing an application that is useful in practice, and will help a reader to better understand the purpose and aims of the project. A deep understanding of the technical details of specific swarm behaviours or implementations is not a priority for understanding this project, as the application aims to be more broadly applicable to a wide range of swarm systems. Emphasis has instead been placed on understanding the general classification of swarm robotic systems, relevant problem domains, and recurring concepts, so that the system might better serve researchers in the field.

Sahin [3] presents a summary of the key concepts of swarm robotics, and attempts to offer a coherent description of the topic. He notes that a key difference from other multi-robot systems is the lack of centralised control, and the idea that

desired swarm level behaviour should emerge from simple local interactions between robots, and between the robots and their environment. He also notes some of the key motivators behind Swarm Robotics research, stating that a swarm robotics system would ideally exhibit "*robustness*", "*flexibility*" and "*scalability*" [3].

Robustness refers to the swarm's ability to continue to function should one or more individual swarm members suffer a failure of some kind. Flexibility refers to the swarm's ability to adapt to changes in the environment without the need for re-programming. Scalability describes the idea that a swarm should be functional at a range of sizes, and that ideally the number of robots in the swarm could be increased or decreased depending on the demands of the task. These descriptors should be taken into account by the design of the system presented in this project. In order for the system to be able to work well with robust, scalable swarms it should adapt easily to variation in the number of robots being monitored. Newly detected robots should therefore be incorporated seamlessly. The video feed component of the system will allow a user to see the environment the swarm is interacting with and observe how robots respond to changes within it, and therefore judge the extent to which the swarm exhibits flexibility.

Sahin [3] goes on to describe several classes of application for which Swarm Robotics systems might be well suited. Tasks that cover a region could benefit from a swarm's ability to distribute physically in a space according to need. Dangerous tasks could benefit from the relative dispensability of individual robots in the swarm; should one be damaged or destroyed the swarm could continue to function, and it would be less costly than the loss of a single, complex, expensive robot. Tasks requiring scalability are good candidates, as discussed before, and tasks that require redundancy are also highlighted, as swarm systems should have the ability to degrade gracefully, rather than suffering a single catastrophic failure. Through this generalisation of the application areas, insight can be gained into the kinds of work swarm robotics researchers are likely to be doing, and this should inform the design of the application. Overall Sahin's paper [3] provides a coherent, succinct overview of the field, and although it is now over a decade old the concepts covered remain relevant.

The book '*Swarm Intelligence: From Natural to Artificial Systems*', by Bonabeau, Dorigo and Theraulaz [4], provides in its introductory chapter a good overview of the biological concepts and animal behaviours which inspired much of the research that led to the creation of the field of swarm intelligence. Fundamental concepts such as self-organisation and decentralised control are discussed. In order to be a useful tool in the swarm robotics research space, it is important that the system developed during this project does not violate these core principles of the swarm robotics paradigm. Therefore the system should not facilitate low level control of the robots or allow for forms of communication and data exchange that might invalidate the self-organising, decentralised nature of the swarm's behaviour.

FIGURE 2.1: A swarm of army ants forming a living bridge to cross a gap, in an example of a self-organised swarm behaviour [5].

The later chapters [4] provide a detailed look at several key insect behaviours, and how mathematical models and algorithms can be derived to mimic them. An understanding of these behaviours and models can offer insight into what information the application might need to expose to allow a user to validate the correct operation of a swarm behaviour based on these concepts. A number of these algorithms focus or rely on data related to the position and movement of individuals within the swarm, and the swarm as a whole. The system developed during this project should reflect the importance of positional data and provide features which aid the user in interpreting it.

In a more modern work, Brambilla et al. [6] focus heavily on the engineering practicalities of designing, implementing and testing swarm robotic systems. The authors then apply this focus as a means by which to classify and critique a large body of swarm robotics research, noting that although much work has been carried out regarding the design and analysis of swarm behaviours, other areas including maintenance and performance measurement are heavily lacking in research contributions [6]. Debugging can be considered a maintenance task, and the system developed in this project has potential in both maintenance and performance measurement applications. It is hoped therefore that this work might contribute to this area of the field, by investigating through implementation the practicalities of a generalised swarm maintenance and observation software application. The authors [6] later note that human-swarm interaction remains an open issue, and will be key to

realising functional, real-world swarm robotic systems. They identify a number of works related to human-swarm interaction, almost all of which focus on the task of controlling a robot swarm, and investigate different methods by which a user might insert data into the swarm system. Little consideration appears to be given, both in this paper and throughout the literature, to the task of monitoring a swarm, and best practices for retrieving information in a manner that is useful to a human operator.

2.3 Human Swarm Interaction

This project focuses on a piece of software which forms an interface between a human operator and a robot swarm. A relevant area of research is therefore Human-Swarm Interaction (HSI). Research in this area focuses on the different ways in which humans and robot swarms can interact, the different roles humans take whilst interacting with robot swarms, and the best practices for facilitating this interaction given different aims, and different user roles (developer, researcher, end user, etc.). The two key challenges of HSI are *control* and *monitoring*. The former refers to how best to allow a human operator to direct the behaviour of a decentralised swarm, whilst the latter refers to how best to retrieve data from a swarm and present it in a useful, human readable manner. This project is related to debugging robot swarm behaviours, which is primarily a swarm monitoring task, and therefore this section focuses on the monitoring side of HSI.

In their paper '*Human Interaction with Robot Swarms: A Survey*' [7] Kolling at el. begin by noting the lack of research into methods for interfacing humans and robot swarms. They suggest that real-world applications for swarm robotics systems are now within reach, and that discovering effective methods for allowing humans to control and/or supervise swarms is a key barrier to realising these systems. The paper [7] provides a detailed analysis of human swarm interaction from a number of different perspectives. Of relevance to this project is the statement on page 15 that "*Proper supervision of a semiautonomous swarm requires the human operator to be able to observe the state and motion of the swarm, as well as predict its future state to within some reasonable accuracy*" [7]. This statement lends credence to the aims of this project, as swarm supervision and swarm debugging are highly comparable tasks; both involve observing the swarm whilst performing its task and determining the validity of the behaviour observed. The system developed in this project should allow the state of the swarm, including the internal state of individual robots, to be observed simultaneously with the physical positions and motions of the robots within their environment. The paper [7] goes on to suggest that by observing the swarm over time the human operator will be able to provide '*appropriate control inputs*'. In the case of this application, rather than providing control input, the human operator

will be seeking to identify faults, and provide appropriate corrections to the system, however the concept of state visualisation remains relevant.

Rule and Forlizzi [8] present a thorough examination of the complexities of human robot interaction (HRI) when dealing with multi-robot (and multi-user) systems. Much of the paper focusses on control methods, which are not directly applicable to this project, however section 2.4 titled *Salience of Information* discusses the task of designing interfaces for displaying information about multi-robot systems to a human operator in a manner which is both information dense and rapidly understandable. The authors note that the use of colour has been shown to improve interface readability [9], and that the brain has been shown to process text faster than images [10], hence complex icons should be avoided. These ideas should be incorporated into the design of the application user interface for this project. A range of different designs could be explored, including finding a balance between the amount of information displayed graphically, and the amount displayed textually, and deciding whether to use colour to differentiate between individual robots, or to differentiate between different types of data, or a combination of both.

The authors [8] then go on to use '*contextual inquiry*' interviews with robot operators to determine a set of questions which, if answered, should allow for robust robot operation. Of relevance to this project are the four questions in the *human-robot* category. These are 1) "*What mode, state, and environment is the robot in and how will this affect my commands?*", 2) "*Which robot needs my attention?*", 3) "*What is each robot accomplishing?*", and 4) "*What can I accomplish with this robot?*". Providing the user with answers to questions two, three and four is beyond the scope of this project, however the system should provide the user with an answer to question one, by allowing access to state and environment information simultaneously.

Once again although the authors [8] consider only a robot control perspective, these questions remain broadly relevant to a debugging perspective also. For example, question one could be rephrased as *What mode, state, and environment is the robot in, and is this having the correct effect on its behaviour?* to better fit the use case of this project. When discussing the appropriate level of display salience for robot state awareness in section 4.8 [8], the authors note that users wanted the ability to "*select which aspects of robot information to view at any one time*". They also state that users wanted basic overview information, with the ability to access a more detailed view specific to one robot when "troubleshooting". This configurable, layered approach to information display and access should be incorporated into the user interface design for the system.

The authors [8] do not state how they determined that the set of situation awareness questions presented ensure robustness. The contextual inquiry method is potentially subjective, suggesting that answering only these questions may not guarantee full awareness. Other factors which a system user could benefit from an awareness of should be considered. For example the authors mention the robot's mode, state and environment, but do not include the robot's sensor data. Furthermore care must be taken when applying the results of this work to swarm robotic systems, as it was not produced with swarms specifically in mind. Therefore applying the results within, without also considering needs specific to swarm robotics, could lead to too strong a focus on the actions of individual robots, and a lack of focus on the overall behaviour of the swarm.

Podevijn, O'Grady and Dorigo [11] present a novel method for obtaining feedback from an active robot swarm. The authors propose that information sent back from a self-organised swarm to an operator should itself be self-organised. They note a number of motivations for this approach. Firstly, should every robot simultaneously report information regarding its own "*local worldview*", the authors suggest [11] that the operator would be over-burdened with an excess of information, and would therefore need further tools of some kind to parse this data effectively. They also suggest that the required communication infrastructure would increase the overall system complexity, and the hardware requirements of each robot. The volume of communication traffic for a large swarm might also prove impossible for a network to handle.

Their work [11] demonstrates a number of possible methods by which a swarm of robots might report information in a self organised manner, in the hope of combining and reducing the information that needs to be reported. Whilst completing a grouping task, coloured LEDs were used to indicate visually which group each robot was a part of. This can be seen in figure 2.2. Another proposed mechanism [11] would use the formation of the swarm to indicate information about their behaviour, for example when moving collectively in one direction the robots would arrange into an arrow formation indicating the direction of movement. This mechanism was proposed only in concept, and not implemented on a real swarm.

This self organised approach to information reporting [11] has a number of issues. Firstly, the additional behaviour required to report information in a self organised manner may negatively impact the original desired behaviour of the swarm. In the case of the arrow formation example, having the robots move into a readable formation might disrupt other position-based behaviour. Implementing these information-reporting behaviours might also add significant complexity to the robots behavioural code, further increasing the difficulty of the already complex task of implementing a swarm behaviour. Furthermore, from a debugging perspective, this kind of high level, aggregate data reporting does not allow access to the low level

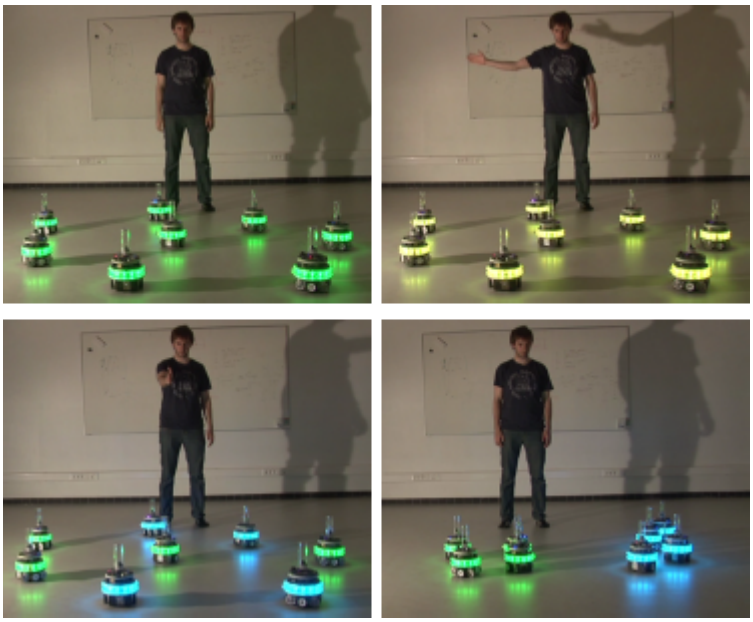


FIGURE 2.2: Self-organised, colour-based feedback during a selection and grouping task [11].

internal state information that can be essential to isolating the cause of an observed issue.

The communication infrastructure requirements of a system that allows a large number of robots to communicate individually with the host are a concern, however in practice robot swarms tend to be composed of numbers of robots which are not unmanageable given modern networking technology. Standard WiFi networks are capable of supporting relatively high numbers of connected devices, whilst developments in mesh-network technologies driven by the emerging Internet of Things sector have made it possible for networks of very large numbers of devices to be achieved with relatively modest hardware [12]. In addition, the complexity increase due to greater networking and communication requirements is a known quantity; these technologies already exist and have standardised implementations. In contrast, implementing additional swarm behaviours to report information adds unknown complexity, as a reliable, formalised method for developing arbitrary swarm behaviours is still an open problem.

Finally the authors' assertion that the volume of data received from a swarm of individually reporting robots would overwhelm an operator is potentially true. However, a centralised host receiving all of the reported data is far better positioned than the swarm itself to process and condense this information into a form that the user can handle, without sacrificing on detail. This can also provide the user with access to variable levels of information, giving them greater control overall. In summary the concept of self-organised feedback as presented in [11] shows promise for situations where relatively simple, high level information is required to be directly

apparent to a system user. However in practice the added complexity at the behavioural level, and the potential to interfere with other desired behaviour, may well outweigh the benefits of reduced communications complexity and infrastructure. By creating a generalised platform for swarm robot information reporting, this project could establish communications infrastructure requirements as a known, fixed quantity, which can potentially be re-used across multiple swarms.

McLurkin et al. [13] discuss the practical considerations of interfacing a human operator with their large swarm of 112 robots. The relevant portion of this work discusses the use of "*utility software for centralised development and debugging*" [13]. The authors note the success of graphical interfaces based on those found in *Real Time Strategy* video games for centralised data collection and display. Their *SwarmCraft* graphical user interface (GUI), shown in figure 2.3, can display information received regarding individuals, groups, or the whole swarm in a number of graphical forms. The authors [13] note that one potential issue is keeping the code for displaying the data and the code on the robot side in sync, such that the GUI can always correctly display the received data. The idea of a standardised data transfer format investigated by this project could provide a potential solution to this issue.

The authors [13] go on to discuss the problem of using monitors to display the robots' internal state information when debugging group behaviours, as a user must continuously switch between watching the swarm and looking at the monitor for detailed information. The authors use a 'global output' system of LEDs and sounds to solve this issue. This output is however inherently limited by the amount of information that can be represented using the small number of LEDs, and the quality of information acquired from a swarm of robots all generating sound at once. This approach also requires specific hardware to be present on each of the robots, such as the LEDs and a speaker. Augmented reality (AR) techniques therefore offer a potentially more powerful solution; by superimposing the information that would traditionally be found by looking at the monitor onto a view of the robots themselves, the user can access all available information without looking away from the swarm's activity. Furthermore the requirement for the global output hardware is removed, and the user does not need to learn how to correctly interpret these somewhat cryptic output systems, as an AR solution could display information in any form necessary, including text.

2.4 Robotics Debugging

'*Debugging*' is the name given to the process of fixing problems or issues (commonly referred to as bugs) in a piece of software. Well established tool-sets and techniques exist for debugging traditional software. However debugging robotic systems is a fundamentally different and more complex problem, as discussed in section 1.2. The

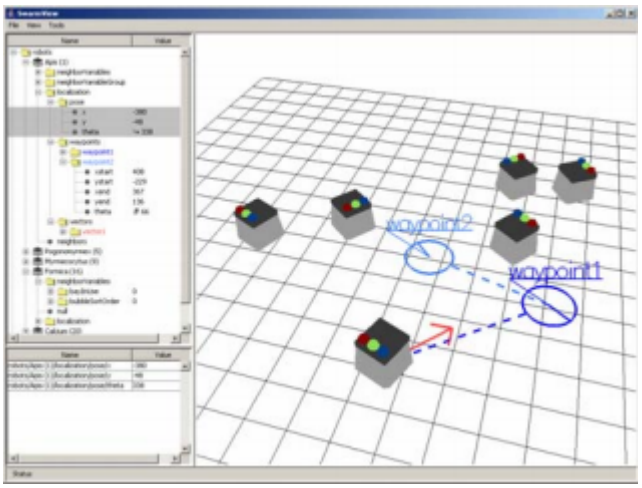


FIGURE 2.3: A screenshot of the *SwarmCraft* GUI, developed by McLurkin et al. [13].

literature reviewed in this section examines the challenges of robotics debugging, and presents some tools and techniques for reducing the difficulty of this task.

Collet and MacDonald [14] describe in detail the difficulties in debugging robotics systems. The authors identify that the difficulties in developing and debugging robotics applications when compared to traditional software arise from either the environment of the robot - which will often be "*uncontrolled*" and "*dynamic*" - or from the mobile nature of the robot. Because the environment a given robot operates in is a real world space, the level of control that can be exerted over it by the researcher or operator is inherently limited [14]. The environment may therefore change over time, exhibit imperfections, and include other time-varying elements. A robot is a physical actor and will likely experience dynamic change in its sensor readings and its relationship to the environment over time. This is especially true for mobile robots, whose position and orientation will change over time. The behaviour of the robot often largely depends on these highly variable factors, and therefore replicating a given behaviour exactly becomes almost impossible.

The authors go on to state that difficulties in debugging often arise from "*the programmer's lack of understanding of the robot's world view*" [14]. It can therefore be extrapolated that for a multi robot system such as a robot swarm this problem would be exacerbated. Each robot will have its own perception of the environment, which will differ based on differences in the robots' positions and orientations as well as variations in the instrumentation of each robot. For a multi-robot system the programmer is required to have an understanding of not just one but multiple world views, adding yet more potential for error and inaccuracy, and further obscuring bugs or behavioural issues that the programmer is trying to diagnose. This work [14] suggests that developers will need specific tools which enhance their understanding of the robots' world view in order to develop effectively for swarm robotic

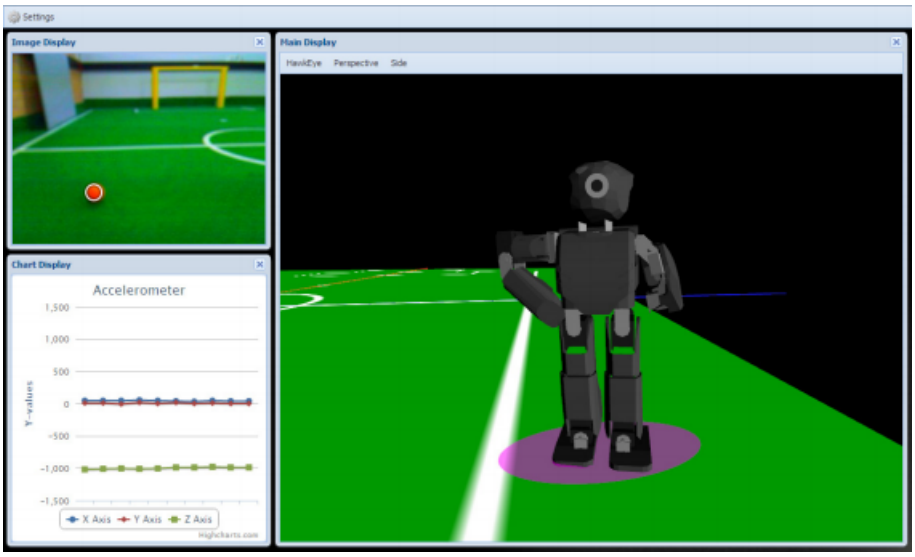


FIGURE 2.4: The *NUbugger* GUI, a real time, visual, robotics debugging utility developed by Annable et al. [16].

platforms. This need forms the mandate for the majority of this project. Collet and MacDonald [14] go on to discuss the use of augmented reality to satisfy this need, and present an augmented reality based software tool for this purpose, which is discussed in Section 2.5.

Gumbley and MacDonald [15] identify one of the core issues in debugging robotic behaviours using traditional debugging software to be the assumption of a "*deterministic, suspendable environment*". This assumption rarely holds true for robots, due to their existence in a real world environment. The authors note that traditional debugging constructs such as breakpoints pause code execution, but cannot for obvious reasons also pause a robot's environment. This allows the robot's environment to change whilst execution is paused, therefore affecting the robot's behaviour. In the case where a breakpoint has been inserted to try to isolate the cause of a previously observed fault, this change in behaviour may also stop the fault from occurring. The authors [15] go on to consider a number of possible methods by which a developer can obtain information without pausing execution, including live data extraction which is the method chosen in this project. They note that adding code to extract information without pausing execution has the potential to affect robot behaviour. This is a salient point, and should be considered when implementing the robot-side portion of this project, where care must be taken to minimise the affect data reporting has on the execution of the robot's actual behaviour. Issues of this nature could be caused by the data reporting code taking too long to execute, thus disrupting robot behaviour. Keeping the data reporting code lightweight and efficient should therefore be a priority.

In their '*NUbugger*' system [16], Annable, Budden and Mendes implement a real

time, visual debugging system for robotics. The authors begin by noting that the continually increasing complexity of robotic hardware and software has lead to typical debugging approaches based on diagnosis via observation of high-level symptoms becoming increasingly less useful. They note the need for real time, visual tools, to reflect the active, physical nature of robots. The authors system [16] streams all critical system information from their robot to a web server in real time, where it can be viewed in a graphical client. The graphical interface, shown in figure 2.4, includes a visualisation of the humanoid robot position, orientation and pose, based on its own self-localisation belief and servomotor sensor readings. It also includes a live feed from the robot's built in camera, and graphs of real time sensor data.

The system has been used to diagnose a number of real, low-level issues [16] relating to the robot's image processing and object detection functionalities, demonstrating the effectiveness of real time, visual debugging tools. The use of a web server in the authors' implementation [16] allows for the collection of information regarding multiple robots, and the display of this information in multiple clients, making the system highly versatile. However one possible limitation introduced as a result is the need for a number of relatively complex networking libraries to be running on the robots, and a high connection bandwidth, in order to send the required information, limiting the system to use on relatively complex, high performance robot platforms. The system is also currently only targeted at a single robot platform.

2.5 Augmented Reality and Single-Robot Systems

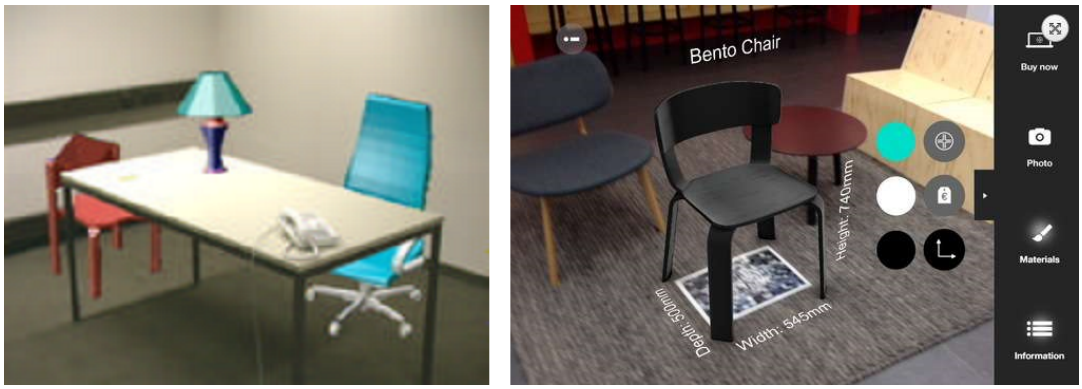


FIGURE 2.5: Left: An early example of an AR lamp and chairs augmenting a real table [17]. Right: A modern furniture visualisation app, augmenting a real space with a virtual chair [18].

Augmented Reality (AR) is the term used to describe the process of super-imposing 'virtual' objects and elements, in the form of computer generated graphics, on to a live video image of the real world [17]. The general aim of any augmented reality

FIGURE 2.6: The Microsoft HoloLens head mounted display [19].

system is to give the impression that the virtual objects being rendered are actually situated within the user's real environment, and to allow the user to interact with them as such. True augmented reality is sometimes defined as requiring the use of a *head-mounted display* (HMD), as this leads to much more immersive qualities akin to virtual reality, to which augmented reality is closely related. However this is not a universal requirement, and in one of the most widely cited surveys of the topic, Azuma [17] provides a definition which states only three requirements of an AR system; the combination of virtual and real content, real time interaction, and 3D rendering. Figure 2.5 shows on the left the example image used in Azuma's [17] survey. It includes a view of a real table, augmented by virtual furniture. On the right is a modern example from a furniture testing app [18].

Augmented reality presents a powerful tool for debugging robots as it allows information gathered by a robot about an environment to be superimposed onto a real world view of that environment, such that discrepancies and inconsistencies between this information and the real world become inherently obvious. More generally, augmented reality can be used to create a shared space between a human and a robot. Even as early as 1997, Azuma [17] notes that augmented reality has a potential application in robot path planning. One limitation of Azuma's survey is its age, as the two decades since its publication have seen rapid development in computing power and computer graphics, as well as augmented reality hardware. High fidelity, consumer HMDs, such as the *Microsoft HoloLens* [19] shown in figure 2.6, have been beginning to emerge into use within real world applications since the mid-to-late 2010s [20]. A more recent survey of the topic from 2014 [21] agrees with many of Azuma's earlier definitions, but provides many examples of new applications of AR technologies. The author notes the power of AR as a new paradigm for human computer interaction, and the wealth of potential application areas, including robotics.

Milgram et al. [22] discuss the different communication formats used to interface between humans and robots, grouping them into "*continuous*" and "*discrete*"

formats. For any communication involving a spatial or temporal component, the process of converting to and from a discrete format in order to transmit this information is an unnecessary burden. Both humans and robots use the continuous spatial dimensions, and humans have an inherent, instinctive understanding of physical things expressed in three dimensions. The authors [22] therefore identify that augmented reality provides an excellent means of supporting the communication of spatial information. Their paper focuses on the combination of stereoscopic displays and computer generated graphics to allow for more intuitive control of robotic systems. In the case of this project the concept is reversed; robots reporting spatial information for validation by a user should do so in a format which is inherently continuous such as a graphical visualisation in AR, rather than one that is discrete such as text-based numerical output. This should in theory reduce the time required for a human to process the information. The authors note [22] that the ideal system utilises both discrete and continuous formats where appropriate to best communicate the required information, and is ergonomically designed to allow the user to make use of both easily and intuitively.

Like much of the HSI literature surveyed, this paper [22] is focused primarily on robot control, rather than observation and monitoring. In spite of this much of the content of the paper remains applicable. Since the paper was written, just over twenty five years ago, major advancements have been made in virtually every area mentioned, including the quality and precision of robotic systems, their cognitive, perceptive and decision making abilities, augmented reality technologies and robotic autonomy. Because of this, some of the content of the paper has fallen out of date. Specifically, the assumption that robots lack the level of autonomy required to survey their environment and then form and execute a series of steps to carry out a relatively high level task, such as "find and go to object Q"[22], is no longer necessarily true. A number of modern robots possess sufficient sensing capabilities, processing power and cognitive programming to perform such tasks based on high level commands. This does not however devalue the AR methods discussed within, and given the increased complexity and sensing capabilities of modern robots, AR based methods of interaction actually have more potential than ever. The paper however makes no mention of the potential for an AR system to report data from a robot's sensors visually, which may be attributed to the technological limitations of the time rather than an oversight by the authors.

Collet and MacDonald [14] suggest that augmented reality tools can address and mitigate some of the robotics debugging issues discussed in section 2.4 by superimposing graphical representations of the robot's understanding of the environment on top of a live view of the environment itself [14]. Hence the programmer is able to see how the robot has interpreted the environment, and identify inconsistencies. The authors describe the image of the real world environment as the "*ground truth*" against which the robot's view can be compared and contrasted [14]. Figure 2.7

FIGURE 2.7: Sonar sensor data visualisation in Collet and MacDonald's system [14].

shows a visual example of this technique, where the data received from the robot's sonar sensors is converted to spatially situated 3D shapes and superimposed over the live image, and can therefore be verified visually by the user.

The application developed during this project closely follows this paradigm; allowing the user to identify bugs by comparing the robot's knowledge of its environment and its decision making factors (collectively referred to as its state) with a view of the environment, in real time. The application aims to apply this concept specifically to swarm robotics systems, and therefore must allow the user to compare the states of multiple robots with the environment simultaneously. From the perspective of each robot in the swarm, the other robots will form part of the environment, therefore the application must take this into account when displaying the information. Because of the large increase in information from a single-robot system to a multi-robot one, it becomes important that the application provides a way for the user to filter what information is displayed, allowing them to focus on the primary aspect under test. Filtering also allows the user to compare and contrast specific robots against one another by filtering out information related to other robots, or by displaying in more detail information related to the robots of interest.

2.6 Augmented Reality and Multiple-Robot Systems

A number of pieces of research have investigated different ways to monitor and interact with multi-robot systems, using a combination of augmented reality technology and HSI techniques. These similar works are summarised and reviewed in

this section.

Daily et al. [23] present a technique for retrieving information from a swarm of robots, and displaying this to a user in a "*world-embedded*" manner using augmented reality techniques. One of the stated aims of this work was the use of minimal communication bandwidth. The authors present [23] an infra-red (IR) LED based solution for both indicating robot position, and communicating a small amount of information, in the form of coded pulse sequences, simultaneously. The HMD worn by the user decodes this information and superimposes a graphical representation on top of the user's view of the robots using a see-through optical display. Communicating via IR LED directly to the user's HMD removes the need for communication infrastructure, however it requires the user to have a direct line-of-sight to the robots in order to receive information. The data rate that can be achieved via this method is also heavily limited. However in the use case demonstrated by the authors [23], involving the display of a gradient in the direction of a target known to the swarm, the system is effective. The hardware requirements for this system are quite specific, and unlikely to be available on most robotics platforms, however the power of this kind of spatially situated or world-embedded data visualisation is clear.

Ghiringhelli et al. [24] present a system for augmenting a video feed of an environment containing a number of robots with real time information obtained from each of the robots. This is similar in concept to the system described by Collet and MacDonald [14], but is designed specifically to target a multi-robot system. The authors identify the ability to overlay spatial information exposed by the robots on to the video feed in real time, in the form of situated graphical representations, as the most important debugging feature of the system. Figure 2.8 shows a spatially situated overlay of data exposed by robot thirty two, in the authors' system [24]. A viewer is able to immediately verify the validity of the robot's world view from this image by comparing the blue overlay to the image beneath. Each robot features a coloured LED blinking a unique coded pattern to enable tracking, and the system uses homography techniques to map between each robot's frame of reference and the camera's [24]. The author's system therefore requires hardware to be present on the robots to support the tracking method. The homography technique also requires a reference frame to be established on the floor of the robot space.

The system developed in this project uses a simpler approach, with position and orientation tracking achieved through the use of the ArUco [25] marker-based tracking system. The camera is positioned with a bird's-eye view of the space, to simplify mapping by effectively reducing the 3D space to a 2D approximation. The tracking LEDs used by the authors' system [24] are active components, and therefore require specific hardware to be available on the robots in order for them to be tracked. This limits the system's generalisability. In contrast, using a passive, marker based technique means that no specific hardware is required, and the tracking could in theory be performed on any robot to which a marker can be attached.

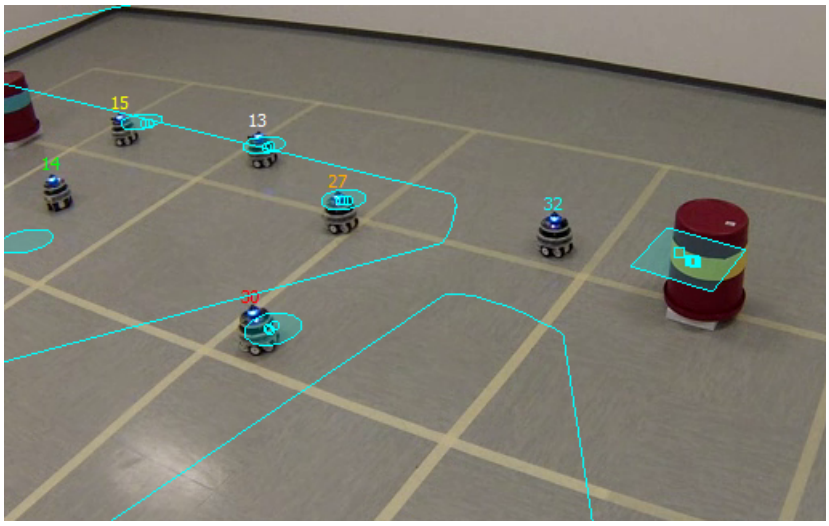


FIGURE 2.8: Example of spatially situated data, overlayed on a live image in [24].

The authors [24] also note the inclusion of a number of interaction modalities which allow the user to manipulate the visibility and display of specific elements of the data, which is grouped according to its associated robot and its type. This allows a user to focus only on pertinent information by hiding other information which is irrelevant. This follows the guidelines established in Rule and Forlizzi's work [8] on HSI, and the requirements of users regarding information salience and display when interacting with multi robot systems. Following this precedent, display configuration was identified as a priority and implemented within the system described in this project. This project also looks at expanding the methods used to display data to include more standard interface elements not directly related to the video image, as previously seen in [13], which is not addressed in [24]. Overall Ghiringhelli et al. [24] demonstrate how effective augmented reality-based tools can be for debugging a multi-robot system, and this project attempts to build on this success by providing a more generalised tool, which can meet the requirements of a large number of swarm-robotics systems.

2.7 Summary

The system developed during this project seeks to combine a number of ideas and techniques from existing systems and previous work to create a tool which specifically benefits swarm robotics development, but is generalisable in terms of target platform and required hardware. The system aims to collect and display swarm data in a central GUI similar to [13], in the spatially situated, AR-based manner of [14], [23], and [24]. The system is generalised by avoiding the specific hardware requirements of [13] and [14], which tie those systems to specific platforms, and by

using communication technologies which are more standard and more commonly understood than in [23], and more flexible and information rich than in [11]. The use of a standardised, defined data format helps avoid some of the code synchronization issues of [13], and aids in keeping the system general. Specific concepts regarding the display of data, as described in [8] and applied in [24] are also considered and applied within this system. Overall this project aims to help further efforts to improve human-swarm interaction, and add to the available set of swarm debugging tools one that is easy to understand, simple to extend, and well generalised such that it might be ported to different robot platforms in future. It is hoped that the software developed will be useful within the work of the YRL, as well as having scope for extension and use by the wider swarm robotics community.

Chapter 3

Problem Analysis

This chapter focuses on analysing the problem domain, including details of an initial survey conducted prior to the system's implementation regarding its potential feature set, and information about the two key hardware aspects of the system; the target '*e-puck*' robot platform, and the tracking camera. The chapter finishes with details of the software development tools used during the project, and explains the reasoning behind their choice over a number of other, similar tools.

3.1 Initial User Survey

A survey was carried out within the YRL in order to determine how best to implement the system, and to ensure that the system would be useful in practice once finished. The respondents were all actively engaged in robotics work, either in a research capacity or as a technician, and had experience and understanding of swarm robotics specifically. The survey aimed to determine if a level of interest existed for the proposed system, and then ascertain which specific features were most desired. This would go on to influence design choices and inform priorities during development.

3.1.1 Questions and Response Data

Question 1: Do you believe that a system for displaying internal robot data for a swarm of robots in real time would be useful when debugging swarm robotics behaviours and/or conducting swarm robotics experiments?

Answer	Votes	Percentage
Yes	4	80
No	1	20
No Opinion	0	0

TABLE 3.1: The responses to question one of the initial survey.

Question 2: Do you believe that such a system would benefit from the inclusion of an 'augmented reality' component - whereby data retrieved from the robots could be displayed in graphical forms, overlaid on a live video feed of the robots themselves?

Answer	Votes	Percentage
Yes	5	100
No	0	0
No Opinion	0	0

TABLE 3.2: The responses to question two of the initial survey.

Question 3: Please rate each of the following potential features based on your opinion of their importance or usefulness to the system proposed, on a scale of one to five, where one indicates a feature is not important or useful, and five indicates a feature is very important or useful.

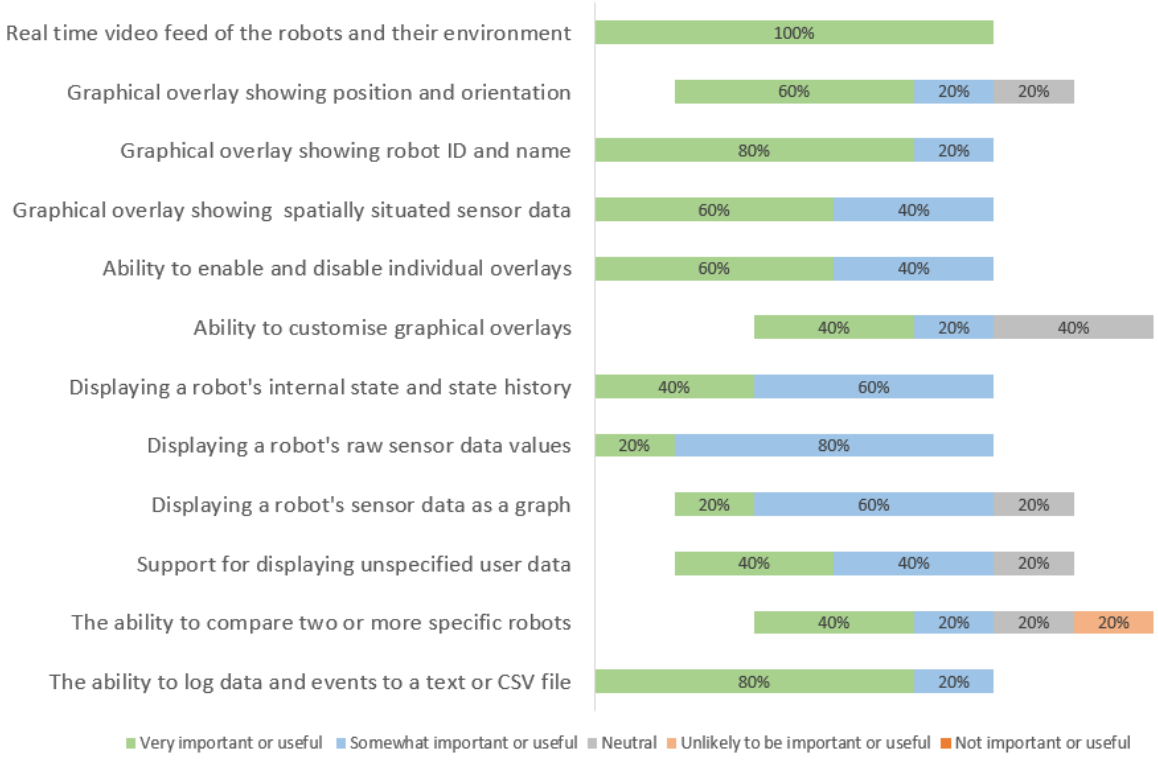


FIGURE 3.1: The responses to question three of the initial survey.

Question 4: Please briefly describe any additional features you believe would be useful, based on your experiences working with swarm robotics systems.

- "*macro-level behavioural data on the swarm; e.g., number of robots in different behavioural states (color coded accordingly).*"
- "*I think the system (already potentially very useful) could be improved/expanded further to add the option of more post-processing/extraction of data. The ability to create video of the over-lay, and perform statistical analysis on the complete run, would begin to turn the system in not only a very useful debugging system but a complete package allow researchers to gather high-quality, publication ready analysis of swarm experiments.*"

Question 5: Which of the following aspects of the robot data do you believe should be made available by the application to aid in debugging and testing? Tick all that apply. Please add any more you may think of.

Data Type	**Votes**
Position	4
Orientation / direction	4
Position change over time (recent path tracking)	4
Internal state machine state	3
Internal state transition history	3
IR sensor values	4
Distance between robots	3
Robot ID	4
Other	2

TABLE 3.3: The responses to question five of the initial survey.

Additional responses:

- "*Option to include user-defined data so if a certain controller implements a timer that facilitates a state transition might be useful to see the value of that timer whilst debugging to compare with state transitions.*"
- "*A simple API to add user-defined variables/statuses.*"

Please add any additional comments you have about the proposed application in relation to your experiences working with swarm robotics systems.

- "*A client-server model between back-end (camera) and remote client would potentially be very useful; also the system should be flexible and not reliant on a specific camera/server setup. Would be very interesting to see if it will work on a R-Pi 3 + camera combination, as this would allow for portable tracking setups.*"
- "*All real-time information is useful!*"

3.1.2 Analysis

The positive response to question one indicates a reasonable level of interest in the system amongst those surveyed. The similarly positive response to question two adds more weight to the idea that graphical debugging tools have the potential to be particularly useful in a robotics context, as established during the literature review. The responses to these two questions indicate that interest exists for the system, and that it is worthwhile implementing it. This satisfies the first aim of the survey.

The remainder of the survey focuses on establishing which features are most desirable, and the results were used when considering implementation priorities. The response to question three indicates that the majority of the core features were thought to be potentially useful, especially those related to the video feed and overlay. Tertiary features such as customising the colours and sizes of the overlays showed less interest. This was as expected, as these kinds of features do not aid directly in the debugging process. Considerable interest was expressed in the ability to log data and events, a feature which was not considered a major priority at the outset. Conversely, the ability to compare two robots was the only feature to receive a vote lower than neutral, despite being initially thought a key feature of the system. As a result the implementation of logging was moved up to a main priority, and the comparison feature was reduced to non-essential.

The respondents were then given the chance to optionally suggest additional features in question four. The first of the two responses suggests 'macro-level behavioural data', a concept considered during the project's inception. It was not included in the initial plan or survey partly because at the outset it was not clear what form the feature would take, or whether it would be feasible in the time frame, and partly because it was seen as a feature related more to analysing swarm experiments and results rather than debugging. As a result of this answer displaying macro level swarm data was considered a desirable but not essential feature, to be implemented if time allowed. The second response offers a broader vision for the system as a whole. This report agrees with the observation of the system's potential to become a complete package for analysing swarm experiments and extracting data, however the majority of the features mentioned are considered beyond the scope of this project. This includes video extraction and post processing of data. The expansion of the system is discussed in chapter 9 where possible future work is discussed.

Question five attempts to establish which specific data types are most desirable in the system. Note that one respondent did not answer this question at all, leading to the lower vote counts. None of the data elements listed received a significant deficit in votes, suggesting that all the data types listed should be included. Respondents were asked to optionally suggest other data types, and the two responses

received both independently identified user-defined data or 'variables' as a desirable data type. The inclusion of custom user defined data was a planned part of the system from the beginning, and is mentioned in question 3, but these responses confirmed that it should be a priority feature of the system.

The final section gave respondents the opportunity to add any additional comments about the system. The first response notes that designing the system in a way which does not make it 'reliant on a specific camera/server setup' would improve its usability. This thinking matches the stated objective of making the system in a modular way that is easily extensible with different camera set-ups and different robots.

3.1.3 Issues and Shortcomings

This survey provided a useful tool for gaining a general impression of relevant opinions regarding the project and the system. However it also had a number of issues in its execution which might somewhat diminish the value of the data. Firstly due to the highly specific nature of the desired participants, the sample size is extremely small. Care must therefore be taken in analysing the results too deeply; for example any statistical analysis performed would likely be flawed. Another issue is that the two larger multiple choice questions present a relatively small selection of possible features and data types, based on those already planned or considered for implementation. This report therefore aims to use the results in an indicative, holistic manner to guide implementation, rather than quantitatively define the feature set.

3.2 Hardware - E-Puck Robot Platform

The e-puck robotic platform, created by a team at the *Ecole Polytechnique Fédérale de Lausanne* [2], is a small, relatively inexpensive, multi-purpose robotic platform designed for education and research pursuits in robotics and multi-robot systems. The platform is widely used in swarm robotics research, featuring in a number of publications. The e-puck was chosen as the first target platform for this system for a number of reasons. Firstly this was one of the platforms available in suitable numbers in the YRL, where the practical work for this project was carried out. Secondly the platform's wide use in swarm robotics research helps to show the broad applicability of the system, and better demonstrates its value when compared to a less widely used or bespoke platform. Finally the platform's extensible design meant that it could be equipped with an extension board featuring an ARM9 processor running a Linux operating system, a configuration frequently used at the YRL. This enabled the use of WiFi for wireless data transfer, and was a large part of the reasoning behind choosing the e-puck. This section provides details of the e-puck robot's

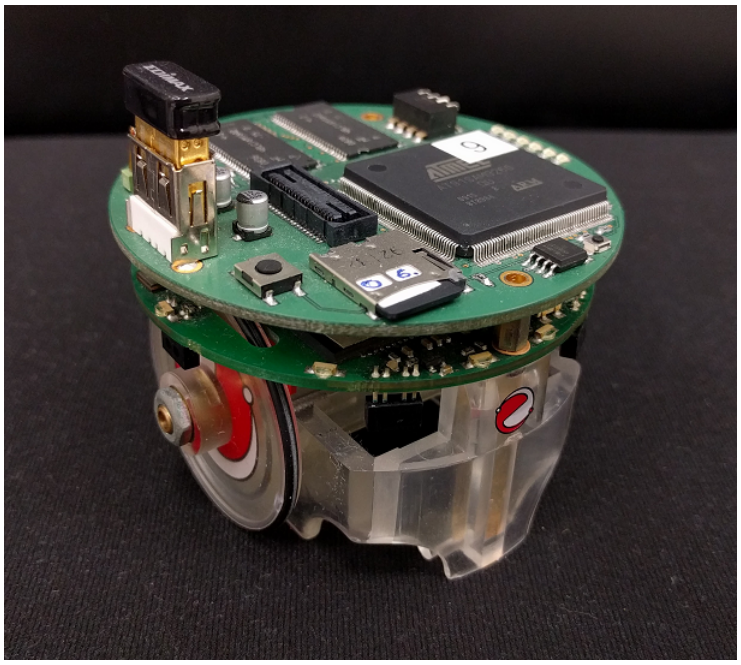


FIGURE 3.2: The e-puck robot, equipped with a Linux extension board.

hardware, including its sensors and actuators, as well as details of its twin-processor set-up and controller code architecture. Figure 3.2 shows the e-puck robot, equipped with the Linux extension board on top.

3.2.1 Actuators

The e-puck robot features two wheels, independently actuated by two step motors. The wheels have a diameter of approximately 41mm. The motors can rotate the wheels at an approximate maximum speed of 1000 steps per second in either direction, where 1000 steps is one full revolution. The robot also features a ring of 8 red LEDs around the edge of the main circuit board, which can be controlled for any arbitrary purpose.

3.2.2 Sensors

The e-puck robot features a number of different sensor sets, of which only some are used in this project. A set of 8 IR proximity sensors are arranged around the circumference of the robot, with four positioned on the forward hemisphere, two positioned at right angles to the forward direction (one on either side) and two more positioned on the backward hemisphere at roughly 45 degrees either side of the backward direction. Figure 3.3 shows this layout.

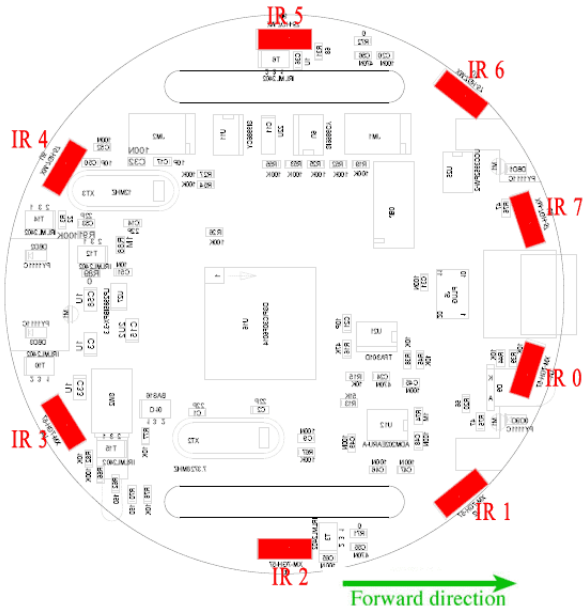


FIGURE 3.3: The layout of the IR sensors on the e-puck robot.

The IR sensors can be used in two modes - active and passive. In active mode the sensor emits an IR pulse and measures the strength of the reflection, whereas in passive mode the sensor simply samples the IR strength without emitting a pulse. The passive mode can therefore be used to get a 'background' IR reading, which can be compared to the active reading to improve accuracy. The IR sensor is of particular interest to this project as it is a frequently used tool when working with robots, especially robot swarms. The robot also features three microphones, a 3 axis accelerometer and a camera. Due to the bandwidth required to use the microphones and camera they were considered a low priority for this project.

3.2.3 Processors and Code Architecture

The standard e-puck features a *dsPIC 30F6014A* microprocessor, designed and manufactured by Microchip Technology Inc. This is a general purpose 16-bit CPU, with relatively low performance by modern standards. The processor features 68 I/O pins, which are connected to the e-puck's various peripherals. For this project the PIC processor was configured to act as a simple hardware controller. In this role the PIC receives commands via a UART serial interface, activates any relevant sensors or actuators, and returns sensor data also via the UART. Prior to the start of this project members of the YRL had already developed a library of low level code allowing the PIC to function in this hardware interface role, controlled through a UART serial connection. This firmware code was used as-is on the e-pucks' PIC processors throughout the project.

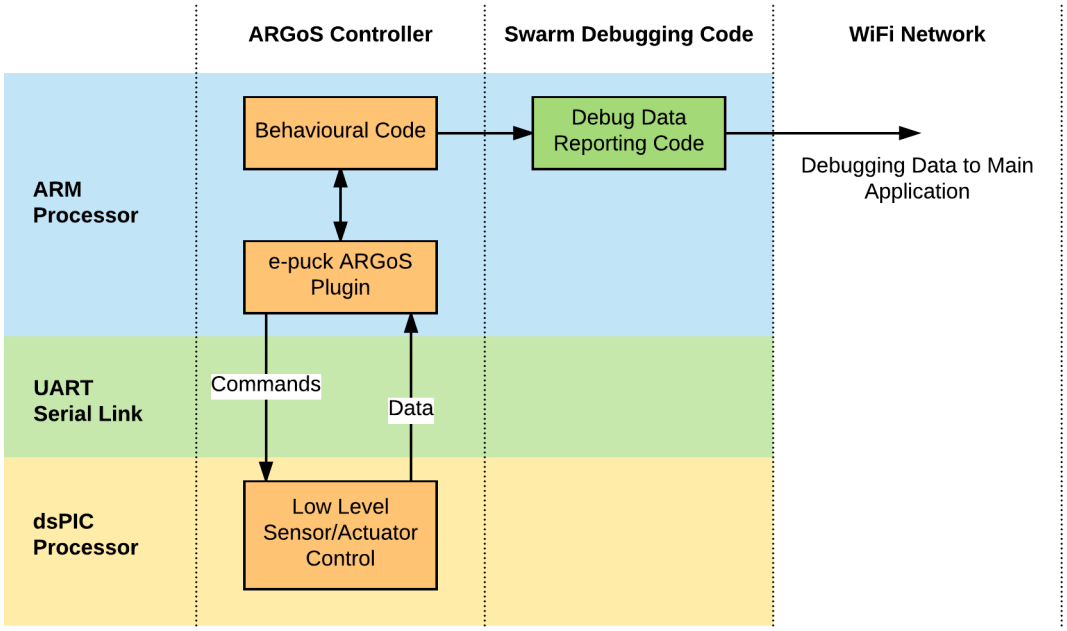


FIGURE 3.4: The architecture of the various code components running on the e-puck, organised by hardware layer and controller partition.

For this project the e-pucks were fitted with an extension board featuring a 32-bit ARM9 processor running a modified Linux operating system, developed by Wenguo Liu, and Alan F.T. Winfield [26]. In this configuration the ARM processor, an Atmel AT91, takes charge of the high level robot control logic, as well as any intensive data processing operations. When an actuator or sensor needs to be used, commands are sent to the PIC processor via the UART serial interface. The extension board also provides a USB port, and for this project a WiFi adapter was connected to each robot. The controller code running on each robot could then make use of the standard IP network layer protocol, and the standard transport protocols TCP and UDP.

The code for running robot behaviours on the ARM processor, and for communicating with the PIC processor, exists as a plug-in for the *ARGoS* robot simulator [27]. The ARGoS simulator is an exceptionally complex physics-based simulator for multi-robot and swarm robotics systems, and an in-depth explanation of its many features is beyond the scope of this report. Of relevance to this report is the fact that the ARGoS framework allows behavioural code for a robot to be implemented once, and then run both within the simulator and on the real robot. This is achieved by defining each robot as a plug-in for the ARGoS system, with two sets of the low level code to control the robot's sensors and actuators; one for use within the simulation and one for use on the real hardware. Behavioural code can therefore be the same for both situations, with the correct low-level code being selected at compile time.

The code for communicating data from the robot to the debugging system developed in this project was designed to be as portable as possible. For this reason it was kept fully independent of the ARGoS framework, and implemented as a stand-alone API. Calls to this API were then added to the robot's ARGoS controller code at the behavioural level. Figure 3.4 shows the various layers and components of the overall e-puck architecture. The debug data reporting code block, implemented during this project, is highlighted in green. Existing code components are highlighted in orange.

3.3 Hardware - Video Tracking System

In order to implement the augmented reality element of the system, and satisfy the related objectives, a live video feed of the swarm was needed. A method for tracking the positions of each individual robot in the swarm within each video frame in this feed was also required, in order to correctly position the graphical overlays. Prior to the start of this project infrastructure had been put in place at the YRL for performing this kind of video-based tracking task, in the form of a machine vision camera placed above a robot '*arena*'. Figure 3.5 shows the arrangement of the machine vision camera and the robot arena. Figure 3.6 shows a photo of this set up at the YRL. Ongoing research being carried out by members of the YRL has already made use of this camera and arena set up, in conjunction with image processing software for tracking fiducial markers within the image, to achieve robot tracking with good accuracy and relatively high frame-rate.

The image processing in these ongoing research efforts was performed using the 'ArUco[25]' fiducial marker based tracking system, discussed further in section 3.3.2. It was determined that incorporating this existing infrastructure into the swarm debugging system would be the quickest way to achieve an operational tracking system, allowing work to focus on the novel aspects of the project sooner.

3.3.1 Camera

The camera used in the aforementioned tracking set-up is a JAI Go 5000C-PGE colour, area-scan camera, shown in figure 3.7. It features a 1-inch, 5-megapixel CMOS sensor, with a maximum resolution of 2560 by 2048 pixels, capable of capturing 22 frames per second at full resolution, and up to 163.5 frames per second with reduced resolution and colour quality. The camera also features a global shutter, meaning it captures the full area of the image simultaneously, as opposed to a rolling shutter which captures portions of the image sequentially. This camera is well suited to marker tracking applications for a number of reasons. The high resolution means that even small markers, or markers that are relatively far from the

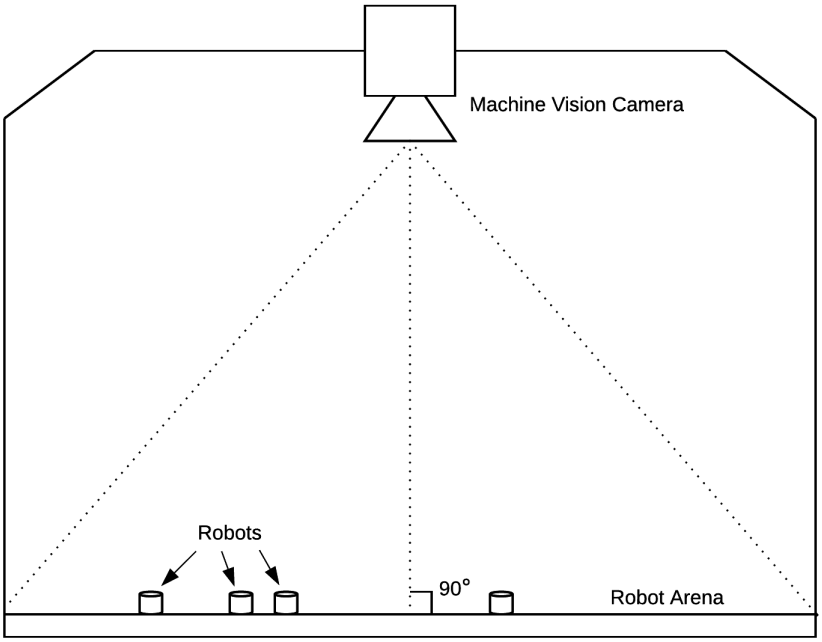


FIGURE 3.5: Arrangement of the tracking camera over the robot arena.

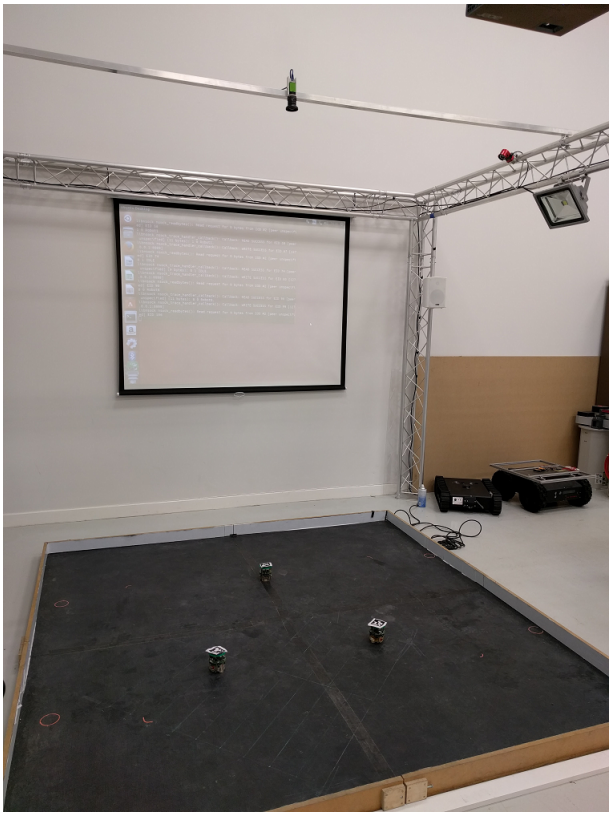


FIGURE 3.6: The robot arena set up at the YRL, with tracking camera visible.

FIGURE 3.7: The JAI-Go 5000C-PGE camera [28].

camera will be captured with sufficient detail to be tracked and identified. The relatively high frame-rate means that the tracked positions can be recalculated regularly, and the video will appear smooth. The use of a global shutter avoids issues present in rolling shutter cameras, where a moving marker appears broken due to the time difference between the capture of two areas of the image and the movement of the marker during this time.

The camera is connected to a server rack using Gigabit Ethernet cable, which is required to support the high bandwidth output of the camera due to its resolution and frame rate. This cable also provides power to the camera. A high resolution lens is fitted, with a relatively wide range of possible focal ratios from $f/1.4$ to $f/16$, allowing the camera to be adjusted to work well in a range of light conditions. With the lens included the camera has approximate dimensions of $29 \times 29 \times 100$ mm, and a weight of 246 grams, making it a very compact, lightweight solution.

The image data from the camera is transmitted in accordance with the '*GigE Vision*' interface standard [29]. This standard was first introduced in 2006, and is designed for transmitting video from high-performance industrial cameras over Ethernet networks. The 'Common Vision Blox' (CVB) machine vision programming library is used to map the data from the GigE format to the OpenCV image format, so that it can be processed by an application.

3.3.2 ArUco Tracking System

Developed by a team from the Computing and Numerical Analysis Department of Cordoba University in Spain, the ArUco tag generation and detection system [25] is a powerful fiducial marker creation and tracking tool. It comprises an algorithm for

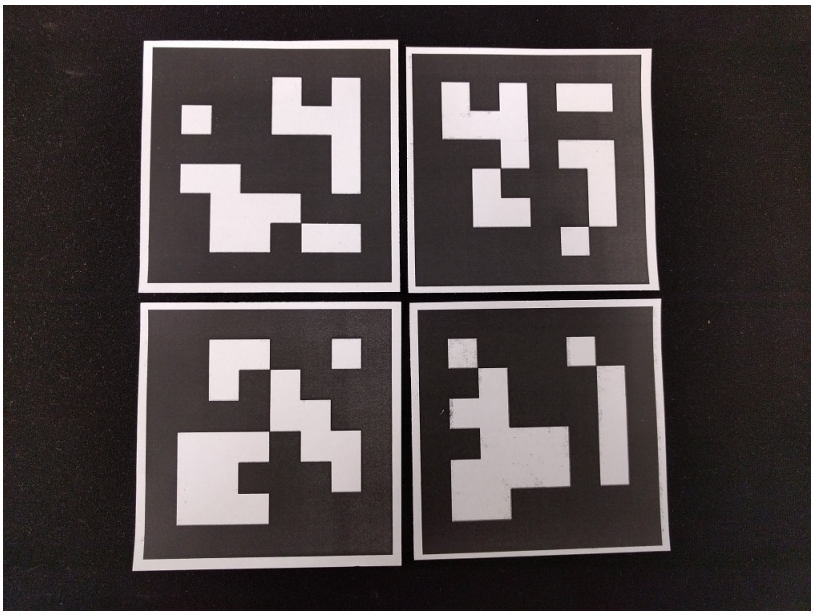


FIGURE 3.8: Four markers generated by the ArUco fiducial marker generation and detection system [25], and printed using a standard printer.

producing a dictionary of square, black and white, coded markers, and further algorithms for automatically detecting these markers in a given image. These markers can be easily printed using a standard printer, and attached to surfaces and objects. Figure 3.8 shows four possible marker variations on standard printer paper. The stated applications for the ArUco system include augmented reality, machine vision, and robot localisation.

One of the main benefits of this system over other fiducial marker systems is the execution speed. By first using edge-detection methods to isolate the outlines of potential markers in the image, the system can eliminate a large portion of the image before applying the more complex processing to identify and differentiate individual markers [25]. This makes the algorithm extremely efficient, and therefore makes it possible for the ArUco system to be run in real time, even with relatively modest computational power. In a conventional use case the orientation of the camera can be calculated based on the positions of the corners of a marker, if the marker's orientation is known. In the use case of this project the reverse is true; the camera's position is fixed, and therefore the orientation of a robot can be determined based on the position and orientation of the corners of its marker.

Each of the e-puck robots used in this project were assigned an ID number, and a dictionary of ArUco markers was generated to match. The markers were affixed to the top of the robots, oriented to match their forward directions. Figure 3.9 shows a laser printed ArUco marker mounted on top of an e-puck robot. Some cursory preliminary tests of the tracking system confirmed that the markers could be accurately

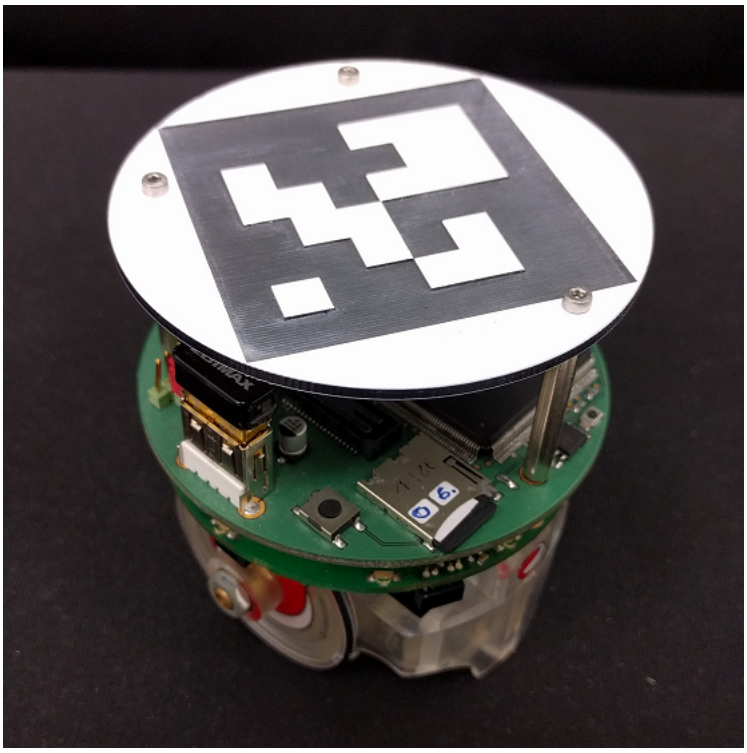


FIGURE 3.9: A laser printed ArUco marker mounted on top of an e-puck robot.

and reliably detected in the camera images, at a decent frame rate. Further detail of the integration of the ArUco marker tracking system into the application developed during this project is given in section 6.6.

3.4 Software Development Tools

Throughout the implementation process a number of tools were used to enable and organise development. Software development is a tool-rich field, and selecting the right tools, and using them effectively, can result in significant increases in development efficiency.

3.4.1 *Git* Version Control System

Version control is an important concept in software development [30], and a variety of tools exist to help developers manage the process. The purpose of a version control system (VCS) is to handle additions and changes to a code-base in an organised, structured fashion, such that each new change is logged, and can be reversed if necessary. This means that if some feature or functionality is broken by a change, it is trivially easy to revert the code-base to its state before the change in order to

recover the functionality. It also helps with debugging by allowing a developer to examine the specific contents of a change that has caused a problem, without relying on memory. Version control also enables multiple developers to work on the same code-base simultaneously [31], as it automatically checks for collisions between new changes.

Most version control systems are based around a code '*repository*', which stores the code, as well as information regarding every change made. In a standard distributed model developers download a copy (or '*clone)*' of the code from this repository to their development machine. Details of changes made to their local copy are stored in '*commits*', which allows the developer to add a message to describe each change. These commits can then be uploaded (or '*pushed*') to the main repository. These repositories are usually hosted online or on a private server, allowing developers to work in a distributed fashion, and removing the risk of significant code loss if one development machine is lost for any reason.

For this project the *Git* version control system [32] was chosen, in conjunction with the *Github* repository hosting service [33]. Github is a widely used online software repository hosting service which supports all of the functionality of the Git VCS, as well as a number of other convenience features including analytical and project management tools. Github and Git were chosen for this project for several reasons. Firstly, both are widely known and used in the software development world, suggesting reliable and well developed functionality. This combination is also regularly used within the YRL for other software development. The Github service is also free to use, thus avoiding any cost to the project. Finally previous experience with both the Git VCS and the Github repository hosting service had been positive, and suggested they would be suitable for this project.

A wide variety of VCSs exist, and a number were considered for use in this project. '*Subversion*', commonly referred to as just '*svn*', is another VCS with similarly widespread usage to Git [34]. However svn functions in a slightly different manner, requiring a connection to the server where the repository is hosted before any commits can be made. Git does not have this requirement, as commits can be made against the local copy of the repository, and later pushed to the main repository when a connection is available [32], allowing the benefits of version control to be leveraged without a connection to the server. This was the primary reason Git was chosen over svn, as the development work for this project was to be completed on a laptop in a number of locations, without a guaranteed internet connection at all times.

'*Mercurial*' [35] is another VCS with a similar feature set to Git. Like Git, Mercurial allows for commits against a local repository clone, removing the need for a constant server connection during development. For a project with development requirements as simple as this one, the slight differences between Git and Mercurial

are unlikely to be relevant, hence the two were assessed to be essentially equivalent. It was therefore simply prior experience with Git, and the use of Git within the YRL, that led to its choice over Mercurial.

Similarly to VCSs, there are many repository hosting services available today, all with relatively similar feature sets and minor differentiating factors. '*Bitbucket*' is another widley used, free, repository hosting service which supports the Git VCS. The service supports virtually all of the same features as Github, and would have been an equally suitable choice for this project. It was not chosen primarily due to the prevalent use of Github within the YRL, and due to prior experience with Github meaning less of a learning curve and therefore a shorter start up time for the project. Similarly other repository hosting services were ruled out based on the convenience of Github due to prior use. Another relevant factor was cost, as a number of other similar services, such as '*Kiln*', require a paid subscription.

This was a single developer project, and therefore many of Git's team-based features were not necessary, and the tool was used simply for code storage and version control. Development was carried out on a single development branch, as there was no need to maintain a stable release branch, and no benefit to multiple feature branches given the lack of multiple developers. Once development was completed at the end of the project, the development branch was merged back on to the main branch, marking the completion of the first version of the software.

3.4.2 *Qt Creator* Integrated Development Environment

An integrated development environment (IDE) is a software tool used when developing other software, providing basic functionalities such as text editing and file browsing, as well as specific features to help with development, such as syntax highlighting, error checking, build configuration management, built-in compiling, and debugging tools. IDEs are extremely useful tools when developing any piece of software that is non-trivial in size or scope, or has significant dependencies, and serve to accelerate work-flow and minimise issues.

Section 6.2 discusses the choice of the *Qt* application programming framework for implementing the swarm debugging application within this project. *Qt Creator* [36] is an IDE specifically designed to aid in the development of software applications based on the Qt application framework, making it the obvious choice for use in this project. Qt Creator offers a number of features to specifically support the Qt framework, including a graphical interface layout designer tool, which was used to compose the graphical interface for the application, and produced the *mainwindow.ui* XML-like file which describes the application's UI layout. The IDE also provides integrated support for the *QMake* build management system, which describes the files necessary to compile and build the application. The application could be compiled

and run from within the IDE, which also features a debugger and supports standard tools such as breakpoints. The debugger was used regularly during development to locate issues and refine the implementation of features.

A number of other IDEs were considered for use during application development, including several which are more widely used and feature-rich than Qt Creator. 'CLion' [37], a modern, feature rich IDE designed for C and C++ development was considered due to its powerful code analysis and suggestion feature set, cross platform support, and free student license. However ultimately it was determined that the benefits of Qt Creator's built-in support for Qt applications outweighed the benefit of CLion's superior feature set. CLion was considered again when choosing a tool for writing code to be run on the robots, separate from the main application code, however it quickly became apparent that, due to the relatively simple nature and scope of this portion of the code, the features of a full IDE would not be necessary. The '*Sublime Text*' text editor [38] was chosen instead, as this simple tool required minimal set up, but still supported useful development features including syntax highlighting.

3.5 Summary

In this chapter analyses of a number of topics relevant to the project were presented. An initial user survey, carried out prior to the design and implementation of the system, was described and the results presented and discussed. This survey sought responses from potential users of the system, and in general showed an appetite for the system and many of its core features, but highlighted some planned features as less relevant than originally thought. Relevant infrastructure within the YRL was discussed, including the *e-puck* robotic platform, which was the initial target platform for the system, as well as the camera hardware configuration, and the *ArUco* tracking system. Finally the choices of software development tools were discussed, explaining the choice of *Git* and *Github* for version control and code repository hosting, and the *Qt Creator* IDE and *Sublime Text* text editor for code editing and development work. These analyses led to a better understanding of the problem domain, and allowed for the system to be designed and implemented effectively.

Chapter 4

Project Plan

This project was completed between Monday 16th January and Thursday 18th May, 2017. A well-defined breakdown of the tasks required to complete the project, and an organised plan for completing these tasks, was instrumental in ensuring that the project could be completed in the available time. This chapter gives details of this work breakdown, and the timing considerations. It also includes details of a number of potential risks identified at the start of the project, and information on how these were mitigated where possible.

It should be noted that a significant majority of work involved in this project was software development based, and - as with almost all modern software development - accurately predicting the time required to implement every piece of code was a virtually impossible task. The development process inevitably leads to the discovery of unforeseen issues and a deeper understanding of the problem constraints, which in turn requires the allocation of project time to be re-evaluated. Hence wherever possible an '*agile*' methodology and approach was employed. This included frequent re-assessment of the remaining work and the feasibility of individual features. The agile methodology, and its application within this project, is discussed in more detail in section 4.4.

4.1 Work Breakdown

At the start of the project the software related work, including both development and testing stages, was divided into logical tasks. These tasks are shown in tables 4.1 and 4.2 respectively. The time required to complete each task was estimated based on prior experience of software development work, and these approximate timings are listed in the tables. Other tasks, not related specifically to the software development, are listed in table 4.3.

TABLE 4.1: Development tasks.

Task	**Objective**	**Approximate Time**
Read and Understand Existing Code	To understand existing code related to the tracking camera and networking on the e-pucks.	14 Days (Alongside other development)
Establish Development Environment and Toolchain	To enable organised development by establishing a tool set and workflow.	3 Days
Learn to Re-Program e-puck Robots	To understand the cross compilation process for the e-pucks.	2 Days
Outline Software Architecture	To design a coherent code structure in order for code to remain organised and modular.	2 Days
Design General User Interface	To create a high level design of the basic UI and implement a skeleton framework of this UI.	3 Days
Incorporate Tracking Camera Code	To incorporate existing low-level code for acquiring images from the tracking camera and performing tag detection.	2 Days
Implement Tracking Camera Controller	To implement code to create a layer of abstraction between the application code and the tracking code.	2 Days
Implement Wireless Data Receive	To implement code to allow the application to receive data wirelessly.	3 Days
Determine Robot Data Types	To establish an initial set of data types that will be supported by default, and a packet format for these.	2 Days
Design and Implement Data Model	To design the back end data model of the application and implement it in code.	6 Days
Implement Mapping Received Data to Model	To implement code to store received robot and tracking data in the application data model.	3 days

Implement Basic Visualiser	To implement code for displaying the video feed and augmenting it with basic geometric primitives.	5 Days
Design UI Data Representation	To establish a design for the representation of the different data types.	2 Days
Implement Graphical and Textual Data Visualisation	To implement code to convert the data in the data model into relevant visualisations.	10 Days
Implement Data Visualisation Filtering	To implement code to allow the user to filter out unnecessary visualisation elements.	5 Days
Implement Robot Data Comparison	To implement code to allow the user to compare the data of specific robots.	3 Days

TABLE 4.2: Testing tasks.

Task	**Objective**	**Approximate Time**
Continuous Integration Testing	To continually test newly implemented features with the system as a whole during the implementation process.	Throughout development
Verification Testing	To verify the correct operation of the software through formal testing, with a specific focus on the data model and the user interface.	10 Days
Verification Fixes and Changes	To make the necessary changes to correct issues identified in the verification testing.	5 Days
Final Fixes and Changes	Some leeway time is available to make any final changes or fixes based on the results of the user evaluation sessions.	Remaining time

TABLE 4.3: Other tasks.

Task	**Objective**
Initial Report	To produce an initial report in the early stages of the project, outlining the preliminary research completed and the project plan at this stage.
Create Initial User Survey	To create a survey to be answered by potential users of the system such as robotics researchers to gauge interest levels for the proposed system and specific features.
Distribute Initial User Survey and Collate Results	To distribute the survey to swarm robotics researchers and others with relevant experience, and collate and analyse the responses.
Create a System Evaluation and User Testing Plan	To devise a plan for evaluating the effectiveness of the system, including a detailed description of the user testing procedure.
User Evaluation Sessions	To evaluate the system by allowing a number of users to use it in a structured evaluation session.

4.2 Timing and Plan

Having estimated the time required for each development task, a plan for completing the project within the available time frame was devised, and can be found as a Gantt chart in appendix B. The plan comprised three main phases; a preliminary phase of research and design work, the main implementation phase, and a final testing and evaluation phase. Where possible, tasks were organised to leave weekend days free, in an attempt to provide a more manageable schedule, but also to allow some slippage time each week for any overrunning tasks.

4.3 Risk Analysis and Mitigation

Engineering projects in all fields are subject to a wide range of risks which may prevent their successful completion. A common technique to reduce the chances of an unsuccessful project is to analyse the potential risks involved in the project before work begins, and attempt to mitigate these risks wherever possible. Risks with a high likelihood of occurring or a high impact on the project are prioritised for mitigation first. For this project the following risks were identified, and mitigation steps taken.

1. Failure to complete the work in the available time.

The primary source of risk within this project was time. Software development is an inherently time consuming process, with most implementation tasks suffering from a degree of uncertainty. The time frame for this project was also relatively short. Therefore a major risk to be considered was the potential for features not to be implemented due to a lack of available time, and for the system to fall short of its stated aims and objectives as a result.

A number of steps were taken to mitigate this risk. Firstly, the features of the system required to satisfy the core aims were assessed, and the task schedule was organised such that a '*minimum viable product*' (MVP) would be completed as quickly as possible. The MVP describes the smallest possible set of features which can be implemented such that the system still satisfies its core objectives.

The second step taken to mitigate the time-risk was the inclusion of slippage time in the project plan. This extra time was worked into the plan to absorb any overrun in individual tasks, reducing the chances of a single overrunning task having a knock-on effect on the rest of the plan. Slippage time was introduced in two places; by arranging tasks to leave weekend days free, as previously discussed, but also by planning for a flexible testing phase, where time previously allocated for testing could be re-allocated for development if this was deemed necessary. The knock-on affect of reduced testing was determined to be an undesirable but acceptable consequence, partially mitigated by the use of continuous integration testing.

2. Loss of development computer due to a hardware issue.

Another source of potential risk in this project was the development hardware. The majority of the development work was carried out on a personal laptop, running a Linux operating system in a virtual environment. Any damage to this computer could have led to the loss of important code, and interrupted the development process while a new machine was found. In order to mitigate the potential risk of code loss all code was stored in an online repository, and the latest code was uploaded to this repository frequently. The development environment required to compile the project code was also set up on the tracking server which would be running the final application. This was done so that the code could be tested on the target environment during development, but this also partially mitigated the timing risk involved in the loss of the main development machine, as work could be continued temporarily on the tracking server while a replacement machine was found and set up.

3. Loss of a key system component due to a hardware issue.

All electronic hardware is susceptible to breakage, faults and malfunctions. The system developed in this project includes a number of hardware elements working in tandem, and the loss of any single component could have halted the implementation and testing phases of the project until a replacement could be found. In the

case of the robots this risk was mitigated by the fact that a reasonable number of robots were available, and a broken or malfunctioning robot could be replaced without much disruption. This was especially true if the Linux extension board was still operating correctly, as this could easily be moved to another standard e-puck.

Mitigation by redundancy in this manner was however not an option for some of the larger, more expensive components of the system. At the time of the project the YRL possessed only one tracking camera. Had it suffered a fault or been damaged this would have disrupted the project greatly whilst it was fixed or replaced. However, due to the fact that the tracking camera required no moving parts, was not moved during the course of the project, and had shown no signs of unreliability in the past, the chances of a fault occurring or the camera being damaged were considered to be quite low. Therefore, although this was a potentially high-impact risk, its low-probability meant that it could be accepted without mitigation. The tracking server itself was considered more likely to suffer a fault, and this risk was partially mitigated by the fact that another server was available in the YRL that could potentially have served as a replacement.

4. Significant, un-addressed bugs or issues in the software.

Bugs are a risk inherent in all software development, and can cause 'finished' software to be unstable or functionally flawed if undetected or left unaddressed. In order to mitigate the chances of a major bug going undetected a thorough testing phase was included in the project plan. During this time testing methodologies were applied to the software in an attempt to detect and identify any bugs in the software. Any bugs discovered could then be fixed, or if a fix was not possible, mitigated in some way. The testing applied was based on established, proven methodologies, in an effort to ensure that bugs did not go undetected due to a poor testing strategy or poor test coverage.

4.4 Application of Agile Methodologies

'*Agile Software Development*' [39] is the term used to describe a variety of software development principles and practices, which focus on the continuous, incremental delivery of functional software, and the ability to rapidly adapt to changing requirements, among other things. A number of key principles from the agile development methodology were embraced during this project. The first was the idea that functional, working software should always be a priority, and should be the primary measure of progress. The software for this project was therefore developed incrementally, and was verified to still be in a usable state after the addition of each new element or feature. This also tied into the MVP approach discussed previously, as

after each incremental addition the software could undergo a cursory evaluation to determine if it yet satisfied the stated aims.

The second agile development principle that guided this project was the idea that reacting to changes in requirements or changes in the feasibility of features is more important than strict adherence to a plan. Whenever an issue arose that changed the estimated time required to implement a feature, or brought into question its feasibility, the necessity of implementing that feature was reconsidered in light of the new information. Features that were determined to be likely to require too much time when compared to their value were discarded in favour of other key features that could be realised more quickly. In practice relatively few issues arose during the development of this project, and most features were able to implemented as planned, within the original estimated time frames.

One further principle from the agile development methodology that also had an effect on the project was the principle of customer collaboration and continual requirement capture. Although this project did not have a customer in the traditional sense, the researchers within the YRL who responded to an initial survey regarding the software acted as potential customers, guiding the design and implementation of the system with their input. The priority of certain features were adjusted following their comments, with the hope that this would lead to a more useful and practical implementation of the system.

4.5 Summary

This chapter has presented the plan that was devised early in the project for completing the necessary work in a timely fashion. The plan featured three main phases; an analysis and design phase, an implementation phase, and a testing and evaluation phase. Each phase was then broken down into individual tasks necessary to satisfy the project requirements and functional specification, and a Gantt chart was generated based on the estimated time requirements of each task. Potential risks to the project were also presented and analysed. Hardware malfunction or loss was identified as a significant existential risk, however a number of mitigation techniques were employed, including redundancy hardware where possible. The other key risks - insufficient time, and bugs in the finished software - were to be mitigated through planning and adaptive scheduling, and thorough testing respectively. Finally a summary of the application of elements from the *Agile* software development methodology to this project was given, including a focus on functional software and incremental development, reactive planning, and frequent requirements capture.

Chapter 5

Design

This section describes the design of the system, and gives details of the reasoning behind some of the design decisions. This design work was done largely prior to implementation, with some elements of the design being re-factored during the implementation phase in adherence to the agile development methodology being followed. The design process was separated into a number of key areas. Firstly a software architecture design was produced, including a breakdown of the system in terms of individual components and a description of the intended path of data through the system. This served as a road map during the implementation stage. The second key area was the user interface design. This involved determining how the main application should appear to the user, and how best to provide the user with access to the various features, and then creating a template of the window and component layout required to achieve this. The graphical representations for each of the data types to be displayed in the visualiser were also designed.

5.1 Software Architecture Design

The guiding principles of the software architecture design were 'Object Oriented Programming' (OOP) practices, and the 'model-view-controller' (MVC) software architecture pattern. OOP [40] is an extremely widespread concept in modern software development theory. It states that code should be organised into units based on individual functionalities, commonly referred to as classes. Each class collects together the data that describes an object and the routines to perform actions with and on that data. A class then acts as a template, and a new instance of the class can be instantiated each time an object of that type is needed. OOP aims to make code easier to understand and maintain, reduce code duplication, and increase re-usability and modularity. Designing software in an OOP fashion is standard practice for most modern programming tasks, and modern languages are often designed around OOP concepts.

C++ was selected as the development language for this application as it offers much of the low level control and efficiency of the C language, whilst also supporting OOP practices natively. The majority of the existing software infrastructure in the YRL had also been implemented in C++, hence following suit would help with maintainability and integration in the future. Considering the project's requirements for interfacing with low level hardware such as the tracking camera (via a driver) and the robots (via network sockets), and for performing image processing, the speed and low level capabilities of C++ also seemed beneficial. Higher level languages such as Java were considered, as they offered a number of different benefits such as better portability and automatic resource management, but were ultimately deemed less suitable.

The MVC [41] software architecture design pattern is another widespread concept in software development theory. It primarily relates to the programming of application user interfaces. The three words that give the pattern its name - model, view and controller - define the three 'layers' into which code components are organised. The model refers to the application's data, and includes all of the information that defines the application's current state. The view refers to the code used to produce the user interface from the data in the model layer. It acts as the method by which the user 'views' the data. Finally the controller layer acts as the intermediary between the two, retrieving data from the model and processing it if necessary before passing it to the view for display. The controller also responds to inputs, and modifies the data in the model accordingly. This includes user input, as well as input from other sources. In the case of this application these other sources include the tracking camera and the robots, which send their input via the Ethernet cable and WiFi network respectively.

The MVC pattern specifies that all components must exist in one of these layers, and the layers must be kept separate. Adhering to an MVC pattern ensures that state data is not maintained or duplicated within the UI code, and that only one, true copy of the application data exists, stored in the model layer. Separating the functionalities in this way also helps to keep code structured and organised, making it easier to understand, maintain and extend, as changes to the internals of a component in one of the layers should not affect components in the other layers.

With these two principles in mind, the software design process could begin. First the application was broken down into individual components based on the functional specification. The following key components were identified:

- Code to handle communicating with the camera and retrieving the image data.
- Code to perform the robot tracking.
- Code to handle the networking, including receiving data from the robots.
- Code defining a model to store the robot data in.

- Code to enter new data into the model.
- Code to augment the video feed based on the stored data.
- Code to display the video and augmentations feed to the user.
- Code to respond to user input via the user interface.
- Code to store settings related to the application.

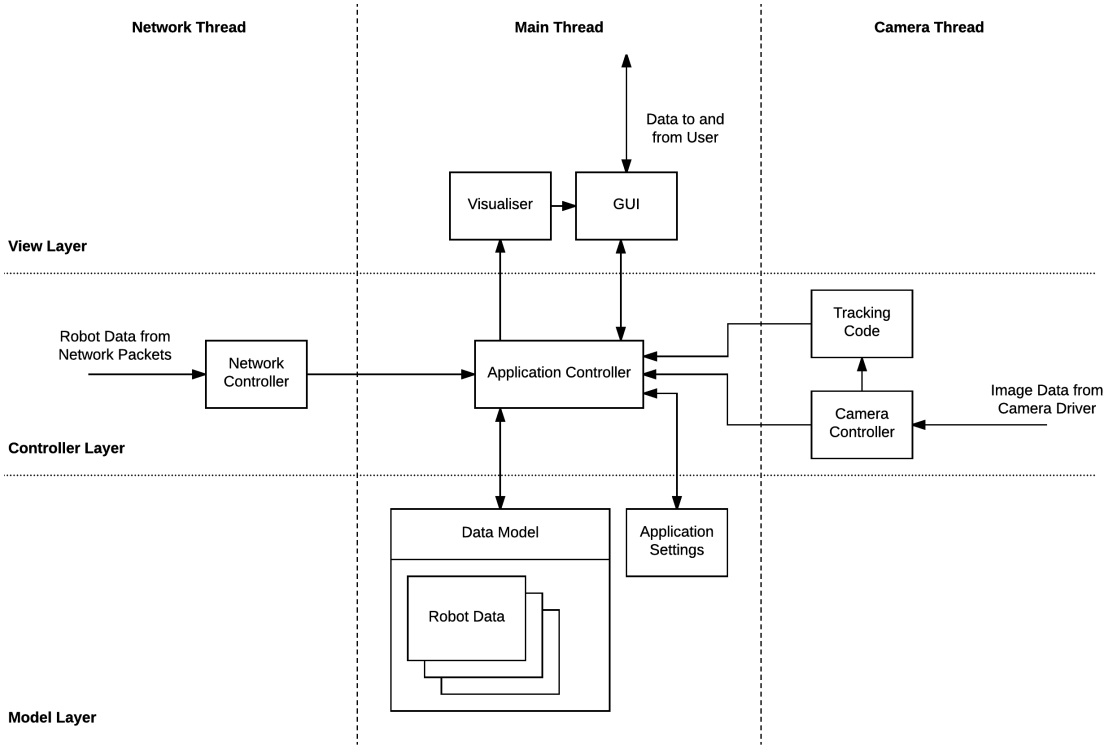


FIGURE 5.1: A diagram of the software architecture design and data flow.

Once separated into functional components, the required code blocks could be organised into a structure, and the data path of the application examined. Figure 5.1 shows this structural arrangement, with boxes for each of the functional objects and arrows indicating data flow. The three layers of the MVC design pattern are shown by the vertical partitioning. The horizontal partitioning is used to show another key design consideration - threading. In order to maximise performance and ensure responsiveness, functionality which has the potential to 'block' execution whilst waiting for a result or response should be run in a separate thread. This application was therefore designed with three threads in mind. The main thread handles the core of the application, including all data model access and GUI operations. The network thread handles communicating with the robots via WiFi. It was anticipated that this networking would involve low level socket code, which meant the potential for blocking socket-read operations, therefore requiring a separate thread. The camera thread handles reading the machine vision camera and performing the tag tracking

using the ArUco library. It was anticipated that the camera read operation could block until the next frame was available in the camera driver's buffer. Tracking the robots in the image using the ArUco library also had the potential to be CPU intensive, so keeping this off the main thread was considered a potential performance benefit.

As can be seen in figure 5.1, the software architecture is structured around a central application controller. The data from all other components of the application flows through this central component, which routes it to where it needs to go. It is also responsible for updating the visualiser and GUI with new data when necessary, and acting on the user input signals received through the UI. The other key controller layer components - the network controller and the camera controller - exhibit data flow in only one direction. The design of these components was therefore relatively simple, as they needed only to perform the following loop of tasks:

1. Receive data from external source.
2. Process data as necessary, extracting required information.
3. Notify the application controller of the newly available data.
4. Wait for new data to become available. Return to step 1.

The remaining components are more complex, and required greater software design considerations. The design of the data model is discussed in section 5.1.1. The design of the visualiser code is discussed in section 5.1.2.

5.1.1 Data Model

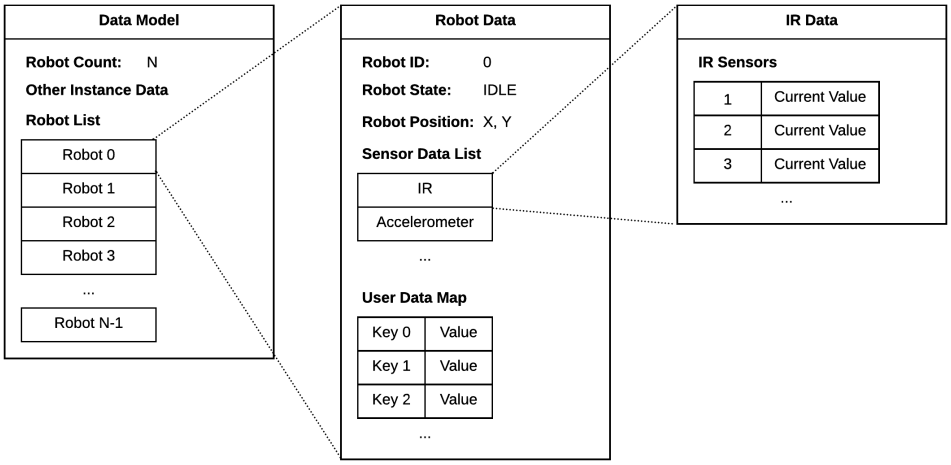


FIGURE 5.2: A diagram of the data model design.

The purpose of the data model component is to store all of the data required to describe the application's current state. This includes all data related to each of the robots being tracked by the system. In order to achieve this in an object oriented manner the data model itself was designed to comprise a number of smaller components in a hierarchical structure. The main data model object maintains a collection of smaller objects, each containing the data related to a single robot. These in turn maintain a collection of different data objects relating to the robot's state and sensor data. Figure 5.2 illustrates this hierarchical data model design using the IR sensor data of a single robot as an example.

This follows object oriented programming practices by defining a single class to describe a robot, from which a new robot data object can be instantiated each time the system begins tracking a new robot. When data is received regarding a robot the system is already aware of, it can simply update the correct existing robot data object with the new data.

5.1.2 Visualiser

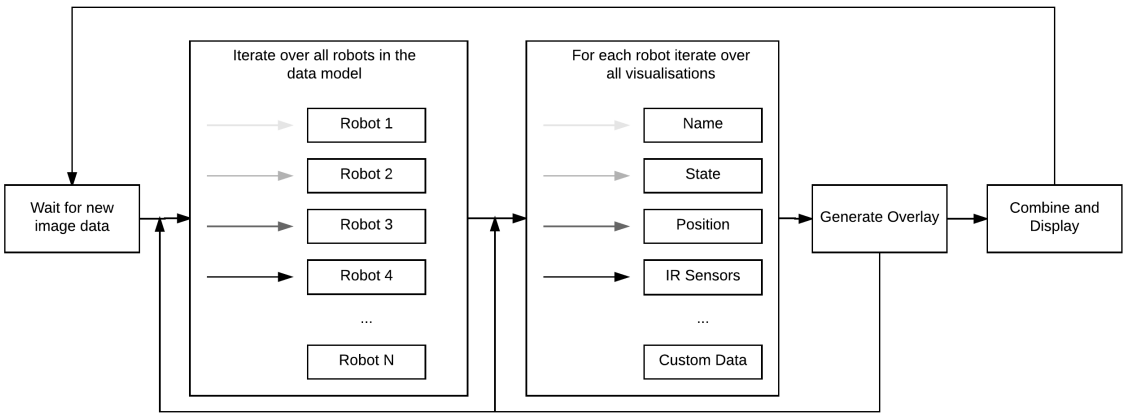


FIGURE 5.3: A diagram describing the design of the visualiser rendering process.

The purpose of the visualiser component is to generate and display the augmented video feed to the user. The component must receive the latest camera image and the most up to date robot data, and generate the graphical overlays based on a combination of the robot data and the current settings for each visualisation. It can then display the latest image, with the graphical overlays applied, to the user by rendering it as a single image within the appropriate GUI element. This rendering process must occur each time a new video frame is read from the camera, hence both the video and the overlays should update regularly and respond rapidly to changes in the robot data.

In order to further modularise the visualiser code, it was broken down into a number of smaller components. For each type of data visualisation a separate component was defined, which describes how the visualisation for that data type is generated, based on a single robot data object. The main visualiser component could then be designed to generate the graphical overlays iteratively, by stepping through the available set of robot data objects, and for each one stepping through the different visualisation components, generating an overlay for each combination. The overlays can then be combined and displayed as a single image. Figure 5.3 describes this process diagrammatically.

5.2 User Interface Design

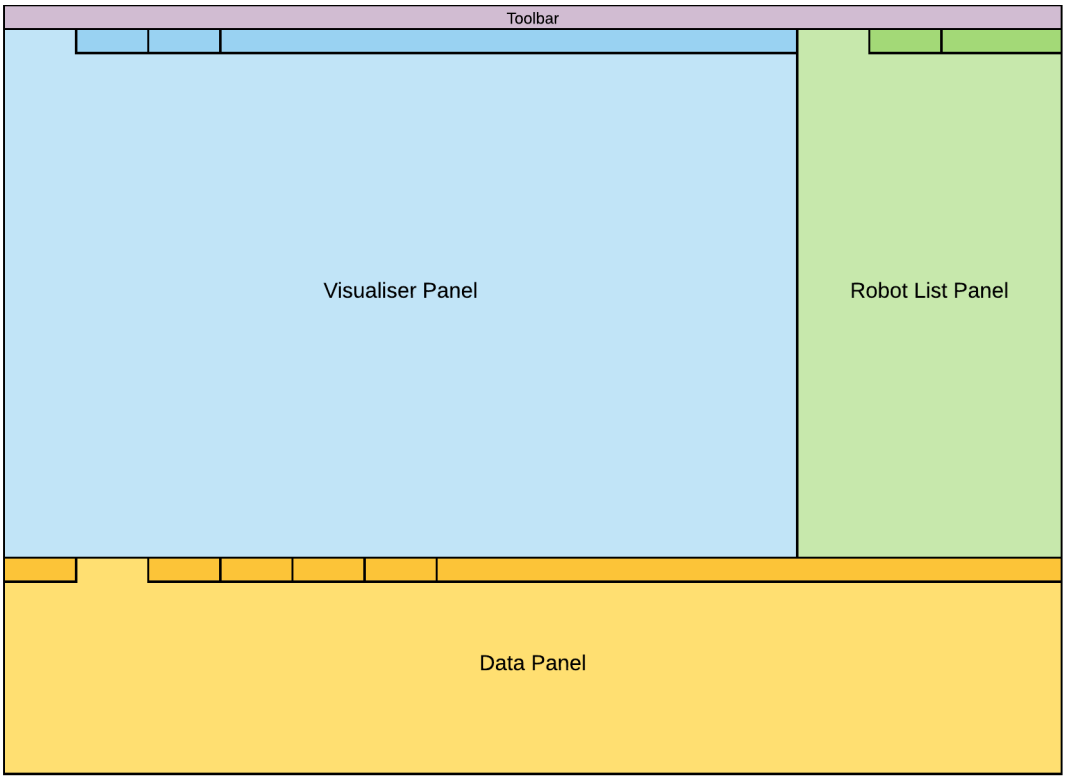


FIGURE 5.4: The design of the basic user interface layout.

The second main area of consideration during the design phase was the Graphical User Interface (GUI). It was determined that creating a well-designed, intuitive interface would be essential to satisfying the objective of providing data in a 'human readable' form. Having real time data would be useless if the user cannot parse the data displayed sufficiently quickly, due to a poorly designed or confusing UI. The first decision made was to try and keep the interface familiar to a computer user, through the use of standard, widely understood user interface elements.

There exists a well-defined 'language' in computer interface design, using constructs such as windows, panels, tabs, buttons, text fields and other elements which have well-understood functions. It was thought that basing the user interface design on this well-established standard would minimise the time for a new user to become accustomed to the system. The next step was determining how many panels would be necessary for the intended functionality to be possible, and how best to lay these out within the window and organise the smaller elements within each panel. Three main panels were determined to be necessary. The first would display the video feed and visual overlay, the key component of the application. A second panel would be used to display a list of the robots known to the system, so that they could be selected without obscuring the video feed. Finally a third panel would be used to display more detailed information about the selected robot in a number of different tabs. Figure 5.4 shows the basic layout design for these main panels.

The visualiser panel, highlighted in blue, takes primary place in the layout. In order to maximise the size, and therefore the readability, of the augmented video feed it was determined that this panel should occupy as much space as possible. A number of tabs allow the user to change the focus of the panel, giving access to various settings, including settings for the visualiser and camera. The robot list panel is highlighted in green on the right hand side. This requires less width, as its main function is to display a list of the known robots, and allow the user to select one. Again tabs within the panel allow the user to access functionality related to the robots, such as settings for the network connection. Finally the data panel is highlighted in yellow at the bottom of the layout. Each of the tabs provide more detailed information on a specific type of data collected about the selected robot, as well as a tab for a general overview, and a console-style log of application events. It was noted that when displaying data in this panel it should be formatted to maximise the use of the available space. This means using the full width of the window and limiting the height to avoid excessive scrolling. The design also includes a toolbar at the top of the window, another standard feature of window-based software applications. The toolbar is provided to allow quick access to useful functionalities.

Figure 5.5 shows how this layout might look when filled with some example content. The robot list identifies the two robots being tracked by the system, and shows that robot 1 is selected. The visualiser component shows an example of how the video image might be augmented with graphical overlays generated from data regarding the two robots. These overlays include indicators for each robot's position and orientation, as well as text showing the name and state of the selected robot. The dotted line shows the recent path taken by the robot, which is an example of one potential type of data visualisation. Colour is also used to differentiate the two robots. The data view at the bottom of the window shows some example data for the selected robot, which might be seen on a general 'overview' tab.

Time was also spent creating more detailed designs for some of the individual

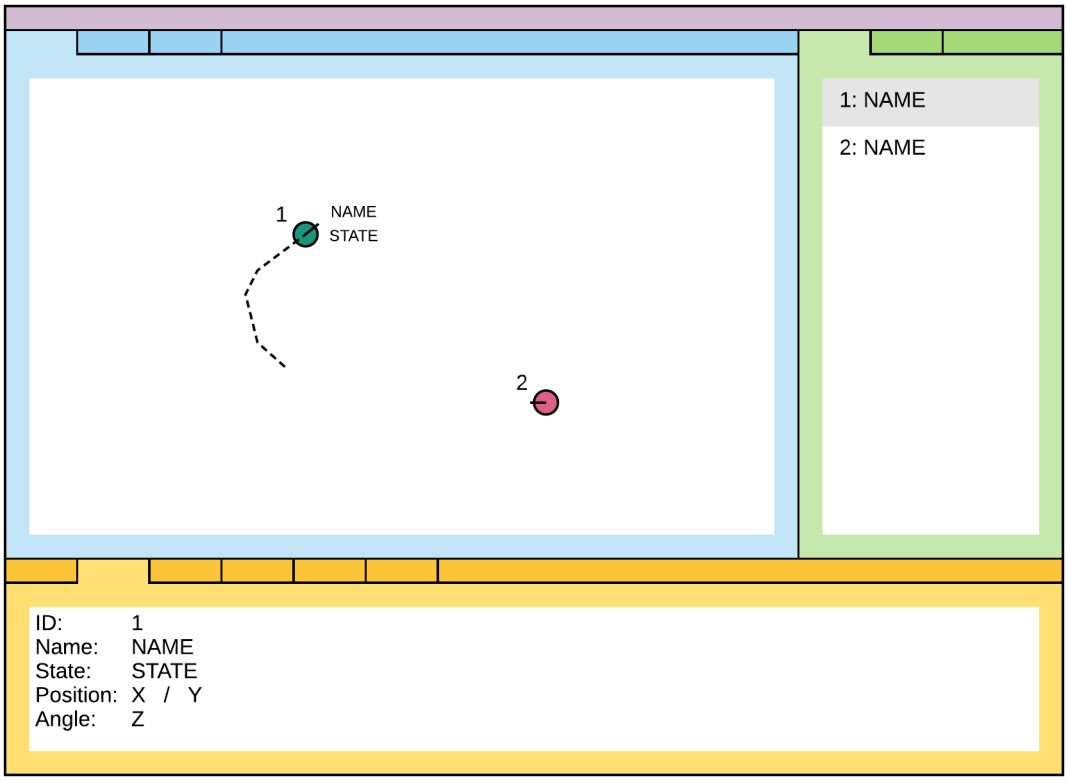


FIGURE 5.5: The basic UI design, filled with some example content.

tabs within the main panels. Each tab within the data panel needed to display a different data type, and therefore required a unique layout and design. Figure 5.6 shows the initial designs for four of the key tabs, using both textual and graphical approaches to data representation. Layout 1 shows the overview tab design, as seen previously. Layout 2 shows the design for the console tab, which displays messages about the application sequentially. Layout 3 shows the state tab, which offers additional information about the state of the selected robot, beyond simply its current state. The first box displays a list of all of the robot's known states, and the second box displays a list of the robot's recent state transitions, including the original state, the new state and the time the transition occurred. Finally layout 4 shows one possible design for displaying the robot's IR sensor data, using a bar graph to give a relative, comparable impression of each sensor's value, and the numerical displays beneath to provide the actual sensor values.

Figure 5.7 shows the design for the settings tab of the visualiser panel. The purpose of this tab is to provide access to general settings related to the visualiser, and specific settings for each of the data visualisation types. The actual settings shown in the design are representative. General settings are changed using basic form controls such as tick boxes. Settings for specific visualisations can be changed using

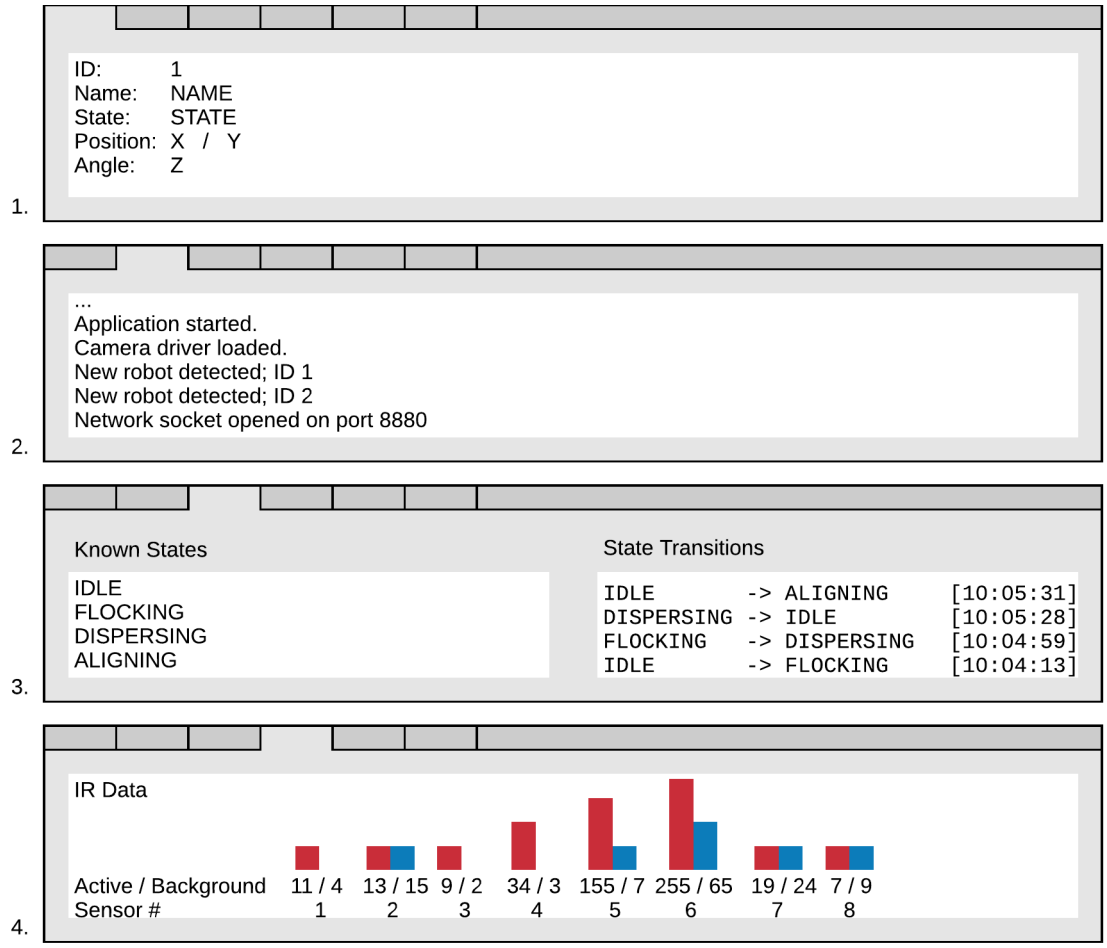


FIGURE 5.6: The initial UI designs created for some of the tabs within the data panel.

extra pop-up windows accessed by double clicking the relevant visualisation in the list. In figure 5.7 the IR data visualisation settings are shown as an example.

5.2.1 Data Visualisation Designs

As well as designing layouts for some of the more complex individual UI elements, designs were also created for some of the graphical overlays that would be shown within the visualiser. The best way to visualise the data could not be determined prior to implementing the visualiser, hence a number of variations for the overlays of each type were generated. These could then be tried in practice to determine which were the most effective. Figure 5.8 shows a selection of the overlay designs for the key data visualisation types, with the data visualisation shown in orange. Each row of the figure shows the visualisation design for a different type of data. The details of these designs are as follows.

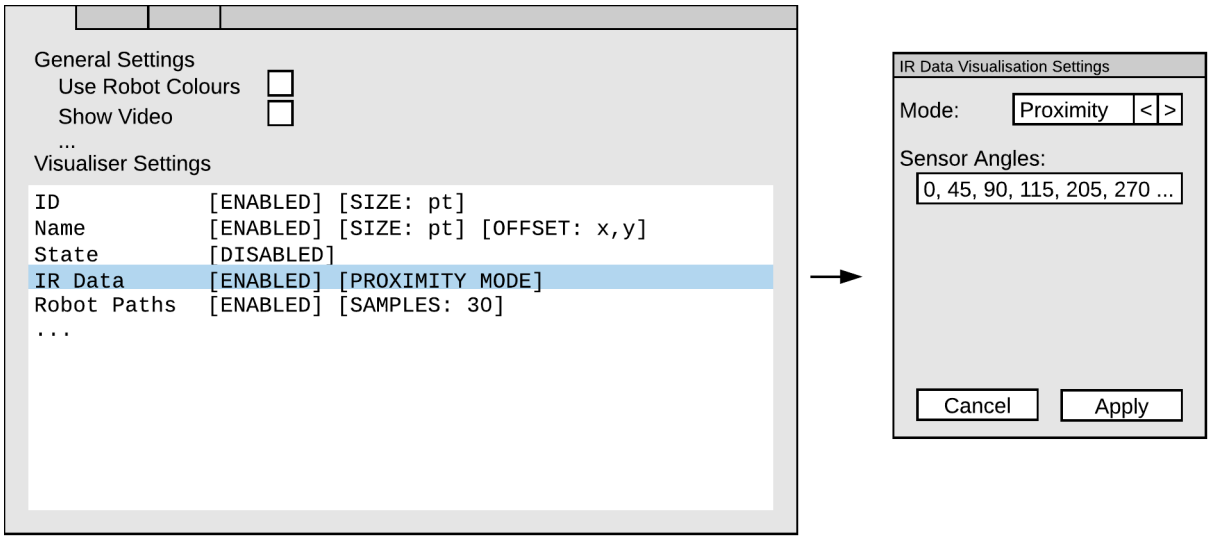


FIGURE 5.7: The initial UI design for the visualiser settings tab, and pop-up visualisation settings windows.

Row 1. Shows the representational 'robot' (a grey circle with an ArUco tag on top) with no overlay applied, for reference purposes.

Row 2. Shows three options for the position overlay, including circular and square outline designs, and a filled circle design. It was thought that the filled circle might be easier for a user to see, but had the downside of obscuring the robot, whereas the outline designs might be more difficult to see depending on the width of the lines.

Row 3. Shows three options for the direction overlay. The first uses a simple line on top of the robot, from its centre outward, indicating its direction. The second develops on the first with an arrow, which might potentially add clarity, but would be more difficult to render at small sizes. The third also uses an arrow, rendered slightly away from the robot. This arrow could rotate to match the robot's orientation, but maintain the same positional offset, or it could simply rotate with the robot about its centre point. Offsetting the arrow makes the shape clearer as it does not overlap the robot, however in practice it could overlap other overlay elements, reducing the clarity of both.

Row 4. Shows three options for the positioning of text overlays, including above and below the robot, and offset both vertically and horizontally. Text overlays are unlikely to be readable if rendered directly on top of the robot, and will always be prone to overlapping other elements if rendered with an offset.

Row 5. Shows two possible IR data overlay designs. The first uses lines to display the sensor values, where the lines are oriented to face in the direction of the sensors, and their lengths vary with the relevant sensor value. If the lines

were to vary inversely with their related sensor values this could be used as an approximate proximity display. The second design shows a 'heat map' style overlay, where the values of the sensors are indicated by boxes that change colour in relation to the relevant sensor value.

Row 6. Shows three variations of the robot path overlay design, including smooth and stepped lines as well as solid and dotted variants. A smoother path line would likely require more data to be stored, which might be a disadvantage.

5.3 Summary

In this chapter the design of the application was described, including details of both the software architecture design and the user interface design. The software architecture is designed with OOP practices practices in mind, and is based on the MVC software architecture pattern. Details of the internal structural design of two of the key functional components - the *Data Model* and the *Visualiser* - were also presented. The user interface design included details of the three main interface panels and their intended functions, as well as specific designs for the contents of different tabs within each panel. A number of possible designs for each of the different data visualisation overlays to be displayed in the visualiser were also presented. With these designs in place the implementation stage could begin to realise the application in code, as discussed in the next chapter.

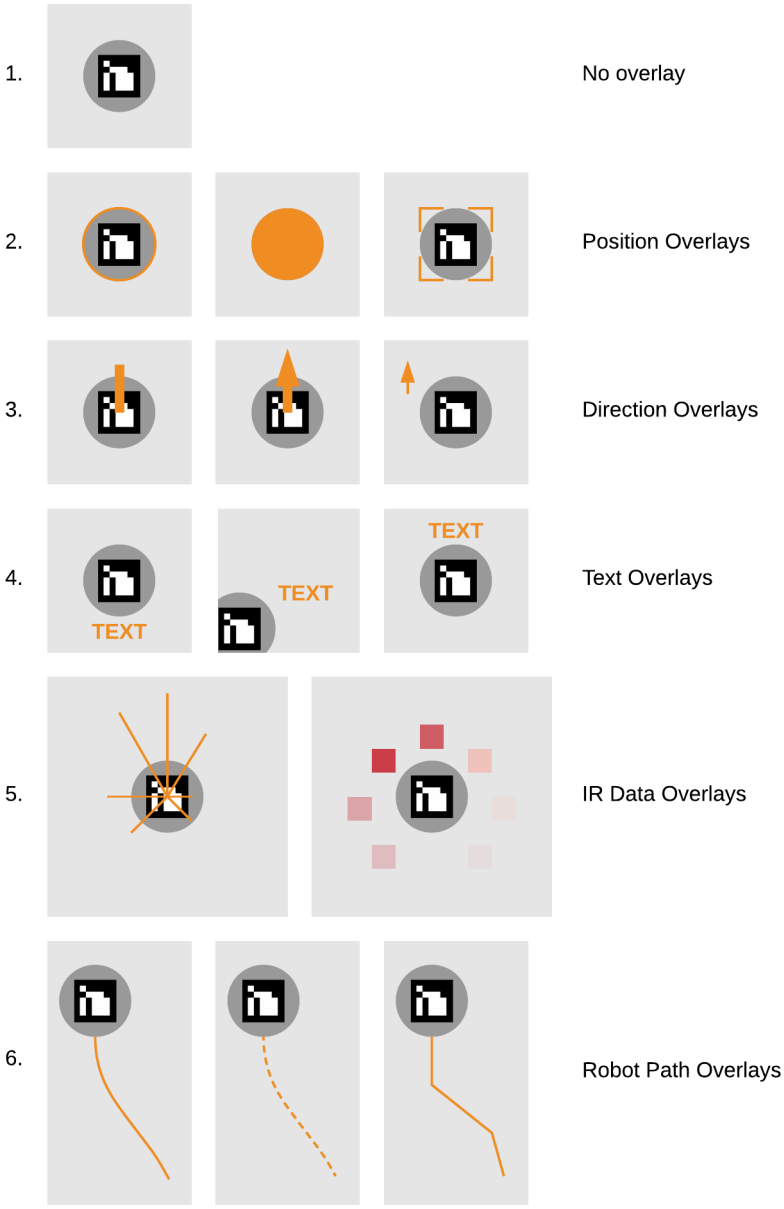


FIGURE 5.8: A selection of the designs for visualiser overlays, for a number of data types.

Chapter 6

Implementation

This section gives details of the implementation phase of the project, including descriptions of how parts of the system function, information regarding the development process and explanations for some of the key decisions made regarding the implementation.

6.1 Overview

The system was implemented closely following the design laid out in chapter 5, and comprises two parts; the main computer software application (referred to henceforth as 'the application') and a collection of software routines designed to be included within the code on the robots, which form an 'application programming interface' (API) . This API (referred to as the 'robot side code' or 'robot side API') provides the developer with functions to send data from the robot back to the application, and contains routines for handling the networking requirements to achieve this, and for correctly formatting the data. Details of this robot side API are provided in section 6.11. Both parts of the system implementation are fully independent. The application can be run on its own and receive data from any source, provided that this data correctly follows the format outlined in section 6.8. The robot side code could be used to send data to another host, provided that the robot uses the correct target IP address and port number, and again provided that the host can correctly interpret the data format. The application is the much larger of the two system parts, and therefore will be the focus of the majority of this implementation chapter.

Figure 6.1 shows the user interface for the application as it is seen at start-up. This can be compared to figures 5.4 and 5.5 to see the relationship to the user interface design. The key features of the application can be seen in this image. The visualiser component shows the video feed, augmented with information about the three visible robots, including position, direction, ID number, and the selected robot's name and current state. The robot list panel is visible on the right hand side, showing the

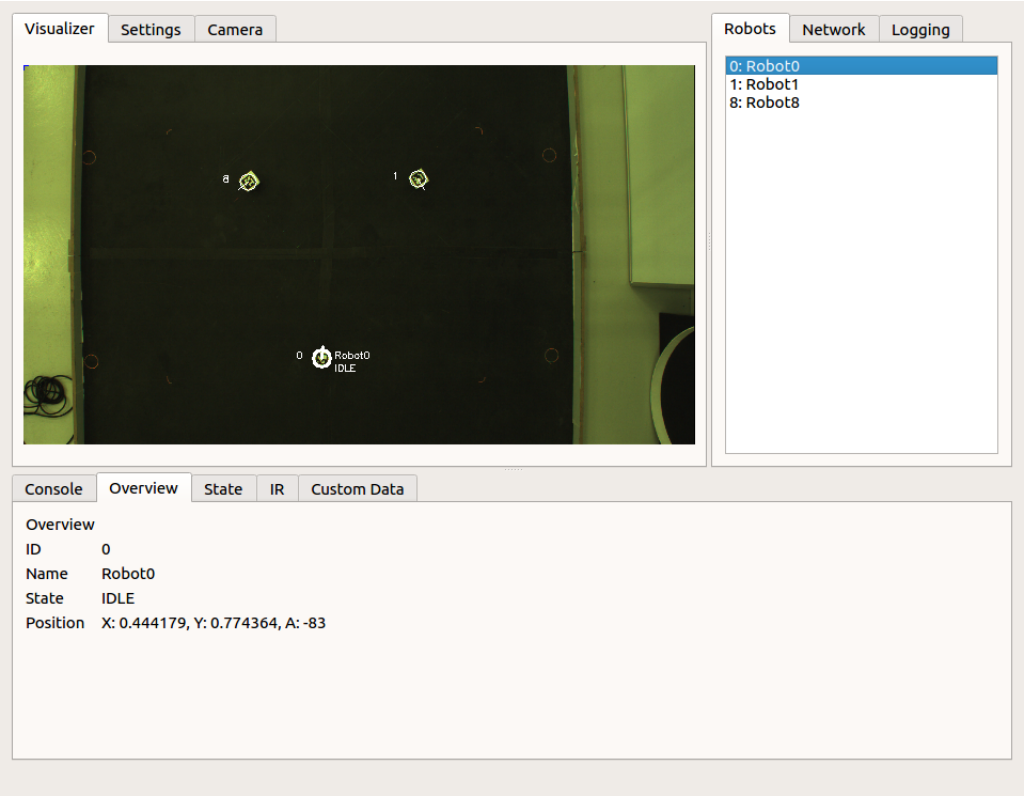


FIGURE 6.1: The user interface for the application.

IDs and names of the known robots, and which of them is currently selected. Finally the data panel can be seen at the bottom of the application, currently set to the overview tab, providing summary information about the selected robot. More detailed information about the implementation of the UI itself can be found in section 6.9.

6.2 Application Framework Selection

Developing computer software applications often involves implementing a large amount of the same low level functionality, regardless of the application. This includes low level back end functionality such as event management and dissemination, standard constructs such as timers and threads, as well as standard user interface elements such as windows, menus, panels, lists, tables, buttons and many more. It is clear that these functionalities are independent of the purpose of the application being developed, and implementing them from scratch for each new application would require a huge amount of time, and therefore be extraordinarily inefficient. For this reason the vast majority of modern software is created using some kind of application programming framework.

The purpose of these frameworks is to provide common, low level application functionality in the form of an API, which the developer can then leverage to develop their specific application. The API will usually include a number of classes to define the common UI elements, and a hierarchical tree- or node-based structure the developer can populate with these elements to create their desired UI. Most computer applications are event driven; when the user provides some kind of input such as a click or a key press an event is generated, and this event will be different depending on which UI component the user was interacting with. This event then needs to be disseminated through the application in order to trigger the correct functionality. Most application programming interfaces will handle the generation and dissemination of these events, allowing the developer to '*register*' code to be executed in response to specific events. The developer can therefore simply focus on implementing the functionality that is unique to their application.

Most modern application frameworks contain a wide variety of features in addition to those responsible for the UI and event management. These can include everything from low level components such as timers, input/output (I/O) interfaces, and networking components, to higher level multimedia handlers such as video and audio players, and rich HTML viewers. Frameworks also might include classes for creating data models which can be easily mapped to more complex user interface elements such as responsive tables.

Selecting an appropriate framework to use as the basis for this application was one of the first steps in the implementation process. All of the available frameworks have different benefits and limitations, and target a variety of different platforms, operating systems, and languages. During the design phase it was determined that the application should be implemented in C++, for reasons discussed in section 5.1. This therefore eliminated a number of frameworks, such as the Oracle Application Development Framework which is specific to the Java language, and Microsoft's .NET framework, which is C# specific. The application also needed to run on the server connected to the tracking camera, which runs a Linux operating system. This therefore ruled out any Windows specific frameworks, including one of the most widely used C++ frameworks, the Microsoft Foundation Class Library (MFC).

6.2.1 The *Qt* Framework

It was determined that the '*Qt*' application framework was the most suitable for this application, as it provides support for cross-platform compilation, including Linux, and is implemented natively in C++. A number of factors, in addition to the target platform and language, influenced this decision. The framework is widely used, and therefore has a well-tested, refined, and mature API, with a good body of documentation available. It provides a comprehensive library of classes for a range of

common application functionalities, including GUI front-end and low level back-end components. The framework also includes built-in support for multi-threading, and a structured '*Signals and Slots*' system for sending and receiving event notifications, and moving data between components and across threads. For an application such as this, with a number of external data sources in addition to the user input, and a multi-threaded design, this was determined to be a highly beneficial feature which could reduce development complexity significantly. Furthermore previous experience with Qt outside of this project had been positive, and meant a smaller learning curve would be necessary to begin developing the application. For non-commercial projects Qt is available free of charge, making it a good fit for an academic project.

The following primary features of the Qt application framework were used within this project:

- Standard GUI components and layout management classes used to create the majority of the user interface.
- Event management system to handle user input events.
- '*QTimer*' component to implement timers and recurring events, such as camera polling.
- '*QThread*' API to partition the application components into threads.
- '*Signals and Slots*' system for inter-component and inter-thread signalling and data transfer.
- '*QPainter*' component for rendering custom user interface elements.

6.3 Application Structure

The application was implemented closely following the software architecture design outlined in section 5.1. Figure 6.2 shows the structure of the application at a class level following implementation, with the arrows showing the flow of data. All of the main classes are displayed, with some of the minor classes omitted for clarity. Comparing this with the software architecture design we can see that the *MainWindow* class encapsulates the functionality of the application controller block, forming the core of the application and routing data between the other components. This class is instantiated when the application begins, and performs all of the set up operations. This involves constructing and initialising the user interface based on the layout defined in *mainwindow.ui*, creating instances of the main components, connecting the related signals and slots of each component, and moving the components to their appropriate threads.

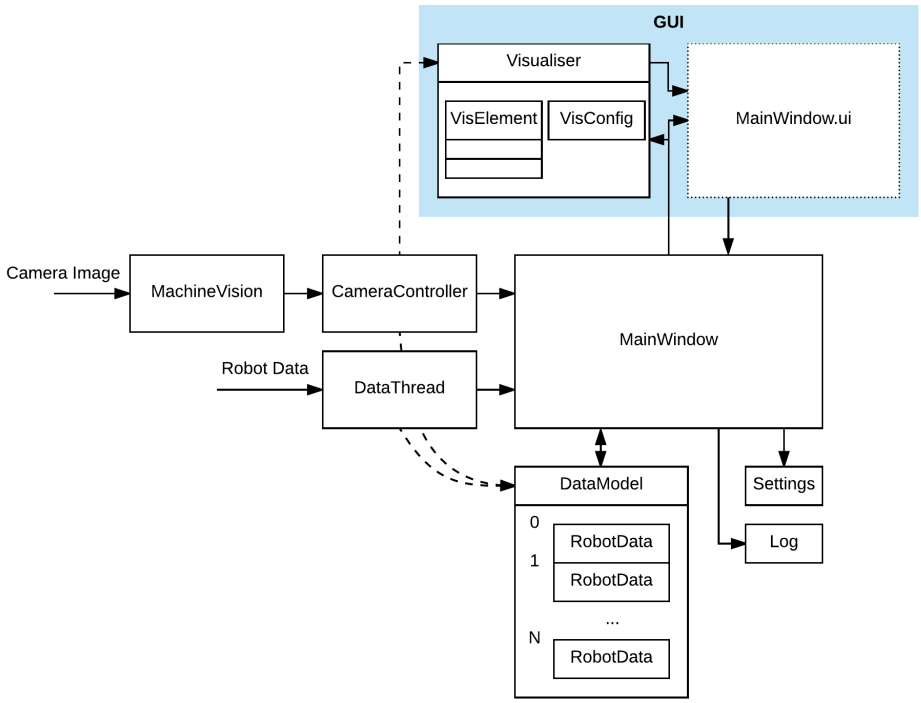


FIGURE 6.2: The structure of the classes within the application. Arrows represent the data path.

The output signals generated by the *DataThread* class when new robot data arrives, and the *CameraController* class when new tracking data is available, are routed to the *DataModel* class's *new data* input slot. Similarly the signal generated by the *CameraController* class when a new video frame has been acquired is routed to the *Visualiser* class's *image data* slot. In figure 6.2, these signal/slot connections are represented by dotted arrows. Singleton instances for the *Settings* and *Log* classes are also created during initialisation, which allow the settings and logging functions to be accessed from anywhere in the application.

Following initialisation, whilst the application is running, the *MainWindow* class is responsible for handling user input events sent from the UI, modifying the other classes as necessary depending on the input event received. Finally, when the application is closed, the *MainWindow* class is responsible for tearing down the other components. This involves stopping any active timers, stopping the various threads, and releasing all allocated memory.

The GUI portion of the software architecture design is encapsulated into the UI description file *mainwindow.ui*, which then connects back to the *MainWindow* class through Qt's UI event management system. Section 6.9 provides details of the user interface implementation. The visualiser portion of the GUI is encapsulated into the *Visualiser* class, which handles generating the graphical overlays, applying them to

the latest video image, and displaying the resulting augmented video image. Generating the overlays involves a number of smaller classes, as shown in figure 6.2. Section 6.10 provides more detail about the implementation of the visualiser class and its sub-classes. The camera controller and tracking code components are encapsulated in the *CameraController* and *MachineVision* classes respectively, discussed further in section 6.6, and the network controller component is implemented in the *DataThread* class, which is discussed in section 6.7. Finally the data model component is implemented in the *DataModel* class, making use of the smaller *RobotData* class as per the design in section 5.1.1. This implementation is discussed in section 6.5.

6.4 Source Code

The system source code is implemented in two groups of files; a collection of C++ source and header files which make up the main application, and a smaller collection of C++ source and header files which handle the robot-side portion of the system. In addition to its source files, the main application also relies on a number of other files which are used by the Qt system to define the user interface layout, and manage the build process. A full list of all source code files is given in appendix C. Table C.1 details the names and purposes of all files within the main application. Table C.2 details the files that make up the robot side API.

6.5 Data Model

The data model forms a core element of the back end of the application. For each robot being tracked by the system, the data model is required to store the information related to that robot. This is done in a structured manner, such that other parts of the application can query specific data within the model easily. The contents of the data model are updated whenever new data arrives, and are consulted when rendering the user interface. Section 5.1.1 describes the design of the data model and its hierarchical structure. The implementation follows this design closely, with the *DataModel* class contained in *datamodel.cpp / .h* encapsulating the top level data model container, and the individual robot data object encapsulated in the *RobotData* class in *robotdata.cpp / .h*.

The *DataModel* class uses a standard C++ '*vector*' container to maintain a list of *RobotData* objects. Each of these *RobotData* objects describes one robot that is currently known to the system. The list is ordered based on the robots' IDs, and is sorted after each new insertion. The vector container was chosen as it does not require a fixed size, and can be sorted and iterated efficiently. The *DataModel* class

provides convenience functions for data retrieval functionality, such as retrieving the data object for a given robot based on ID number. It also provides a function for entering new data into the model, which inherits the properties of a '*slot*' within the Qt framework, allowing it to be called from other threads. Each new data packet received from the network is routed to this slot, as well as position data obtained from the tracking system. All data is supplied in the form of a string in one of the packet formats described in section 6.8. Code within the data model class then handles interpreting the string, determining its purpose and source robot, separating out the data content, and updating the relevant robot within the model. Whenever data arrives from a previously unknown robot, this code handles creating a new *RobotData* object, and adding it to the list.

The *RobotData* class encapsulates the data for a single robot. This involves storing a number of different data points, in a variety of different formats. The class then acts as a relatively simple container for this data, providing functions for retrieving and changing values. Table 6.1 outlines the data points contained within the class.

TABLE 6.1: The contents of the *RobotData* class.

Data Point	**Type**	**Description**
Robot ID	Integer	The numerical ID of the robot, used by the tracking system and when transmitting data.
Robot Name	String	The name associated with this robot. Set by the user when programming the robot, and reported in watchdog packets.
State	String	The current state of the robot.
Known States	List of Strings	A list of all states the robot has previously reported.
Position	2D Vector	The current position of the robot expressed as a proportional coordinate vector.
Angle	Integer	The current angle of the robot in degrees.
Colour	OpenCV Scalar	The colour used for this robot in the visualiser, if colours are enabled, expressed as an OpenCV Scalar struct in BGR format.
IR Data	Array of Integers	The most recent IR sensor readings for this robot. One value per sensor.
Background IR Data	Array of Integers	The most recent background IR sensor readings. One value per sensor.
Custom Data	Key Value Map	All custom user data. Stored as a map of key/-value pairs.

In addition to this data, the class also maintains a list of recent state transitions, and a short term history of the robot's position. The state transition list uses a custom structure to store the state before the transition, the state after the transition, and the time the transition occurred. A fixed size array of these custom structures is maintained and updated each time a state transition occurs, acting as a first-in first-out (FIFO) queue. The position history is stored in a similar fashion, using a fixed size array of coordinate pairs, which is updated every Nth position update. This interval can be configured by the user, with a lower interval giving a higher resolution but a shorter history, and vice versa for a higher interval. Functions are provided to retrieve data from these queues when needed.

6.6 Video Feed and Tracking System

Functionality for acquiring images from the machine vision camera, and running the ArUco tag tracking algorithm, is encapsulated in the *CameraController* and *MachineVision* classes, which are implemented in *cameracontroller.cpp / .h* and *machinevision.cpp / .h* respectively. Video is retrieved from the camera one frame at a time, and can therefore be thought of as a sequence of discreet images. A call to the camera driver to obtain the next image might block execution whilst it waits for the image to become available. Hence in order to maximise application performance and ensure responsiveness these classes are run on a separate thread dedicated to camera functionality.

The *CameraController* class handles the higher level operations such as running a timer to periodically poll for the next image, supplying the correct dimensions for the image, and converting the image and tracking data into formats which can be passed back to the main thread and used in the UI and data model respectively. The application threading is handled through the use of the Qt framework's *QThread* API, and communication between threads utilises the framework's '*signals* and *slots*' feature, which allows components on different threads to send and receive data in a managed, thread-safe manner. The *CameraController* class therefore utilises two signals; one for emitting the camera image data, and another for emitting the robot position data. At initialisation time the application's core class, *MainWindow*, connects these signals to matching slots within the *Visualiser* and *DataModel* classes respectively.

The *MachineVision* class handles the lower level operations related to the camera and the tracking system, including setting up the camera driver, retrieving and resizing individual images from the camera, and running the ArUco tag detection algorithm. The ArUco software is implemented as an additional component of the OpenCV image processing library [42] (discussed below), and provides a function to run the tag detection algorithm on a given image, for a given dictionary of tags.

For each tag detected in the image that matches one in the chosen dictionary, the ArUco algorithm returns the pixel coordinates of the four corners of the tag. The *MachineVision* class includes code to average these four coordinates, acquiring a central pixel coordinate, which is then converted to a 'proportional' coordinate; two numbers between 0 and 1 which represent the horizontal and vertical components of the position as a proportion of the full height and width of the image respectively. This ensures that a robot's position can still be correctly displayed after the image has been resized, without having to maintain information related to the resizing operation. The angle the robot is facing is also calculated. This is done by first calculating the coordinate of the fourth corner in a coordinate system where the third corner is positioned at the origin, and then applying an arctangent function to this coordinate to obtain the angle. This angle is then converted to degrees and stored as an integer in order to reduce complexity, as a precision greater than one degree was not deemed necessary.

6.6.1 The *OpenCV* Image Processing Library

A number of the components within the application are required to manipulate image data. The image data acquired from the tracking camera by the *CameraController* and *MachineVision* classes must be stored in a format that can be processed by the ArUco tag detection algorithm. This image data must then be passed to the *Visualiser* class, which renders the graphical overlays on top. Selecting a suitable image processing library was one of the early steps in the implementation process. The ArUco tag detection algorithm is implemented as an extension to the *OpenCV* image processing library, so this seemed to be the obvious choice. However the Qt framework supports its own image data format and drawing classes, so the image could be converted to this format once the tracking data had been extracted. OpenCV is a widely used, feature-rich, powerful image processing library, and ultimately it was decided that all image manipulation should make use of it where possible. It was hoped that this would improve maintainability in the future, as OpenCV is more widely used. If the code was ever to be ported away from the Qt framework, image processing functionality implemented using OpenCV would require less change.

6.7 Networking

A key element of the system is the wireless retrieval of information from the robots. This required the implementation of networking functionality within both the application and the robot side API. As mentioned previously the Linux extension boards on the e-puck robots feature a WiFi adapter, so WiFi was selected as the target wireless networking technology to implement this functionality with. The next step was

to look at the networking requirements in more detail, and decide on which transport layer protocol to use.

The requirements for the networking portion of the system were relatively simple, and can be summarised as follows:

1. Must utilise a WiFi network.
2. Must allow a large number of sources to transmit data to a single host.
3. Must allow for frequent transmission of small packets of data.

WiFi networks utilise the standard Internet Protocol (IP) [43] network layer protocol. There are two commonly supported transport layer protocols which run on top of IP, the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). TCP is a managed and delivery-error checked protocol, and therefore guarantees that packets will be transmitted in the correct order, with lost packets being retransmitted. This adds overheads such as acknowledgements to the protocol, and requires an established 'connection' in order to function correctly. TCP also operates a queueing system, whereby packets for transmission are sometimes held until a number of them are ready, and can therefore be grouped together and sent.

UDP [44], by contrast, does not error check the delivery of packets, making no guarantees that a packet will be received, or that packets will be received in the correct order, removing the need for an established connection and reducing the overheads involved. Packets can therefore be sent from any application to any target IP address and port on the network, without first establishing a connection with another application. Packets are also sent immediately, with no queueing system in place. It was determined that UDP would be the most suitable for this system, for a number of reasons. The connection requirements of TCP would require the application to form a connection with each robot prior to transmitting data, which would add unnecessary complexity. Using UDP also ensured that the packets were transmitted immediately, reducing the potential for latency in the system. The lack of delivery checking was not considered an issue, as the robots would be transmitting updates frequently enough that a single lost packet would not cause a significant issue.

The networking functionalities of the application are encapsulated in the *DataThread* class, found in *datathread.cpp / .h*. This class is run on a dedicated thread to ensure that potentially blocking operations do not impact application performance. The class contains routines for dealing with the low level network requirements, such as establishing a socket through which to receive data packets from the robots, and continually listening on this socket for new data. Figure 6.3 shows the operation of the *DataThread* class as a flow diagram. The '*Packet Listening Started*' signal, highlighted in green, is generated when the user presses the '*Start Listening*' interface button in

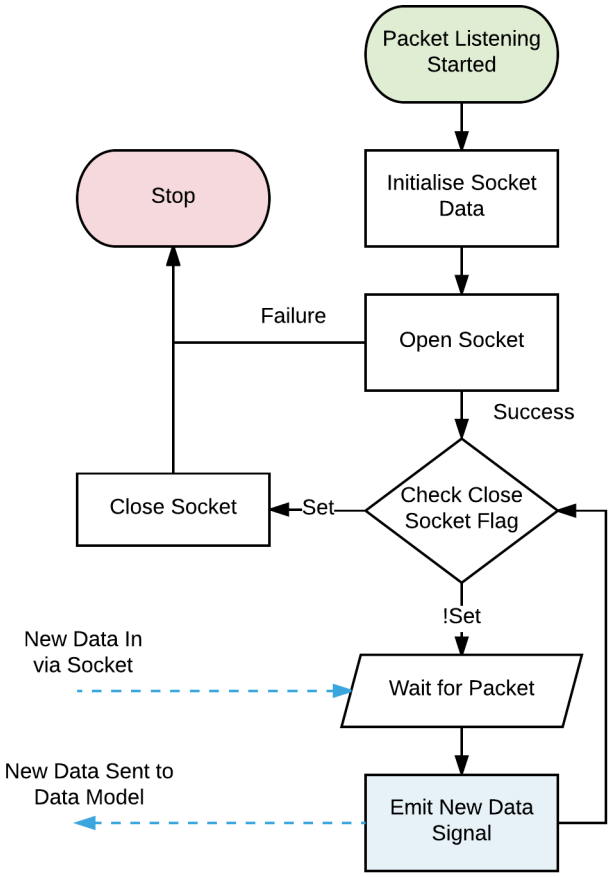


FIGURE 6.3: A flow diagram showing the sequence of operations carried out by the *DataThread* networking class.

the networking tab. All socket operations are implemented using structures and definitions from the standard C++ networking libraries.

Data received from the robots is passed to the main thread through a Qt signal, using the Qt signals and slots interface mentioned previously. The main application class, *MainWindow*, connects this signal to the appropriate slot in the *DataModel* class at initialisation time. The application side networking can be configured by the user in the *network* tab of the right-hand panel of the user interface. Here the user is able to enter a desired port number on which to receive data, and can start and stop the application listening for packets on this port by pressing the '*start/stop listening*' button.

For the robot side API, the networking functionality is implemented in a similar way. The initialisation function establishes a target socket based on a supplied IP address and port. This should match the IP address of the computer or server running the main application, and the port chosen by the user within the main application. When any of the functions for sending specific data packets are called, the data is sent to this target socket as a UDP packet. Section 6.8 discusses the format of the

data within these packets. All socket operations are once again implemented using the standard C++ networking libraries to increase portability.

6.8 Data Transfer Format

In order for the application to interpret and use data received from the robots, a common format for exchanging this data needed to be defined. Both sides of the communication link then need to use this format correctly when constructing and de-constructing packets. A number of different options were considered for achieving this, ranging from super-lightweight custom packet formats using the minimum number of bytes, to established existing solutions such as the JSON data interchange standard [45]. The primary concerns when making this decision were a desire to minimise any overheads in terms of extra code needed on the robot side, as the robots have limited memory, and to ensure the format remained as simple as possible so that future extensions to the system, such as implementations for other robots, could be programmed with relative ease. The size of packets was also a concern, as minimising network traffic where possible would benefit the system if used with a large number of robots.

It was ultimately decided not to use JSON, to avoid the need for any additional code libraries to be stored in the robot's memory, and to instead use a custom, simple, string-based packet format. All data to be transmitted from the robot to the application is therefore converted to a string which is then transmitted in the data packet. As well as containing the data, the string must identify the robot and describe the type of data within. The format for these strings is defined as three sections separated by space characters. The first section contains the numerical ID of the robot sending the packet. The second contains a number identifying the type of data contained in the packet. The last section contains the packet data, and has a variable format, depending on the packet's type. Figure 6.4 gives a visual representation of this format. Table 6.2 lists the packet types, and describes the format of the 'packet data' section for each. An example of each packet type is given, using the arbitrary robot ID of 6.

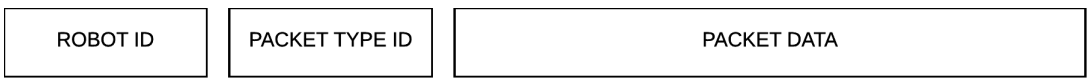


FIGURE 6.4: The general format for each data packet.

TABLE 6.2: The format of the data section and the purpose of each packet type.

Watchdog Packet	
Type ID	0
Format	[ROBOT NAME]
Purpose	Sent periodically to inform the application that the robot is still active. Also contains the robot's name, as should be displayed in the application.
Example	"6 0 Robot_6"
State	
Type ID	1
Format	[CURRENT STATE]
Purpose	Informs the application of the robot's current state.
Example	"6 1 IDLE"
Position	
Type ID	2
Format	[X POSITION] _ [Y POSITION] _ [ANGLE]
Purpose	Provides the application with a robot's position and orientation. This data is not sent by the robot, but instead comes from the tracking code.
Example	"6 2 0.23 0.682 110"
IR	
Type ID	3
Format	[SENSOR 1 DATA] _ [SENSOR 2 DATA] _ ... [SENSOR N DATA]
Purpose	Contains a robot's infra-red sensor readings. Each sensor value is separated by a space, and the packet can contain as many values as the robot has IR sensors.
Example	"6 3 101 93 115 112 103 98 365 2850"
Background IR	
Type ID	4
Format	[SENSOR 1 DATA] _ [SENSOR 2 DATA] _ ... [SENSOR N DATA]
Purpose	Contains a robot's background infra-red sensor readings. Formatted the same as the standard IR data packet.
Example	"6 4 95 87 99 110 89 98 103 82"
Message	
Type ID	5
Format	[MESSAGE STRING]
Purpose	This packet allows any general message to be sent from the robot to the application, and will be displayed in the application console and recorded in the logs.
Example	"6 5 Robot entering hibernation mode"
Custom Data	

Type ID	6
Format	[KEY] _ [VALUE]
Purpose	Contains a piece of custom user data, in the form of a key value pair.
Example	"6 6 DebugCounter 540"

6.8.1 Constraints

For a number of the packet types constraints are placed on the input data, both in terms of format and in terms of the range of valid values. The space character is used as a delimiter to separate portions of the packet, hence in all cases except the message packet a space character cannot appear in the middle of any of the data. This means that robot names and states cannot include spaces, nor can custom data keys or values. The message packet type is an exception, and will treat any characters following the second space, which marks the end of the header data, as a single string. This string then forms the message content.

For the watchdog, state, position and custom data packet types, the correct number of space-separated data portions must be included, or the string is invalid. This means one, one, three and two data portions respectively. In the case of the IR data a minimum of one data portion must be included, but there is no upper limit. Any portions above the number of supported sensors will be ignored. The IR data values must be positive integers in the range of 0 to 4095. Values outside of this range will be clamped to the closest value that satisfies this requirement. Floating point values will cause the packet to be marked as invalid and ignored. The numerical data in the position packet must be two floating point values (X and Y position), followed by one integer value (angle). A floating point value can be interpreted from an integer (the decimal point is therefore not necessary), however an integer angle value cannot be interpreted from a floating point data portion, and this will cause the packet to be marked as invalid and ignored by the application.

6.9 User Interface

In order to implement the user interface the designs shown in section 5.2 had to be realised using the UI component classes provided by the Qt application framework. *Qt Creator*'s layout tool was used to arrange all the basic elements of the UI, and generate an XML-based descriptor file (*mainwindow.ui*) which describes this layout. When the application is initialised this file is parsed and used to populate the user

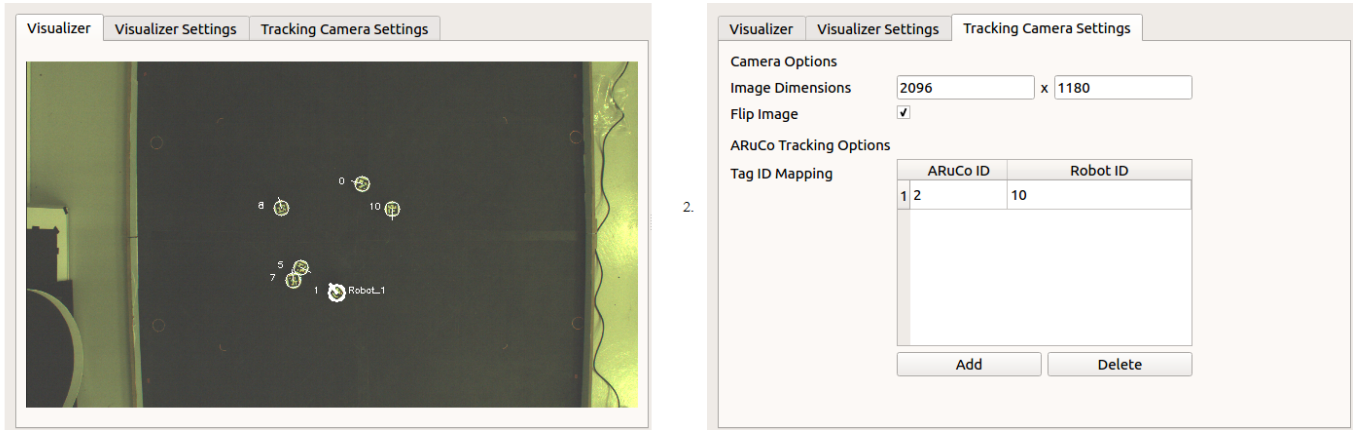


FIGURE 6.5: The visualiser panel, with two of the different tabs shown.

interface. Events generated by the interface are linked to functions within the *MainWindow* class using the signals and slots system. Code was then implemented within each of these functions to execute the correct actions based on the input received.

The user interface was implemented following the three panel layout outlined during the design phase. The panels are divided by splitters which can be dragged by the user to adjust the size of each panel to suit their screen. When reduced past a certain minimum size the panels will be minimised completely, freeing up more UI space for the other panels. This allows the user to hide the robot list panel or data panel in favour of a larger visualiser display. Each panel features a tab controller component to allow the user to switch between multiple views. This was necessary to provide sufficient space for the required features and all the various settings.

The visualiser panel has three tabs, the first showing the main visualiser view, the second providing access to the settings related to the visualiser, and the third providing access to the settings related to the camera and tracking system. The visualiser and its settings are discussed in section 6.10. The camera settings tab allows the user to input the image dimensions of their camera output, so that the visualiser can display this at the correct aspect ratio. It also allows the user to set up mappings between specific ArUco tag numbers and robot IDs. This is useful if the ID number of the tag attached to a robot does not match the ID number the robot is using to report data. By applying a mapping the user can instruct the system to apply tracking data for a specified ArUco tag ID to the robot entry within the data model with a different specified robot ID. Figure 6.5 shows two of the tabs within the visualiser panel; the left hand image shows the actual visualiser tab, whilst the right hand image shows the camera settings tab. The visualiser settings tab is shown in section 6.10.

The robot list panel also features three tabs. The first displays the robot list, and allows the user to select from the robots currently known to the system. By selecting

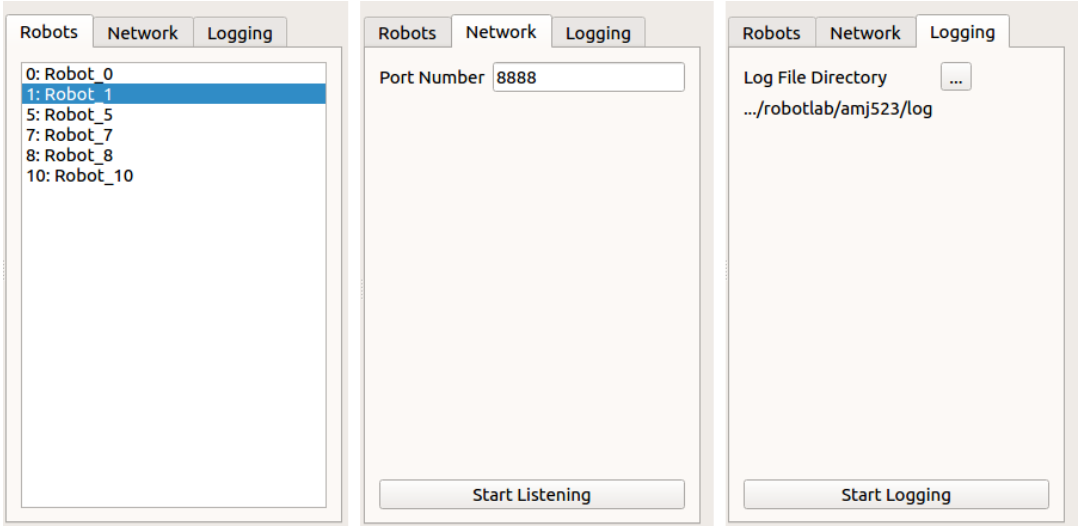


FIGURE 6.6: The robot list panel, showing all three tabs.

a robot the user makes it the current focus of the application. This can cause the visualiser to display more visualisations for the selected robot depending on the visualiser settings. It will also make the selected robot the focus of the information shown in the data panel. The second tab provides controls for the user to configure the networking functionality, including a text box to enter the desired port number, and a button to start and stop listening for data packets. The third tab provides controls for the user to configure the data logging functionality, including a button which opens a file browser for setting the directory path where logs should be stored, and a button to start and stop the data logging. Figure 6.6 shows the three tabs within the robot list panel.

The data panel features five tabs. The first tab displays a simple text based console, which reports messages regarding the application itself, as well as any messages received from the robots in message packets. The messages are displayed sequentially, with the most recent message always being added as a new row at the bottom of the console. The second tab is the overview tab, which provides a summary of the selected robot's key data. The third tab is the state tab, which displays two lists of state information regarding the selected robot. The first lists all of the robot's known states, and the second lists recent state transitions, including the state before and after the transition, and the time at which the transition occurred. This should help the user to determine if a robot is moving through its states correctly, and check that it is not changing states out of order or too frequently.

The fourth tab displays the selected robot's IR sensor data in the form of a bar graph. This includes two bars for each IR sensor, one for the active sensor reading and one for the background reading, with colour being used to differentiate between the two. The numerical values of both are also displayed below the bar, so that the user can ascertain the actual values if necessary. Finally the fifth tab displays the

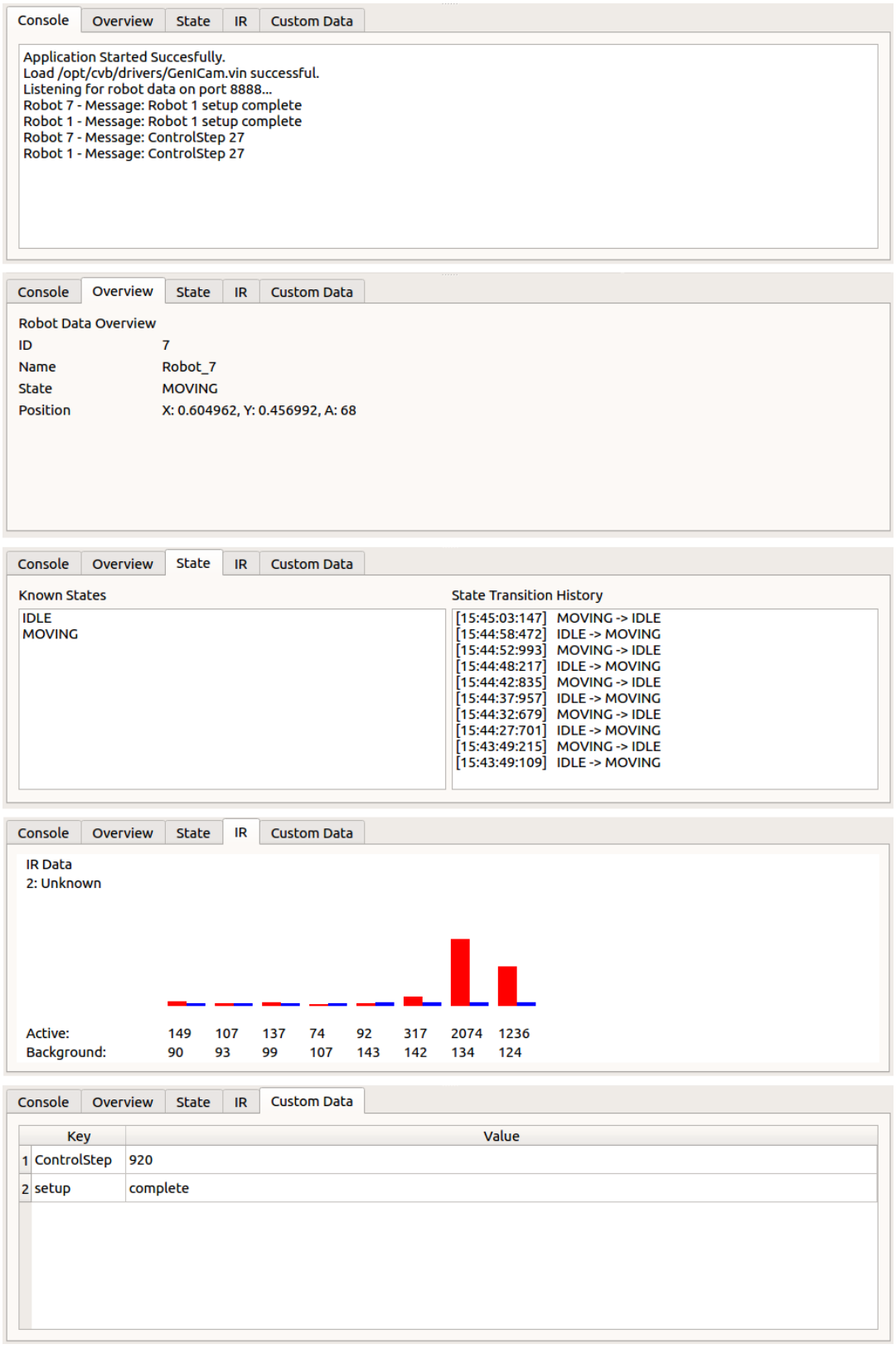


FIGURE 6.7: The data panel, showing all five tabs.

custom user data for the selected robot. This data is organised into a table showing the custom data keys in the first column, and the values for each respective key in the second column. All five tabs update their data in real time, in response to received packets. This means the user is kept up to date with the latest information immediately. Figure 6.7 shows the five data panel tabs.

In addition to the main user interface, a number of extra 'dialog' windows are used to provide the user with access to various settings. For each of the data visualisations which feature settings beyond a simple enable/disable , a dialog window class was implemented. The names of these classes take the form **SettingsDialog*, where * is replaced by the visualisation type. The files containing definitions for these classes follow the same naming format, **settingsdialog.cpp / .h*. Dialog windows are a commonly used tool within applications programming. They act as pop-up windows which usually require the user to either confirm or cancel some action or change. In this case the dialog windows present controls for changing specific visualisation settings, and the user can then either apply their changes or cancel them using the standard accept/reject buttons at the bottom of the window.

6.10 Visualiser

The '*Visualiser*' is the name given to the custom user interface component that renders the augmented video feed. Implementing this component was key to satisfying the portion of the project aim related to augmented reality, and it forms one of the most visible elements of the system. The main visualiser component is defined in the *Visualiser* class (*visualiser.cpp / .h*), and a number of extra classes are used to define the associated settings and routines for visualising specific data types (*VisConfig, VisID, VisName, VisState, VisPosition, VisDirection, VisProximity, VisPath and VisCustom*). Section 6.6 describes the process of retrieving images from the camera and tracking the robots. The image data then arrives at the visualiser via a Qt slot function. At this stage the image is augmented based on the data in the data model, by iterating over the list of robots, and for each one iterating over the list of data visualisations, calling the render function for each. These render functions take the image and the current robot's data as arguments, and then add the relevant graphical representation to the image using the drawing functions within the OpenCV image processing library. This process is implemented following the design outlined in section 5.1.2, and figure 5.3.

It was decided that an individual class would be implemented for each type of data visualisation. Each class derives from the abstract class *VisElement*, which defines the general outline of a data visualisation class, and then each specific implementation defines how to generate the graphical overlay for that data type. The *VisConfig* class stores a collection of all the *VisElement*-derived objects, which can be

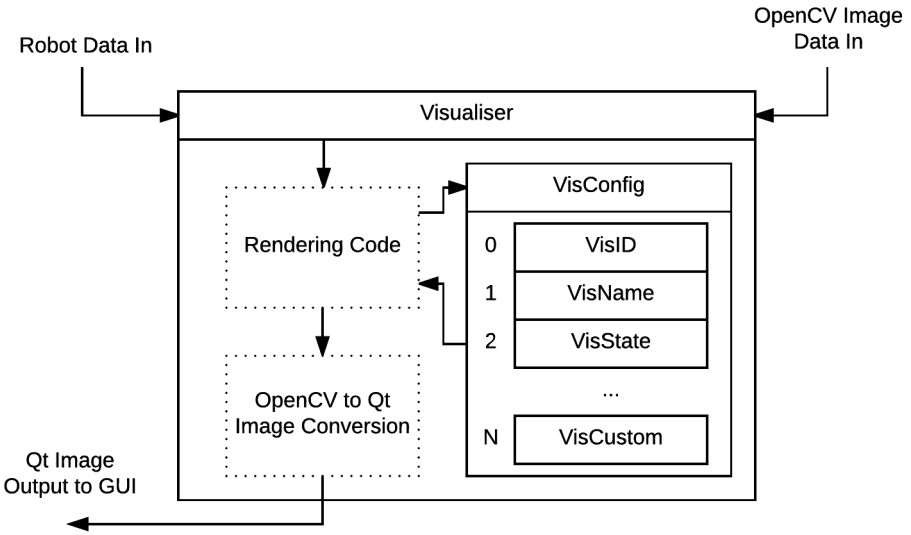


FIGURE 6.8: The path of data through the visualiser when generating each frame.

iterated through to render each one. The use of an abstract class was necessary in order for the different visualisation element classes to be stored in the same collection. The overall aim of this implementation method was to make the visualisation process simpler to manage, and to follow object oriented practices, making it easier for new visualisation types to be added without modifying the underlying system. This also allows each data visualisation object to maintain its own settings, so that the visualiser component itself need not be aware of the details of the configuration of each data visualisation element. Instead each data visualisation element simply checks its own configuration when its render function is called. Figure 6.8 shows the flow of data through the visualiser when rendering a frame.

The rest of the code in the *Visualiser* class relates to embedding the OpenCV image within a Qt UI widget, and tertiary functionality such as detecting clicks within the image frame, and retrieving the frame's size. Code is included to allow the user to select robots by clicking on their position in the visualiser. The OpenCV to Qt image conversion is done by instantiating a QImage instance directly from the internal pixel data of the OpenCV image. This QImage is then drawn onto the widget.

6.10.1 Data Visualisations

The visualiser supports a number of specific data visualisations:

1. Highlighting robot position.
2. Indicating robot 'orientation' by showing forward direction.

3. Displaying the ID of a robot as text.
4. Displaying the name of a robot as text.
5. Displaying the current state of a robot as text.
6. Displaying a graphical representation of a robot's IR sensor data.
7. Displaying the recent path a robot has taken as a line behind the robot.
8. Displaying a specific piece of custom data related to the robot as text.

The designs for these visualisations, as described in section 5.2 and shown in figure 5.8 were tested, and the most effective selected for the final implementation. Figure 6.9 shows the implemented visualisations. Each row of figure 6.9 is summarised as follows.

Row 1. Shows all of the text-based visualisations individually, followed by the full set combined. The text based data is rendered adjacent to the robot, on the right hand side, where even relatively long strings will not overlap the robot's other visualisations. The exception to this is the ID number, which is rendered above and to the left, as this is unlikely to be long enough to cause overlapping issues.

Row 2. Shows the robot position and orientation visualisations, alone and combined. Robot position is indicated using an outlined circle. This was chosen over the filled circle or square designs as it was the least intrusive on other elements of the image. Direction is indicated by a line, starting from the robot's centre and extending outwards in its forward direction. This was selected over the arrow based designs, as rendering the arrows at small enough sizes was not feasible.

Row 3 Shows the two types of IR data visualisation. For the IR data visualisation, both designs were implemented, and the user can select which to use. The first renders a line for each of the sensors, extruding out from the centre of the robot at the angle of the respective sensor. The length varies inversely with the value of the sensor, in an attempt to approximate a proximity measurement. The second mode, referred to as the 'heat map' mode, renders a number of small boxes around the robot, positioned to match the sensors. The colour and size of each box varies to indicate the sensor value. The choice to provide both was made because the line based visualisation is useful in certain scenarios when testing the response of individual sensors, but the box-based 'heat map' visualisation provides a more immediately understandable indication of the sensor values.

Row 4. Shows the path visualisation. The robot's recent path is visualised as a trail of line segments behind the robot. One of the original designs proposed a dotted line, but in the end this proved to be less visually clear than a solid line. The smoothness of the line was also considered in the design, but during implementation it became clear that the smoothness of the line was inherently related to the number of samples. A trade-off therefore had to be made between the smoothness and accuracy of the line, and its length. A setting was added to allow the user to vary the sampling interval for the line, thus allowing them to choose between a longer but less accurate and more jagged path, or a shorter but smoother and more accurate one.

Row 5. Finally row 5 shows two possible combinations of the visualisations. In the first the robot's ID, position and path are shown, as well as its IR data in heat map mode. In the second the robot's ID, name, state, position, direction and path are shown.

FIGURE 6.9: The different data visualisations as implemented.

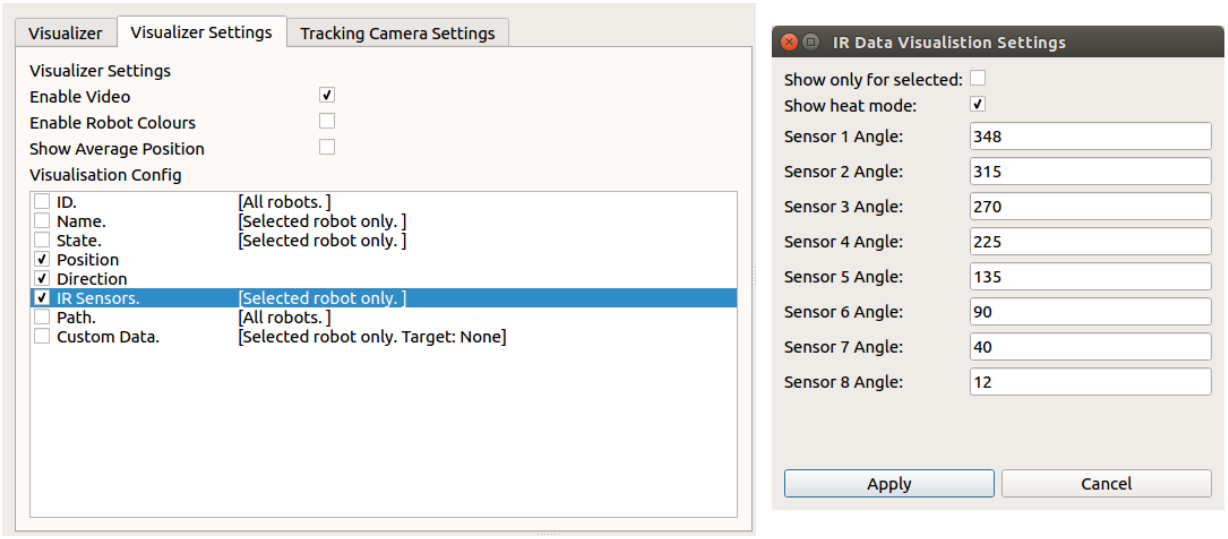


FIGURE 6.10: The visualiser settings tab, and one of the visualisation settings dialog windows.

Each visualiser element can be enabled and disabled via the settings tab. Some of the visualisations have more complex settings, which can be accessed by double clicking the specific element in the visualisations list, also in the settings tab. For the majority of the visualisations the user has the option to set them to render only for the selected robot, or for all the robots. The IR data visualisation also has settings to switch between the proximity and heat map modes, and to set the angles of the sensors. The path visualisation has a setting for the sampling interval, as previously mentioned. Finally the custom data visualisation allows the user to input the key string for the data point they want displayed. Figure 6.10 shows the visualiser settings tab, as well as one of the dialog windows for accessing detailed visualisation settings. This dialog is showing the settings for the IR data visualisation.

6.11 Robot Side API

The robot side API is encapsulated by the *DebugNetwork* class, defined in '*debug_network.cpp / .h*'. The contents of the *DebugNetwork* class are described in appendix D, table D.1. In order to utilise this API a user should modify their robot controller code to include the *debug_network.h* header file, call the *init* function at start-up time, and then call the various data reporting functions whenever necessary. The user is therefore in charge of how frequently data is transmitted, and can decide whether to simply send updates to the application when the robot's data changes, or to transmit a fixed quantity of data every control step.

It was potentially possible for the API to handle all data reporting in the background, without requiring the user to specify when to transmit data in the controller

code, however this approach was not taken for a number of reasons. Firstly it limits the control and flexibility available to the user. Each swarm system is different, and may have different requirements, hence the API should allow the developer to make decisions regarding data reporting which are right for their specific case. It would also limit the portability of the API, as a more automatic system would likely need to hook into the lower level robot drivers, meaning much greater changes would be necessary to port the code to a different robot. Furthermore by allowing the user to have complete control over when and how frequently data is transmitted they are able to manage the amount of traffic they are putting on their network, potentially mitigating congestion issues with very large swarms. Finally allowing users to control the moments within the code when data is reported to the debugging system was deemed to be a more intuitive system, especially as many developers are already familiar with the concept of using 'print' statements within code to display debugging messages in a console.

The *User Guide* for the system, found in appendix E, describes how to incorporate the robot side API into a robot's behavioural code.

6.12 Summary

This chapter began with a discussion of the choice of application programming framework for implementing this application, establishing the reasons for choosing the *Qt* framework. Following this key decision the implementation of the application is then described, including the overall structure of the code, and details of the functioning of each key component. The implementation of code for use on the robots is also discussed. This chapter summarises the work completed for the implementation portion of the project plan, outlined in chapter 4.

Chapter 7

Testing

Software testing is the practice of defining a number of processes which can be applied to a piece of software in order to verify its correct operation. Each of these processes may involve a number of steps, and is usually described as a single test. The results of each test are normally compared against some expected, correct outcome, and used to determine if any bugs or issues remain in the software. It was important to thoroughly test the software in order to verify its correct implementation and operation, and identify any remaining issues that needed to be fixed or mitigated. Some testing was done throughout the implementation phase, after each feature addition, and this testing is referred to here as continuous integration testing. More rigorous testing processes were then applied once the implementation was complete, including testing of both the user interface and the system back end, as well as testing the system as a whole. This section gives details of the different testing processes applied to the system, the results obtained, and details of any resulting fixes that were implemented.

7.1 Continuous Integration Testing

The purpose of the continuous integration testing was to verify the correct operation of individual components as they were completed, and to ensure that different components would work correctly together. This was done during development to reduce the risk of issues stacking up and becoming layered or entrenched as development continued. The modular design of the software made the continuous integration testing process relatively simple. After a software component or a specific functionality was implemented, it was tested informally by supplying relevant data or control inputs and verifying the correct result. The existing modules and functionality were then also re-tested. This helped to check that the newly implemented functionality had not created an issue elsewhere.

Where possible each module was also tested in isolation, prior to being connected to any related modules, to avoid issues becoming compounded. For example

the *MachineVision* class was tested alone, by verifying that it could retrieve a single image from the camera and track the robots in said image, before it was connected to the higher level *CameraController* component. Another example was the testing of the networking code in isolation, which involved using an external tool to supply individual UDP packets, and verifying that they were correctly received by the *DataThread* class by outputting their raw contents.

The continuous integration testing was completed in a relatively informal manner, as the main focus at the time was implementation, however it was still a useful technique and helped to ensure that bugs were caught early when possible. For a larger, more complex application, being developed by a team rather than an individual, integration testing should be applied in a more formal manner. For this application however it was deemed more important to focus on the actual implementation, due the relatively limited time available.

7.2 Manual User Interface Testing

The purpose of any graphical user interface is to present information to a human user and collect their input. It is therefore important that user interfaces be tested manually by a human user, as automated testing methods are often not sufficient for, or not capable of, verifying that information is displayed legibly and correctly, and that user input functions properly. The user interface of the system is one of the most important parts of the project, and therefore a thorough manual user interface testing process was undertaken.

7.2.1 Method

The general approach taken to the user interface testing can be summarised as follows:

1. Separate UI into individual elements.
2. State the purpose and required functionality of each element.
3. Define a general test strategy for an arbitrary single UI element as a series of checks.
4. Identify any special case components, and define different test strategies where necessary.
5. Apply the relevant strategy to each interface element in turn.

The user interface was separated into the elements described in table 7.1. Special case elements requiring different test strategies are highlighted in bold.

Element	**Test Strategy**
Visualiser Panel Tab System	A
Visualiser	**B**
Visualiser Settings Tab	A
Camera Settings Tab	A
Robot List Panel Tab System	A
Robot List Element	A
Network Settings Tab	A
Logging Settings Tab	A
Data Panel Tab System	A
Console Data Tab	A
Overview Data Tab	A
State Data Tab	A
IR Data Tab	A
Custom Data Tab	A
Individual Visulisation Settings Dialogs	A

TABLE 7.1: Individual user interface elements that required testing, and the test strategy applied.

The following test strategies were then devised for the different element categories.

Test Strategy A - Standard UI Elements:

1. Examine UI element visually. Verify that it appears correct. Verify that it contains all elements necessary to satisfy its purpose.
2. Examine all text within the element. Check for errors in both meaning and spelling.
3. Verify that all components within the element which perform actions in response to user input operate correctly.
4. Verify that all components respond quickly to user input.
5. Verify that component actions and functionality do not degrade with extreme use (sustained rapid input, large numbers of input changes, extreme or out of range values, etc).
6. Verify that all data displayed within the element is visible, readable, correctly arranged and correctly labelled.

7. Verify that the element behaves sensibly when window resizing occurs, and that it remains usable and data remains visible whenever possible.
8. Verify that the element updates promptly when responding to changes in data.

Test Strategy B - Visualiser:

1. Verify that the video image is displayed correctly.
2. Verify that user clicks within the visualiser space are located correctly, at a number of different window sizes.
3. Verify that robots can be selected by clicking on their location in the visualiser image.
4. For each data visualisation type:
   (a) Define a set of input data and the expected representation of this data.
   (b) Supply the input data.
   (c) Verify the representation is as expected.
   (d) Check that the visualisation is clear and any text is legible.
   (e) Repeat for multiple sets of input data.
   (f) Verify that integrity is maintained with extreme data, corner cases and zero data, wherever possible.
   (g) Verify that integrity is maintained at a range of window sizes, within reasonable limits.

7.2.2 Results

For each user interface element the appropriate test strategy, as specified in table 7.1, was carried out. The results of this testing can be found in appendix F.1. The testing highlighted no major problems with functionality, but did identify a number of smaller issues, mostly related to aesthetics and usability.

7.2.3 Fixes Implemented

The following fixes were implemented to address the issues identified during the manual user interface testing:

- The settings tabs in the visualiser panel were renamed to better reflect their purpose.

- Long directory paths in the logging tab were truncated to only display the final 25 characters.
- The ordering of messages in the console was reversed, to read from top to bottom in the order that they occurred.
- The redundant heading was removed from the overview tab.
- Entries in the state transition list were reformatted to clearly separate the time-stamp from the states.
- The IR data tab was reorganised to be more space efficient, and display data more clearly. Headings and labels were added to further improve clarity.
- Code was added to flip the video image, and a setting was added to the camera settings tab to enable and disable this feature.
- Robot IDs rendered in the visualiser were positioned slightly closer to the robot's horizontal position.
- The IR data visualisation in proximity mode was adjusted to have a shorter maximum line length, and to use a non-linear mapping to improve the proximity approximation.
- The IR data visualisation in heat mode was adjusted to use white as the base colour for clarity, changing to red and increasing in size with increasing sensor value.
- An upper limit was added to the robot path sampling interval.

7.3 Data Model and Back End Unit Testing

The next large code component requiring testing was the data model, which formed the majority of the application back-end code. Testing the data model manually, by inputting data packets and verifying the correct insertion of data into the model, was deemed to be too time consuming. This manual approach also posed another problem, in that the contents of the data model could only be examined through the user interface, which meant that the data model testing would be inherently coupled to the UI. This would make it harder to determine the source of any bugs found, as they might be related to the data model or the UI. Avoiding this kind of interdependency is part of the reason for adopting an object oriented approach to the software design and implementation. In order to avoid this issue, the back-end was tested using an automated, 'unit-testing' based approach.

Unit testing is a commonly used technique in professional software testing [46], and involves writing specific functions or scripts which test individual 'units' of

code in an automated fashion. These test cases will manipulate the unit in some way, using the data and functions it exposes. Then the test case will assert a fact that should be true after the manipulation, such as a comparison of a data point within the unit and the value it should have following the given operations. Each test might include many of these assertion statements, and the test only passes provided that all assertions evaluate to be true. In order to apply this technique to this system, an extra class - '*TestingWindow, testingwindow.cpp / .h*' - was added to the application. Alongside this class a number of individual test case functions were added. Each of these functions was written to test the data model in a specific way. Table 7.2 lists the test case functions and their purposes.

Test Case	**Purpose and Method**
Robot Insertion Test	Tests whether data objects for new robots are inserted into the model correctly. Supplies a packet of each possible type, using a new robot ID each time. Asserts after each packet that the number of robots stored in the model has increased by one. Supplies another set of packets, this time reusing the existing robot IDs. Asserts after each packet that the number of robots stored in the model has not increased.
Name Data Test	Tests whether data describing the name of a robot, received in watchdog packets, is inserted into the data model correctly. Supplies three watchdog packets, each with a different robot ID and robot name. Asserts that the data model now contains three robots, and that their name data matches the names entered in the packets. Supplies three new watchdog packets for the same set of robot IDs, with different names. Asserts that the data for each of the robots has been updated to include the new name data.
State Data Test	Tests whether data describing the current state of a robot, received through state packets, is inserted into the data model correctly. Supplies three state packets, each with a different robot ID and state. Asserts that the data model now contains three robots, and that their state data matches the states entered via the packets. Supplies three new state packets for the same set of robot IDs, with different states. Asserts that the state data for each of the robots in the model now matches the new states.

Position Data Test	Tests whether packets describing the current position and orientation of a robot are correctly parsed and the data stored correctly in the data model. Supplies three position packets, each with a different robot ID and different position values. Asserts that the data model now contains data for three robots. Asserts that this data matches the values in the packets for x-position, y-position and angle individually. Supplies three more position packets for the same set of robot IDs with new position and angle values. Asserts that the data in the model for each robot has been updated to reflect the new values. Assertions involving floating point numbers are done using a tolerance comparison, with a tolerance of 1×10^{-7}.
IR Data Test	Tests whether packets describing a robot's infra-red sensor values are correctly parsed and the data correctly stored in the data model. Supplies an IR data packet, with each sensor reading containing a different value. Asserts that the IR data in the model matches each of the values in turn. Supplies another IR data packet with new, unique values. Asserts that the IR data in the model now matches each of the new values. This process is repeated for packets of background IR data type.
Custom Data Test	Tests whether packets describing custom data key/value pairs are correctly parsed and the data correctly stored in the data model. Supplies three custom data packets, each with a different robot ID and a different value, for a single key. Asserts that the data model now contains data for three robots, and that each robot has a custom data entry for the given key. Asserts that the value for each of these entries matches the value supplied in the relevant packet. Supplies three new packets for the same set of robots with a new key and a new value. Asserts that each robot now contains custom data for the second key, and that the values match those supplied in the packets. Supplies three more packets using the original key, with new values. Asserts that the values for the original key for each robot have been updated to match the values in the latest packet set.

Position History Test	Tests whether position data supplied for a robot is correctly sampled and stored in the position history portion of the data model. Sets the position history sampling rate to 2. Supplies twenty position packets, all attributed to the same robot ID. Asserts that the position history now contains ten entries. Asserts that the x- and y-position values for each of these entries match the values in every second packet, in reverse order. Assertions involving floating point numbers are done using a tolerance comparison, with a tolerance of 1×10^{-7}.
State History Test	Tests whether the state transition history data is correctly formed from a sequence of state packets. Supplies five state packets, each attributed to the same robot ID and containing different states. Asserts that the state transition history now contains five entries. Asserts that the entries describe the correct state transitions, as described by the packets, in reverse order.
Bad Data Test	Tests whether badly-formed data packets are correctly rejected by the data model. Supplies a number of correctly formed data packets, each with a different robot ID, and asserts that the data model now contains the correct number of robot entries. Supplies a number of invalid and malformed data packets, distributed between the already used robot IDs and several new IDs. Asserts that the number of robots in the model has not changed, and that the data in each of the existing robots has not changed.

TABLE 7.2: Test cases used to unit-test the data model.

The *TestingWindow* class displays an additional user interface window, which can be opened by selecting the '*Testing Window*' option from the developer menu on the main tool-bar of the application. This window displays a list of the available tests, a text area for displaying test results, and controls for running either a single test or the full set. Each time a test is run the class instantiates a new data model object, performs the operations for the test in question, displaying the steps involved as text in the results window. For each assertion the text describes what is being asserted and states whether the result was true or false. The overall result of each test is then stated at the end, and the data model object is destroyed. This therefore allows any developer working on the system to open this window at any time whilst the application is running and verify that the full set of tests still passes. This can be

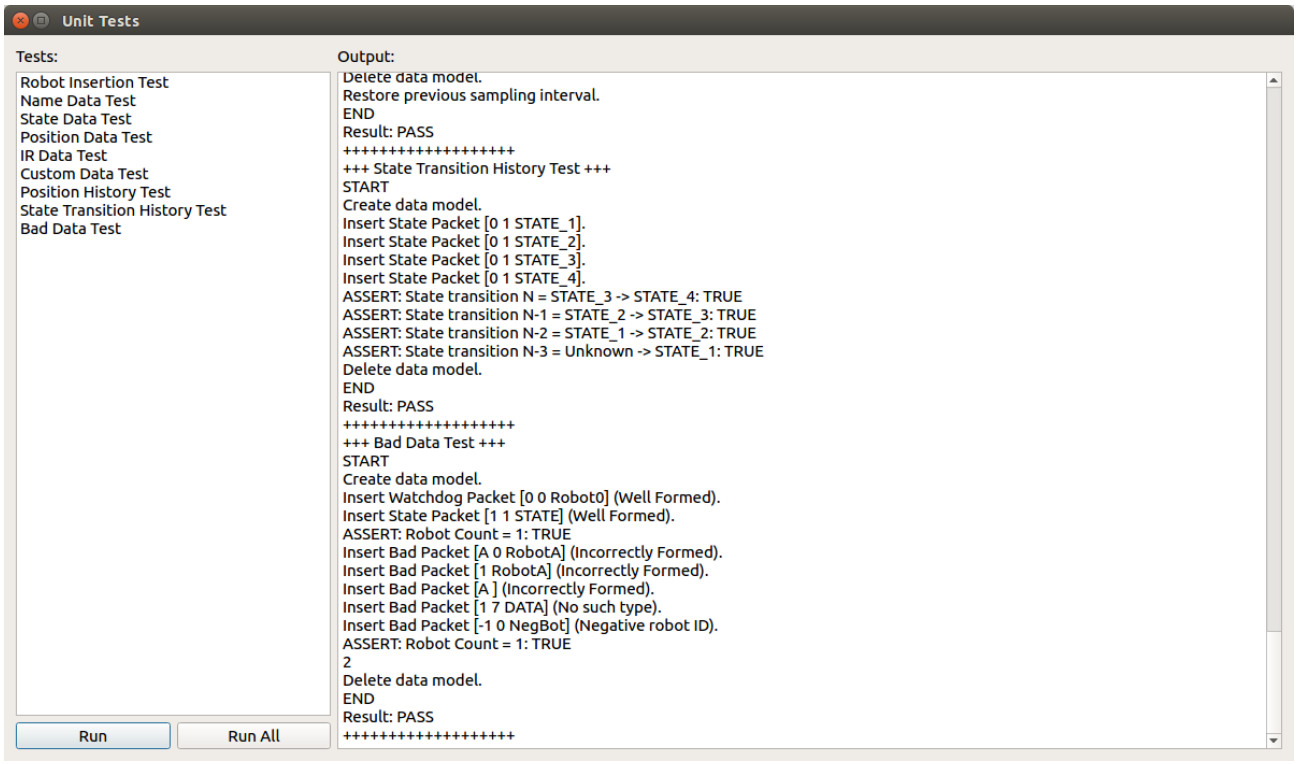


FIGURE 7.1: The testing window, used for running the data model unit tests.

done whenever changes are made to the back end code, and gives the developer a degree of certainty that the data model is still functioning correctly. New tests can be easily added by implementing a new test function and adding it as a test case. The interface for this test window is shown in figure 7.1.

7.3.1 Results

In its final state following this project the data model was implemented such that all of the stated test cases passed successfully when run. This indicated that the data model was implemented to a satisfactory standard, and that if data was supplied in correctly formatted packets, it would be correctly stored in the model. Other application functionality that relied on the data model could therefore be used and tested with the assumption that the model was operating correctly.

7.3.2 Issues with this Approach

The unit testing approach suffers from a number of issues. First and foremost the results of the tests are only as good as the tests themselves - meaning that any bugs in the code of each test could be misinterpreted as bugs in the software itself. To mitigate this the tests are designed to be as logically simple as possible, and perform

the smallest number of operations necessary to achieve the behaviour under test. This minimises the chances of a mistake in the implementation of the test code.

This approach also relies on the developer writing the tests correctly determining, and correctly entering, the required result for each assertion statement. One example of this issue is in floating point comparison. The accuracy with which a floating point variable can describe a number is inherently limited due to the way a floating point variable is constructed. In almost all cases a floating point number will differ from the exact value it is attempting to represent by some small amount. This can present an issue when performing comparisons between the contents of a floating point variable and an exact value, leading to false negatives. In order to avoid this all floating point comparisons have been done using a threshold, rather than a direct comparison. This technique asserts that when the expected value is subtracted from the variable being tested, the modulus of the result is less than some very small tolerance value, indicating that the value in the variable is within an acceptable range of the expected value. A tolerance value of 1×10^{-7} was used in all test cases, as this was deemed to indicate sufficient accuracy for the data in this system. To give some perspective to this number, note that the x-position value of a robot is stored as a floating point value between 0 and 1, representing a portion of a physical distance of approximately two and a half meters. A discrepancy of +/- 1×10^{-7} therefore equates to an error of $0.25mm$. Hence this tolerance value indicates more than adequate accuracy.

7.4 Verification and Validation Testing

Verification and validation (V&V) testing [47] is the process of determining whether the software implemented meets the high level requirements, and whether the individual functionalities operate correctly and without bugs. This was the last step in the testing process, and served to test the system as a whole. The approach taken was to consider the requirements of the system, as stated in the functional specification in section 1.5, and determine whether each requirement is satisfied by the functionality implemented. Where possible each functionality was tested with relevant data and control inputs. The requirements, and the tests carried out for each are given below.

Core Requirements

C1. Must comprise a PC application.

- Verify the software runs on a PC and/or server.
- Verify that the software has the general appearance of a software application.

C2. Must be capable of receiving data related to the state of multiple robots.

- Program a number of robots to send state data through the system, and verify that it is received and displayed by the application.

C3. Must be capable of receiving positional data for the same set of robots.

- Program the robots to move along a number of different paths in view of the system, and verify that their positioned are correctly tracked and displayed by the application.

C4. Must be capable of receiving a live video feed of the robots in their environment.

- Run the application and verify that the video feed is displayed.
- Introduce a number of robots and objects to the environment space and verify that they are seen in the video feed.

C5. Must collate received data and present it to the user in a combined graphical form.

- Program the robots to report a number of different data types, including sensor data, and verify that this can all be displayed by the application simultaneously.

C6. Must present auxiliary, non-spatial data to the user in textual or other forms.

- Program the robots to report name, state and custom data and verify that this is correctly received and displayed in the application.

C7. Must update in approximately real time.

- Program the robots to report data periodically, and verify that the data within the application updates in real time.
- Program the robots to move around in view of the system, and verify that the video shows their movement in approximately real time.

C8. Must at minimum support the e-puck robot platform.

- The previously stated tests should be performed using e-puck robots.

Secondary Requirements

S1. Should use a modularised structure.

- Verifying this through standard testing is not feasible. See section 6.3 for details of the modular implementation.

S2. Should exchange data between the robot platform and the application using a platform-agnostic, extensible protocol.

- Use a platform other than the e-puck robot to transmit packets to the application. Verify that the packets are received and interpreted correctly regardless of the source.

S3. Should provide a basis for interoperability with a number of robotics platforms.

- Verifying this requires another robot platform to be integrated into the system. This is outside the scope of the project. However sections 6.8 and 6.11 indicate how the system has been implemented with portability in mind, and suggest that interoperability should be feasible and relatively easy to achieve.

S4. Should allow the user to configure the displayed data.

- Program the robots to report data of all supported types. Verify that the visualiser settings allow for display configuration.

S5. Should employ a model-view-controller (MVC) software architecture.

- Verifying this through standard testing is not feasible. See section 5.1 for details of the MVC based architecture design.

S6. Could provide the user with ways to configure and display custom data types.

- Program the robots to report a variety of textual and numerical custom data. Verify that this data is received, stored and displayed correctly.
- Verify that the custom data overlay can correctly display any of the received data.

S7. Could allow the user to compare data on two or more specific individual robots.

- Due to time constraints, and a lack of interest in this feature during the initial survey, it was deemed low priority, and ultimately not implemented.

S8. Could calculate and display swarm-level meta-data and statistics.

- Verify that the average position of the tracked robots is correctly displayed.

S9. Could generate log files of robot activity over a user defined period.

- Program the robots to periodically report data of various types. Verify that pressing the start logging button, and then pressing it again (to stop the logging) some time later, results in the generation of a log text file containing entries for all data received in the time period between the presses.

- Verify that setting the log file path directory using the user interface correctly affects where the log file is saved.

7.4.1 Results

Test	Result
C1	The application was run successfully on two machines running Linux operating systems. The application has an appearance consistent with the familiarly understood definition of a software application.
C2	Data of all defined types was successfully sent from six robots to the application. The data was correctly displayed within the application.
C3	The positions of six moving robots were correctly tracked simultaneously. The tracking data obtained was correctly displayed in the application both numerically and visually.
C4	The application correctly displays the input video feed. Objects placed in view of the camera are seen in the video feed within the application.
C5	Data received from the robots, including sensor data, was correctly converted to visual representations and displayed in the application.
C6	Data received from the robots was correctly received and displayed in a textual form within the application.
C7	The application updated the data displayed to match new data from the robots in approximately real time. The application updated the position and orientation data to match the movement of the robots in approximately real time.
C8	E-puck robots used for all core tests involving robots. The e-puck platform is sufficiently supported.
S2	Utility program used to send specific UDP packets to the application via the network, mimicking the behaviour of an arbitrary robot. The system received and interpreted the packets correctly.
S4	The visualisations of the data reported by six robots could be configured using the options available in the visualiser settings menu, to a moderate extent. Full customisation of all aspects of the visualisations was not possible.

S6	Custom data sent from six robots was received and displayed within a table correctly, updating in approximately real time. The custom data visualisation could be set to target any of the specific data points received. The target data point was then correctly displayed, overlaid on the video feed. Configuring the display of custom data beyond a textual representation was not possible. Displaying more than one target data point within the visualiser was not possible.
S8	When enabled, the average position display correctly displayed the average position of the currently tracked robots as a mark within the video feed, when tested with three, four and six robots. This data was not provided in numerical form. Other swarm-level data was not provided.
S9	Was able to correctly start and stop logging whilst receiving data from six robots. The generated text file contained appropriate information. The layout of this information was, however, difficult to parse, making the data hard to understand and use. The directory to store the log files could be correctly set using the UI controls.

TABLE 7.3: Verification testing results.

7.4.2 Analysis

All core requirement tests, C1 through C8, passed satisfactorily, indicating that the application satisfies the core requirements. Secondary requirement test S2 passed satisfactorily, without caveats. Verifying secondary requirements S1, S3 and S5 was not possible using standard testing approaches. Secondary requirement S7 was dropped following the results of the initial user survey, and a reassessment of the feature set. Tests S4, S6, S8 and S9 all passed conditionally, as the requirement was either partially met, or met to a limited standard. In order to fully satisfy these secondary requirements the following developments could be made in the future:

- S4: Further develop the visualisation configuration options to allow the user to fully configure all visualisations, including sizes, offset positioning, font sizes, line widths, etc.
- S6: Develop the visualisation of custom data to allow the user a number of graphical options. Allow the user to visualise more than one custom data point simultaneously.

- S8: Add a new tab to the data panel for swarm-level data acquired through analysis. Display the average position values in this tab. Define and implement a number of other swarm-level data analyses, and display this data within the tab, as well as graphically within the visualiser.
- S9: Edit the log file generation to produce more structured files, using a comma-separated-value (CSV) format or similar.

7.5 Summary

This chapter has described the methods used to test the application software and verify its correct operation. The process of manual user interface testing was discussed, and the test strategies for this application defined. The results of these tests were presented and the resulting fixes described. The use of unit-test style code built into the application for testing the data model was also discussed. The results of the verification and validation testing were used to verify that the software satisfied the functional specification defined in section 1.5. Overall these tests confirmed that the application functions correctly, and only minor fixes were required.

Chapter 8

Evaluation

This chapter contains an evaluation of the swarm debugging system produced during this project, and the execution of the project itself. The first portion of this evaluation is derived from the results of a number of '*user evaluation sessions*' which are discussed in section 8.1, and aimed to determine the extent to which the system is useful and usable in its intended context. The second portion is based on a comparison of the system produced with the aim and objectives of the project, as stated in section 1.4, and a comparison of the execution of the project with the project plan described in chapter 4.

8.1 User Evaluation Sessions

Software testing can be used to verify that a system works as intended, however this does not guarantee that it is useful in practice. The purpose of the user evaluation sessions was therefore to determine whether or not the system satisfied its intended purpose of aiding in swarm robotics debugging. In order to do this, potential system users were asked to use the system to complete a number of tasks, and data was recorded regarding their experience. This data was obtained through direct observation of the participants' interactions with the system, and through a follow up questionnaire designed to gauge their opinions on the system's various features and functionality.

One of the stated objectives of the project was to build a system with a clear and intuitive user interface, hence a secondary purpose of the user evaluation sessions was to determine whether or not the user interface was intuitive to use and easy to navigate, without prior knowledge. It was therefore necessary to select participants who had no direct involvement in the development of the system, and were therefore not biased by an existing understanding of the user interface.

The user evaluation sessions followed an observed testing format. Participants were given full control of the system, which was initialised to a known state, and

asked to complete a number of tasks, ranging from simple information and data location and retrieval to a full simulated debugging task. Data was collected by taking notes on how the participants interacted with the system, including any difficulties they had, and any comments they made whilst using it. The primary areas of interest for these notes were how easily the participant was able to navigate the user interface and obtain information, how often they asked for additional information and what they asked for information regarding, as well as if (and how quickly) they were able to isolate and correct the deliberate behavioural fault during the debugging task, and what features of the system they made use of during their attempt.

8.1.1 Participants

Two groups of participants took part in the user evaluation sessions. The first was composed of '*domain experts*'; researchers and other technical people with experience in the field of swarm robotics. This group were more likely to have an understanding of the needs of the system's target users, and therefore their input was considered more valuable. However because of the relatively niche nature of the field, the availability of participants in this category was limited. Hence only a small number could be found to participate in the sessions.

In order to remedy this lack of participants, the second group was introduced. This group was composed of undergraduate engineering students, who had a basic understanding of the core concepts of swarm robotics, but little or no practical experience. Where necessary an introduction to the core concepts was given before the session. This group did however possess general knowledge regarding software development, as this was necessary to complete the sessions effectively. The results obtained from this group were therefore mostly useful to evaluate elements of the system unrelated to swarm robotics, such as the usability of the user interface and the clarity of information display. This difference is reflected in the analysis of the results.

8.1.2 Set Up

Four robots were used during the sessions, each loaded with the same behaviours. Two simple behaviours were programmed for the robots prior to the start of the sessions. The first was a very simple data test behaviour which had the robot drive slowly in a circular path, whilst reporting data for all of the types defined in the system. This included a watchdog packet every ten control steps containing a unique name for each robot, a varying state which would change every fifty control steps between three possible values, active and background IR data, an arbitrary log message packet every hundred control steps and a custom data packet every ten control steps, containing the current total number of control steps as its data.

The second behaviour was a simple dispersion behaviour, which would cause the robot to turn and drive away from any nearby object. This was achieved by monitoring the values of the robots IR sensors, calculating a vector of highest IR reflection intensity based on these values, and negating it to acquire a heading vector in the opposite direction. Sensor values were only included in the reflection intensity vector calculation if they surpassed a specific threshold, in an attempt to eliminate the effects of random noise. Whilst executing this behaviour the robot was also reporting debugging information to the application. This included watchdog packets, active and passive IR data, and custom data as in the previous behaviour, as well as the state of the robot which would vary between *IDLE* and *MOVING*.

In order to simulate a bug in the robot's behaviour code, the IR value threshold in this second behaviour was deliberately set too low, causing the robots to move randomly when no objects were nearby, as a result of the fluctuations in the IR readings due to noise. A further consequence of this was that the robot would rarely, if ever, enter the *IDLE* state. Participants were then asked to attempt to locate the cause of this bug using the debugging system.

Prior to the start of each session the robots were returned to their initial state, with the two behaviours loaded and the first behaviour running. The application was not set up in any way, and was run fresh for each session. It was important that this initial set up be the same for each participant, in order for the results to be comparable.

8.1.3 Session Sequence

The following sequence of steps was then carried out for each session. Participants were deliberately not told precisely where to find specific information, or exactly how to complete each task, in order to observe the extent to which the user interface was intuitive. Participants were asked to say if they could not determine how to complete one of the tasks, at which point further guidance was given. Relevant observational notes were taken at each step.

1. Introduce the application to the participant, and have them run the executable.
2. Explain the purpose of the application, and the purpose of this evaluation session.
3. Indicate the key features within the application. Specifically identify the visualiser and explain its purpose, as well as the robot list panel and the data panel.
4. Ask the participant to start the application listening for data packets on the network. Mention that this functionality can be found on the network tab only if necessary.

5. The simple circular motion controller code should be running on the robots already. If it is not, run it now.
6. Ask the participant to obtain the following pieces of information:
   (a) The current state of robot 1.
   (b) The recent state changes of robot 2.
   (c) The current IR sensor values of robot 3.
   (d) The current value of the 'ControlStep' custom data point for robot 4.
   (e) The set of known states for robot 1.
   (f) The numerical angle describing the orientation of robot 2.
7. Ask the participant to use the visualiser settings tab to achieve the following:
   (a) Hide the name and state visualisation for all robots.
   (b) Display the recent path for all robots.
   (c) Display the IR sensor data for the selected robot in heat mode.
   (d) Change the IR sensor display to show in proximity mode, and for all robots.
   (e) Hide the position and orientation visualisation for all robots.
   (f) Display the custom data point 'ControlStep' for the selected robot.
8. Explain briefly the potential for discrepancies between ArUco tag IDs and robot IDs. Ask the participant to add a mapping from ArUco tag ID 2 to robot ID 10. Direct them to the camera settings tab only if necessary.
9. Ask the participant to use the logging tab to achieve the following:
   (a) Set the logging directory to a specific folder.
   (b) Start logging for a short period of time and then stop it.
10. Stop the circular motion behaviour on the robots and switch to the dispersion behaviour.
11. Make the dispersion behaviour source code available to the user.
12. Explain the desired behaviour to the participant, and the current erroneous behaviour.
13. Ask the participant to attempt to locate the cause of the issue using a combination of the data made available by the debugging system, and the source

code. Provide hints and advice only if necessary. Make specific notes of the following:

(a) Does the participant use the state tab in the application to identify that the robot is not correctly entering the IDLE state?

(b) Does the participant use the IR data tab in the application to examine the IR sensor values?

(c) Does the participant make use of the visualiser settings to change the data that is displayed visually?

(d) Is the participant able to correctly identify the low threshold value as the cause of the bug?

8.1.4 Observations

The following is a summary of the observations made during the evaluation sessions and the comments made by the participants. Special note was taken of any observation made during more than one session or comment made by more than one participant. Steps in the session for which no comments are noted can be assumed to have been completed without issue by all participants.

All participants were able to quickly find the network tab and enable packet listening, at which point the system began receiving data from the robots. All participants were also able to easily locate all of the requested information. The only exception to this was locating the current state of a given robot, which three of the participants commented was not clearly displayed on the state tab. The state tab displays other state related information, but does not clearly display the current state (it can be inferred from the state transition list), which was an oversight in the user interface design. All participants were able to subsequently locate the current state on the overview tab, and within the visualiser.

All participants were able to locate the visualiser settings tab and determine how to enable and disable visualisations. Three of the participants were initially confused by the fact that the visualiser settings were unrelated to the selected robot, and were instead applying globally. Five participants also commented that the process of double clicking an item within the visualiser configuration list, to access the more specific settings, was unintuitive. Several commented that this was not a user interface behaviour they had seen before, and therefore was not immediately obvious. Suggested alternatives were to use drop-down menus, a right click menu, or a 'tree' style expandable list, to access detailed settings for each visualisation. One participant also commented that the detailed settings for the IR data visualisation were not clearly described, especially regarding the two possible modes; proximity

mode and heat mode. They commented that these would be better presented in a drop-down selector.

During the simulated debugging task all participants made use of the visualiser settings to configure the way that data was displayed, and used the visualiser display when attempting to determine the cause of the issue. Three participants adjusted the visualiser settings more than once. Five participants used the state tab to identify that the robots were incorrectly remaining in the *HEADING* state, and not moving to the *IDLE* state as the code suggested they should. All six participants used the IR data tab to check the values being reported by the robots' IR sensors, and all six were able to identify that the cause of the bug was the low threshold value. As expected the members of the domain expert participant group were quicker to locate the simulated bug than those in the non-expert group. One participant from the non-expert group explicitly noted that familiarity with the robots and their basic capabilities would have made the task significantly easier.

All three members of the domain expert group commented on their belief that the system would be useful, based on their prior experience with swarm robotics. One participant specifically mentioned the system's applicability to their current work, and noted a number of scenarios where the custom data reporting functionality would be useful, including reporting 'pheromone' levels in a swarm path-finding behaviour. Another member of the expert group commented that they liked the idea of re-using the system with a different robot platform, and noted that the use of the *ArUco* tracking tags would improve the ease of the porting process. The three members of the non-expert group each identified the IR and state data reporting as useful in completing the simulated debugging task.

8.1.5 Questionnaire

Following the practical session, participants were asked to complete a questionnaire regarding their experience, and their opinions of the system and its features. The questionnaire included the following questions:

Question 1: How would you rate your overall impression of the user interface? (Scale of 1 to 5, 'poor' to 'excellent')

Group	1	2	3	4	5	Average Score
Domain Experts	0	0	0	1	2	4.7
Others	0	0	0	2	1	4.3

TABLE 8.1: The responses to question one of the user evaluation questionnaire.

Question 2: How intuitive was the robot selection mechanism?
(Scale of 1 to 5, 'not intuitive at all' to 'highly intuitive')

Group	1	2	3	4	5	Average Score
Domain Experts	0	0	0	0	3	5
Others	0	0	0	1	2	4.7

TABLE 8.2: The responses to question two of the user evaluation questionnaire.

Question 3: How easy was it to locate the requested information?
(Scale of 1 to 5, 'very hard' to 'very easy')

Group	1	2	3	4	5	Average Score
Domain Experts	0	0	0	0	3	5
Others	0	0	0	1	2	4.7

TABLE 8.3: The responses to question three of the user evaluation questionnaire.

Question 4: How would you rate the organisation of information in the lower data panel?
(Scale of 1 to 5, 'poorly organised' to 'well organised')

Group	1	2	3	4	5	Average Score
Domain Experts	0	0	0	1	2	4.7
Others	0	0	0	1	2	4.7

TABLE 8.4: The responses to question four of the user evaluation questionnaire.

Question 5: For each of the following data types, how would you rate their presentation within the visualiser, in terms of information clarity?

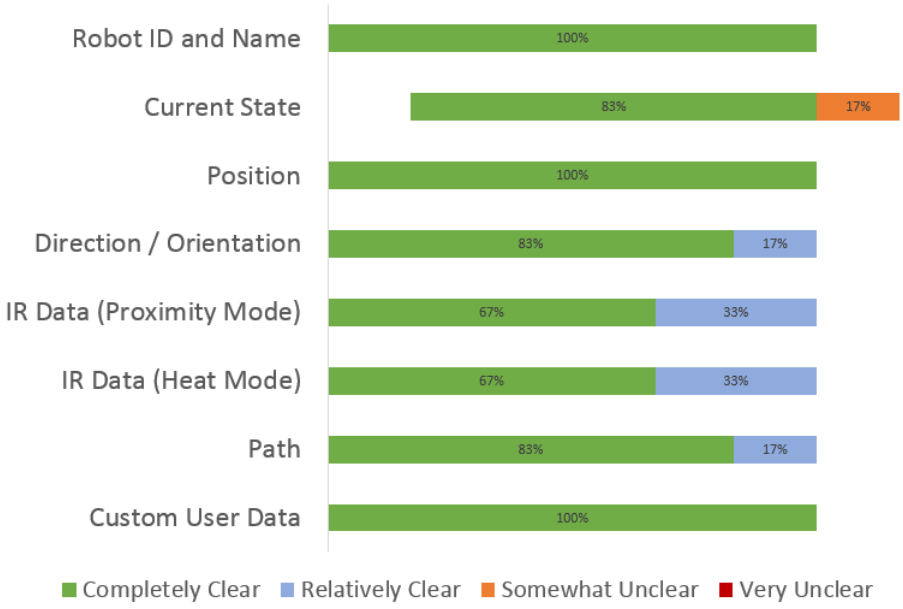


FIGURE 8.1: The responses to question five of the user evaluation questionnaire.

Question 6: How easy was it to adjust the visualiser settings as requested?
(Scale of 1 to 5, 'very hard' to 'very easy')

Group	1	2	3	4	5	Average Score
Domain Experts	0	0	0	1	2	4.7
Others	0	0	0	1	2	4.7

TABLE 8.5: The responses to question six of the user evaluation questionnaire.

Question 7: How beneficial do you believe the visualiser component to be to the overall application functionality?
(Scale of 1 to 5, 'not beneficial at all' to 'essential')

Group	1	2	3	4	5	Average Score
Domain Experts	0	0	0	0	3	5
Others	0	0	0	1	2	4.7

TABLE 8.6: The responses to question seven of the user evaluation questionnaire.

Question 8: How easy / clear was it to use the network tab to begin listening for robot data packets?
(Scale of 1 to 5, 'very difficult / unclear' to 'very easy / clear')

Group	1	2	3	4	5	Average Score
Domain Experts	0	0	0	0	3	5
Others	0	0	0	0	3	5

TABLE 8.7: The responses to question eight of the user evaluation questionnaire.

Question 9: How easy / clear was it to use the logging tab to record received data to a log file?
(Scale of 1 to 5, 'very difficult / unclear' to 'very easy / clear')

Group	1	2	3	4	5	Average Score
Domain Experts	0	0	0	1	2	4.7
Others	0	0	0	0	3	5

TABLE 8.8: The responses to question nine of the user evaluation questionnaire.

Question 10: How would you rate your overall impression of the system's usefulness as a debugging tool?
(Scale of 1 to 5, 'not useful at all' to 'extremely useful')

Group	1	2	3	4	5	Average Score
Domain Experts	0	0	0	0	3	5
Others	0	0	0	1	2	4.7

TABLE 8.9: The responses to question ten of the user evaluation questionnaire.

Question 11: Please feel free to add any additional comments regarding any aspect of the system.

- *"Drop down menu to choose single robot/all robots would be a nice addition. More visual "heat mode" display against dark background would also be useful."*
- *"This seems like an extremely useful addition to the system. I can see how debugging without this application would be much harder and take much longer."*

- "*Exceptionally useful system with intuitive and easy to use interface; shows a high level of polish and thought in UI design. Naturally some areas could have subtle improvements but nothing major!*"
- "*Really intuitive to use and I can already think of multiple use cases where this will be beneficial for robot lab researchers. Updated quickly and software seemed to run seamlessly. Ability to monitor custom data will be incredibly useful too. Loved how data was displayed in bar charts in the bottom panel and the heat mode display of the IR sensors. Excellent way to display information visually for quick diagnosis of problems. Literally only thing I found that wasn't immediately intuitive was the additional settings on the Visualizer Settings.*"

8.1.6 Analysis

A general appraisal of the observations and comments made during the evaluation sessions, and the responses to the subsequent questionnaire, indicates an overall positive response to the system. Participants were able to easily navigate and use the majority of the UI, without detailed direction, indicating that the user interface is intuitive and clear. The following remedies for minor issues with the user interface should be applied:

- The interface for accessing specific visualisation settings should be changed to conform to a more common interface pattern. An expandable list would work well, with main list items for each of the visualisations, and sub-items attached to each for specific settings.
- The current state should be displayed clearly on the state tab, possibly by highlighting the current state within the known state list.
- The IR data visualisation mode selection should be changed to use a drop-down menu selector.

The participants made use of the system in a number of ways during the simulated debugging task, including examining state changes and IR data, and utilising the visualiser. Coupled with their comments, this gave the strong indication that the system was useful in completing the task. The system also received high scores for usefulness in response to question ten of the questionnaire. Comments from the domain expert group of participants regarding the potential use of the system in their swarm robotics work also reflected well on the system, suggesting that these participants believe they can benefit from its use. The specific use of the visualiser by the participants, and the high scores in response to question seven, indicate that the system benefits from the inclusion of the visualiser, and is more useful than a

data reporting system without a visual component. The combination of these factors leads to a reasonable confidence that the system would be useful if used in a real development and debugging scenario.

Overall the results of the user evaluation sessions were positive, suggesting that the system meets the two key objectives being examined in these sessions; being useful in a practical debugging context, and displaying information through an intuitive interface. However the limited sizes of the participant groups, and the simulated nature of the debugging task, restrict the extent to which conclusions can be drawn from this study. Further usage of the system by a larger number of researchers and in real debugging scenarios is necessary to get a more reliable idea of its value as a swarm debugging tool.

8.2 Comparison with Project Aim and Objectives

In order to evaluate the success of the project overall, this section compares each of the objectives stated in section 1.4 against the relevant project outcomes.

1. Utilise existing fiducial marker based tracking technology to track the position of individual robots within a swarm over time.

The system utilises the ArUco tag detection system and camera set up described in section 3.3 to track the positions and orientations of the robots in real time. This is enabled by the *CameraController* and *MachineVision* classes of the application, discussed in section 6.6. The ability of the system to accurately track the robots' positions was verified numerous times during the user interface testing process, including during the testing of the visualiser component's position and direction overlays, described in section 7.2, and in test C3 of the verification testing, described in section 7.4. This objective was therefore met successfully.

2. Develop code to allow multiple robots to communicate information regarding their internal state, sensor readings and decision making to a central application wirelessly via a network.

The system utilises a WiFi network to enable data transfer from the robots to the application. The code controlling the networking functionality of the application, enabling the receiving of data, is discussed in section 6.7. The code controlling the networking functionality on the robot side, enabling the sending of data, is discussed in section 6.11. Test C2 and C7 of the verification testing process, described in section 7.4, verify that the networking functionalities of both sides operate correctly, and that data is transferred correctly. This objective was met successfully.

3. Develop a data model that allows the central application to store information received from the robots, and update it as new information arrives.

The design for the application data model is presented in section 5.1.1. This design was then used during the implementation of the data model, which is discussed in section 6.5. The data model component was tested using a unit-testing style methodology, which is discussed in section 7.3, and the correct operation of the component was verified. The results of test case C7 and by extension test cases C5 and C6 in the validation testing process, described in section 7.4, verify that the data model functions correctly when used in conjunction with the other application components. This objective was therefore met successfully.

4 .Develop code to ascertain higher level data related to the robots, such as recent movement history or state transition history, and add this data to the model.

The application data model includes support for storing robots' recent movement and state transition histories, as discussed in section 6.5. The *Position History Test* and the *State History Test* in the data model unit testing campaign, described in section 7.3, verify the correct storage of these data points in the data model, hence this objective was met successfully.

5 .Design and implement a user interface which is easy and intuitive to use, and presents data from the data model to the user in a human readable manner.

The design of the user interface is discussed in section 5.2, and its implementation is discussed in 6.9. During both stages usability, readability, and the clear presentation of data were primary concerns. The manual user interface testing process, described in section 7.2, verified that the interface itself and all data displayed was readable, clear and sensibly arranged. The user evaluation sessions, described in section 8.1 went on to suggest that, among a small sample of both expert and non-expert users, the interface was intuitive and easy to use. Feedback regarding the interface was overwhelmingly positive, in spite of some minor issues being raised regarding some settings-related interface controls. Overall the testing and user evaluation results indicate that this objective was met successfully.

6. Develop a visual display component for the application, which presents the user with a live video feed of the robot swarm, augmented with relevant and spatially situated information relating to the robots, by fusing data obtained from the robots with data obtained from the tracking system.

The visualiser component satisfies the requirement for an augmented-reality style, graphical data visualisation component. The software design of the visualiser is discussed in section 5.1.2, whilst the design of the appearance of the visualiser and the individual data visualisations is discussed in section 5.2.1. The implementation of the visualiser component is discussed in 6.10. The visualiser portion of the manual user interface testing process, discussed in section 7.2, verified that the visualiser could correctly combine tracking data, robot data and the video feed to present spatially situated graphical and textual data representations, satisfying this objective.

Furthermore, the results and feedback from the user evaluation sessions, presented in sections 8.1.4 and 8.1.5, indicated that users responded positively to the graphical representations of data, and believed the inclusion of the visualiser component significantly benefited the application.

7. Develop the user interface in such a way as to allow the user to filter out information that is not currently relevant, and to compare and contrast information related to specific robots.

Information filtering within the visualiser component is achieved through the visualiser settings tab, discussed in section 6.10. For data outside the visualiser, filtering is achieved simply through selecting a target robot, and a specific data tab, using the interface described in section 6.9. The manual user interface testing process, described in section 7.2, verified that data filtering within the application worked correctly. During the user evaluation sessions, described in section 8.1, participants were able to correctly filter visualiser data displays when asked, and used this feature a number of times during the simulated debugging task, suggesting that the portion of this objective relating to information filtering was met successfully. The latter portion, relating to robot data comparison was not met, as this objective was dropped following the initial survey which indicated a low desire for comparison features among potential system users.

8. Design and implement the system in a modular way so as to allow for relatively simple integration with other swarm robotic platforms and tracking systems in future extensions.

Modularity is achieved within the system in a number of ways. Firstly the use of a standardised data transfer format as discussed in section 6.8, which is not specific to a single robot platform, ensures that data from any source can be interpreted correctly by the application. This was verified in test S2 of the verification testing process, discussed in section 7.4. The choice of WiFi as the networking technology, discussed in section 6.7, also helps to ensure portability due to WiFi's widespread use. The *CameraController* and *MachineVision* application components were also implemented in a modular fashion, as discussed in section 6.6, allowing for other camera drivers to be used with minor modification to the application source code. Furthermore tracking data is incorporated into the standard data transfer format, and can therefore be received via the network. Hence a developer using a different tracking system could implement code to send their tracking data to the application via the network, bypassing the *CameraController* or *MachineVision* components, and allowing completely different tracking hardware to be used. Standard testing procedures could not be used to verify the portability or extensibility of the system, however during the user evaluation sessions one participant from the domain expert group specifically remarked on the ease with which the system could be transferred to another robot platform, due to the use of the hardware-less ArUco tags for tracking,

and the non-robot-specific implementation of the robot side API for data reporting. Collectively these factors indicate that this objective was met successfully.

Having established that all of the stated objectives of the project were met, except for one which was dropped during requirements capture and problem analysis, the overall aim of the project can be considered. This was stated in section 1.4 as the following.

"**This project aims to understand the needs of a swarm robotics researcher or system developer when attempting to debug their system, and create a computer application which allows a user to monitor the state and behaviour of a robot swarm system in real time, employing augmented reality techniques where possible, and thus improving the ease and efficiency of the debugging process.**"

An understanding of the needs of a swarm robotics researcher was established through a review of relevant literature, presented in chapter 2, and through an initial survey of researchers with experience in the field, presented in section 3.1. The design and implementation of the application, which satisfy the creation portion of the project aim, are discussed in chapters 5 and 6. The effectiveness of the system, and its ability to improve the "*ease and efficiency*" of debugging tasks as required by the final portion of the aim, was assessed in the user evaluation sessions, described in section 8.1. Overall these sessions indicated that the application was successful in this regard. Considering this, and the successful completion of all objectives, this evaluation finds that the overall aim of the project has been met.

8.3 System Limitations

In spite of the project's success in satisfying the aim and objectives, and producing a working system, a number of limitations inherent to the system's nature have been identified and are summarised here.

Effect on robot behaviour. In order to receive data from the robots, the system requires that additions are made to the robot's controller code to report this data. Any time code is added to the robot's controller it has the potential to affect the behaviour in unforeseen ways, even if the added code has no direct effect on the code controlling decision making. This is because time is required to execute the data reporting code, and this can have a knock on effect on the execution of the rest of the code. This is most dangerous when deploying a swarm system, as the developer may choose to remove all of the data reporting code prior to producing the 'release' version of the system, as this code is debugging focused, and not needed in a final version once correct operation has been established. If there is a significant amount of data reporting code, removing it

all could have a non-negligible effect on the robots' behaviour, leading to potential unpredictability. In order to mitigate this problem as much as possible the robot side code has been implemented to require as little time as possible to transmit data packets.

WiFi network requirement. The system requires that all robots be connected to one WiFi network in order to operate correctly. This means the current system is limited to robots which can support WiFi, and requires that the network infrastructure be in place. Considering the system's intention as a debugging tool, and therefore its likely use within laboratories, WiFi network infrastructure is not an unreasonable requirement. However not all robot platforms support WiFi, hence this requirement limits the system's applicability to other robot platforms. Section 9.2.3 discusses extending the system in future to support a further wireless protocol such as Bluetooth.

Fixed Camera Viewpoint. The video based portion of the system has been implemented to function with the video captured from a fixed viewpoint, with an overhead, bird's-eye view of the robots. The system cannot correctly augment the video if it is captured from any other angle, even though the tag detection algorithm can cope with a range of angles. The system is therefore limited to augment the space in only two dimensions, and cannot display height-related data in any way. Many consider a three-dimensional reference frame to be a requirement of a true augmented-reality system [17], [21], and this system would require modification to support a non-specific viewpoint in order to work with modern augmented reality hardware. This is discussed in 9.2.4.

8.4 Project Exectuion

Previous sections have established that the project was successful in meeting its technical objectives. This section serves to evaluate the non-technical aspects of the project execution, including the extent to which the plan presented in chapter 4 was successfully executed, and the use of resources.

All of the project work was able to be completed in the stated time frame, between Monday 16th of January and Thursday 18th May, 2017. This was primarily due to the strength of the initial plan, as well as a proactive, agile approach to development. The majority of the tasks within the plan ran to schedule, with minor adjustments made to allow time for the implementation to be finished, and sufficient testing completed, prior to the user evaluation sessions, which took place approximately two weeks later than originally planned. Timing adjustments such as these were expected, and the initial plan included sufficient slack time to account for

them without issue. All of the internal project deadlines, including the initial report, project presentation, and project demonstration were met successfully.

The resources available within the YRL were utilised effectively throughout the project, with regular development sessions taking place within the lab itself, leading to sufficient familiarity with the hardware and infrastructure to enable a smooth, successful implementation. Development tools and resources were also utilised to a satisfactory extent, including use of the *Git* and *Github* VCS and code repository service, as discussed in section 3.4, to organise development. These tools could however have been utilised to a greater extent, including the use of branches within the version control system to organise specific feature implementations, and the use of the issue tracking features of the *Github* platform to track outstanding issues and required features. During development it was determined that, due to the relatively small size of the project and the presence of only a single developer, the use of these features was not essential. The *Qt* application framework, and associated tools and online resources, were utilised effectively to streamline development by reducing the amount of low level general application functionality that had to be implemented from first principles. The *Qt* reference documentation [48] in particular was frequently consulted, and proved instrumental in implementing the application successfully.

Overall the execution of the project was successful in delivering a system which satisfied the project aim, within the available time, making use of relevant tools and resources where necessary. Execution could have been marginally improved through a more thorough use of organisational tools, and larger projects in future would almost certainly benefit from this.

Chapter 9

Conclusions and Future Work

9.1 Conclusion

The stated aim of this project was to create a tool capable of aiding a developer in the task of debugging a robot swarm, by improving their access to, and interpretation of, previously unavailable internal robot state and sensor data. This has been achieved through the implementation of a software application and associated infrastructure system which retrieves this data wirelessly, fuses it with tracking data and a live video feed, and presents it to the user in a number of formats, including graphical formats rendered using augmented reality techniques, all in real time. The system was designed and implemented with modularity and portability in mind, using a standardised packet format and requiring no specific hardware on the robot side, to ensure it can be easily integrated with many robot platforms.

Following its implementation the software underwent testing to ensure its correct and robust operation, and was then evaluated by potential users, including a group with specific experience of swarm robotics research and development. This evaluation included a simulated debugging task in an attempt to assess the value of the system in its intended context. The results of this evaluation indicate that the system is both intuitive to use, and useful in a debugging scenario, with the majority of the participants indicating that access to, and visualisation of, internal robot data was helpful in solving the problem. One participant from the non-expert group went so far as to question how debugging robot swarms was possible without a system of this nature. The evaluation was however limited by its small sample size and simulated conditions, and in future the system could benefit from use in a range of genuine debugging situations.

One of the key motivations for the project was the desire to create a system which is practical, usable, and offers real benefit to swarm robotics developers and researchers. A number of members of the York Robotics Laboratory have commented on the potential usefulness of the system, and expressed a desire to use it in practice. It is therefore hoped that the application will continue to be used and

developed within the YRL to aid in swarm robotics research and development efforts. Taking a wider view, it is also hoped that the system, or some possible future iteration, might be useful to other swarm robotics researchers outside of the YRL. For this reason the source code is made freely available, and can be found at `https://github.com/ajewers/SwarmDebug`.

A wide range of areas for possible future expansion of the system have been identified, and are discussed in section 9.2. This first version lays the groundwork for much of that potential development, whilst also achieving the objectives identified at the start of the project. The application software has been shown to satisfy all of the core requirements of the functional specification fully, as well as satisfying a number of the secondary requirements at least in part. Considering this, and the largely positive nature of the feedback received during evaluation, the project can be considered successful.

9.2 Future Work

This project has focused on developing the first version of this swarm robotics debugging system. Responses to the initial user survey, and anecdotal comments from members of the YRL, have indicated that there is an interest in seeing the development of this system continue and its capabilities expanded to a number of new areas. This section discusses potential directions that future development could take, and considers how these might benefit users of the system. Some key areas for potential development are as follows:

- Implementation of additional debugging features.
- Implementation of analytical and experiment-enabling features, such as macro level analysis and data export, leading to the system becoming a full swarm robotics research and development platform.
- Expansion of the target robot platforms and supported hardware.
- Integration with native augmented reality hardware.

9.2.1 Additional Debugging Features

The simplest way to extend the system further would be to implement support for a number of additional debugging features. This could include support for data from different types of sensor, as well as different methods for interpreting or displaying data. Support could be added for receiving and visualising higher level data types such as directional vectors, which could be useful in a number of tasks. An example use for this would be to display a robot's '*diffusion vector*' whilst running a

dispersion behaviour. This vector indicates the direction in which the robot believes it needs to move in order to increase its distance from nearby objects. Rendering direction vectors within the visualiser, positioned to match the relevant robot's location and orientation would be a good use of the visualiser, and would remove a level of indirection for the viewer when trying to relate this vector data back to the robots current orientation, which may be changing rapidly. This is one example of a further data type that could be supported by the application, but many more exist. Adding support for additional data types would however require new packet types to be defined, and new elements to be added to the data model, as well as new visualisation code. For a single addition this is not an overwhelming task, however as the system continues to grow this could become cumbersome.

It is almost impossible to predict all of the possible types of data, or possible ways of visualising this data, that might be desired by users of the system. Each swarm behaviour will have its own unique features and needs, and each different robot platform will have a different set of sensors and actuators. Therefore creating specific visualisations that cover all these possibilities is simply infeasible. Instead an approach that might yield much more flexible and useful results could be to incorporate a scripting language, and develop an API for the visualiser, allowing users to write simple scripts which define custom visualisations. The user could then save these scripts and import them into the application, telling the visualiser to run the scripts combined with data from the data model to generate the custom visualisations. The custom data portion of the data model could be extended to support marginally more complex data structures such as arrays, allowing these custom visualisations to be more flexible.

These scripts could be written in a language such as *Python*, or *LUA*, and called from within the application using a framework such as '*Boost.Python*'. One downside to this approach would be the requirement for users to have some knowledge of programming, however considering the target users of the system are robotics developers and researchers, it seems unlikely that they would not be familiar with programming concepts. A well designed API could help to keep the requirements for these visualisation scripts simple, reducing the burden on the user. The incorporation of a scripting system of this nature would massively reduce the complexity of extending the application, and would have the added benefit of allowing scripts to be shared, potentially leading to collaborative efforts to extend the system.

9.2.2 Towards a Full Swarm Robotics Research and Development Platform

The system could be further developed to become a full platform for swarm robotics research and development, which would be used not only for debugging swarm behaviours but also for running and monitoring swarm experiments and collecting

experimental data. A number of additional features would be key to making this step forward. Firstly an organised method for managing experiment 'runs', coupled with a more robust data logging system, could be implemented. This would allow the user to specify when an experiment run was taking place, and automatically export organised experimental data related to each experiment run. This data could then be used when analysing the results of the experiment. The user could also control which elements of the data are recorded, to ensure relevance to their specific experiment.

The system could also be extended to support bi-directional communication, in order to send simple control commands to the robots from the application. This could include commands to start, stop, pause and restart specific behaviours or experiments. When coupled with the improved data logging and experiment organisation this could allow a user to start and stop experiment runs without having to interact directly with each robot, allowing them all to be started simultaneously. This bi-directional communication could also send other types of commands to the robots, potentially experiment-specific, user-defined ones, that might give high level directives to control swarm behaviour.

The addition of higher level analytical features, focusing on the behaviour of the overall swarm rather than the individual robots, could also contribute to the system's use as a swarm robotics platform. These could analyse useful swarm-level parameters such as aggregate position over time, distributions such as spread and skew, and grouped state information (how many robots in each state, average time spent in each state, etc.). Robots exhibiting behaviour significantly different from the average could also be highlighted, as a way to detect problems. These higher level analytical features could help a user to test and verify novel swarm behaviours more easily. This extra analysis could also could be coupled with new visualisations, giving the user immediate visual indicators of high level parameters, and their change over time.

The augmented video feed is one of the key features of the application, and provides scope for many possible extensions. Code could be implemented to allow video to be recorded directly from this feed, allowing a user to watch back experiment runs and replay specific sections, perhaps gaining insight into specific interactions or trends within the swarm's behaviour. Exported video could also be used for demonstration purposes, with the graphical augmentations adding context for a viewer who is less knowledgeable regarding the swarm's inner workings.

9.2.3 Expansion of Supported Hardware

Wherever possible this system has been implemented with portability in mind, with the aim of allowing different robots to be easily incorporated into the system. The

use of ArUco tags to achieve robot tracking means this portion of the system is fully independent of the robots, requiring no specific hardware, only simple printed paper tags as a minimum. The robot side code has also been designed to be as portable as possible, however it still has two main requirements which are fairly specific; the robots must be running Linux, and they must be able to connect to a WiFi network. The robot side code is extremely simple, and the API small, so porting this code to other operating systems would be a relatively simple task. However WiFi connectivity is not always common amongst smaller robot platforms, with many instead utilising Bluetooth to achieve wireless communications. Therefore implementing Bluetooth communication support within the application, alongside WiFi, and adding Bluetooth support to the robot side API could significantly increase the number of robotic platforms on which the system could be used.

Currently the system supports only one specific machine vision camera. A major challenge in the future development of this system will be to find a way to support multiple different camera set ups, with potentially bespoke drivers and APIs, in a way which requires minimal or ideally no changes to the code. One way to achieve this might be to extend the application to be able to receive the video feed via the network, leaving it up to the user to implement a system which obtains the video from their specific camera set up, and sends it to the application. This would also allow the application to be run on a machine without a physical connection to the camera hardware. One of the main barriers to this approach, and the reason it was not included in this implementation, is the high bandwidth required to transmit the video feed. As previously discussed, the robot tracking information is included in the standard data format, and can therefore be sent to the application via the network. A user with a different tracking system could therefore implement code to transmit their tracking data to the application in this manner, allowing any tracking system to be used, provided that the data is correctly formatted.

9.2.4 Integration with Dedicated Augmented Reality Hardware

In recent years there has been rapid development in the augmented reality space, with the technology beginning to find its way into real world applications. Dedicated hardware platforms such as Microsoft HoloLens, Google Glass, and Google Cardboard have given consumers a first taste of real, perspective based, head mounted, augmented reality. By contrast, the version of augmented reality used in this project is simplistic, and fairly limited. In the future, a combination of a robotics debugging tool such as the system developed in this project, and dedicated AR hardware, could lead to much more immersive and effective human-robot and human-swarm interaction. The data visualisation could be developed to support full 3D by utilising the motion tracking abilities of modern AR hardware, and the orientation calculation

features built into the ArUco system. This would allow the system to render augmentations and data visualisations from any perspective. This report firmly believes that, considering the immense potential of AR systems when coupled with robotics, the use of AR-based tools when working with robots will become commonplace in the future.

9.2.5 Integration with Tablet Application

Another project being completed simultaneously with this project, also within the YRL, has implemented a similar swarm debugging system as an Android app for a tablet computer. The main aim of this approach is to allow the user to be more mobile whilst using the system, so that they can interact with the swarm in a hands on fashion whilst also having immediate access to internal data from the robots. In the future the tablet application could be integrated with the application developed in this project, to form a single platform, with the tablet acting as a satellite terminal through which the user can access the system.

9.3 Summary

The wide variety of possible future extensions discussed here are a testament to the system's potential. Following the conclusion that this project was successful in satisfying its aim and objectives, it is hoped that the system will serve as a foundation on which future work can build. The field of swarm robotics continues to develop quickly, but much further work is needed to better enable the design and implementation of swarm systems, and robust, constructive human-swarm interaction. It is therefore hoped that this project, and the system developed, contributes to the field by better enabling researchers within the YRL and beyond to develop and debug their systems.

Appendix A

Ethics Documents

A.1 Ethics Form

The signed hard-copy of this ethics form is available on request.

FAST-TRACK ETHICAL APPROVAL FORM (STUDENTS)

This fast-track system is for taught students only. Research students and staff must complete the full Ethical Approval Form.

If you answer **YES** to any of the following you must complete either this Fast-track ethical approval form, to be signed off by your supervisor, or a full Ethical approval application, to be approved by the Physical Science Ethics Committee (allow at least two weeks for this process).

Note that the outcome of the Fast-track system may result in you needing to complete a full ethical approval application.

Does your project involve any of the following?

	YES	NO
Human participants (adults or children)	x	
Human material (e.g. tissue or fluid samples)		X
Human data (e.g. surveys and questionnaires on issues such as lifestyle, housing and working environments, attitudes and preferences)		X
Vertebrates, especially mammals and birds		X
Any other organisms not previously mentioned		X
Military or defence context		X
Funding sources with potential to adversely affect existing relationships or bring the University or Department into disrepute.		X
Restrictions on dissemination		X
Overseas countries under regimes with poor human rights record or identified as dangerous by the Foreign & Commonwealth Office		X
Applications that could potentially involve unethical practice, including potential dual-use applications which could be unethical		X

Students: you should discuss the ethical considerations of your project with your project supervisor and, if necessary, fill in a full ethics form to be submitted to the Physical Sciences Ethics Committee.

Supervisors: Please ensure you are familiar with the University's 'Code of practice and principles for good ethical governance' in order to guide your student effectively. Please seek guidance from the Departmental Ethics Officer if you are uncertain about any ethical issue arising from this application.

FAST-TRACK ETHICAL APPROVAL FORM (STUDENTS)

Project Information:

Student Name: Alistair Jewers

Course Title: Electronic Engineering w/ Music Technology Systems

***Tick one box**:*

Undergraduate project	x	Postgraduate project	□
Undergraduate module assignment	□	Postgraduate module assignment	□
Other (Please state……)	□		

Title of project: Augmented Reality Debugging System for Robotic Swarms

Project supervisor / module leader name: Alan Millard

Protocol:

a): *If you answer **NO** to any of the following you must submit a full ethical approval form*

	If you answer yes to any of the following, this must be explicit in any supporting literature (e.g. consent forms, information sheets and questionnaires)	YES	NO	N/A
1	Will you describe the procedures to participants in advance, so that they are informed about what to expect?	X		
2	Will you tell participants that their participation is voluntary?	X		
3	Will you inform the participants of the purpose / background of the study?	X		
4	Will you obtain written consent for participation?	X		
5	If the research is observational, will you ask participants for their consent to being observed?	X		
6	Will you tell participants that they may withdraw from the research at any time and for any reason?	X		
7	With questionnaires and interviews will you give participants the option of omitting questions they do not want to answer?	X		
8	Will you tell participants that their data will be treated with full confidentiality and that, if published, it will not be identifiable as theirs?	X		

Protocol:

b): *If you answer **YES** to any of the following you must submit a full ethical approval form.*

		YES	NO	N/A
9	Is your study designed to be challenging/disturbing (physically or psychologically)?		X	
10	Will you deliberately mislead your participants?		X	
11	Does your study involve taking bodily samples?		X	
12	Is your study physically invasive?		X	
13	Is there any obvious or inevitable adaptation of your research findings to ethically questionable aims?		X	
14	Could the methodologies or findings of your study damage the reputation of the University of York?		X	

Health and Safety:
Please identify any risks to the participants and state any precautions you will take to ensure their health and safety:

Participants: *If you answer **YES** to any of the following you must submit a full ethical approval form. If you have ticked **YES** to 15 and your participants are **patients**, in addition to the full ethical application you must follow the Guidelines for Ethical Approval of NHS Projects.*

			YES	NO	N/A
15	Does your project involve work with animals			X	
16	Will any of the participants be from one of the following vulnerable groups? **Note that you may also need to obtain satisfactory DBS clearance (or equivalent for overseas students)**	Children under 18		X	
		People with learning difficulties			
		People who are unconscious or severely ill			
		NHS patients			
		Other vulnerable groups (specify)			

Data Protection: *If you answer **NO** to any of the following you must submit a full ethical approval form*

		YES	NO	N/A
17	Any personal / sensitive data will be stored in password protected folders on computers.	X		
18	Any hard copies of personal data (including consent forms) will be stored in a secure place.	X		
19	Only the student and supervisors will have access to the data generated from the study. (The supervisor may share the anonymised data with other researchers at the University of York)	X		
20	The data will be preserved beyond the study in line with University policy and will be placed in the custody of the supervisor at the end of the project.	X		
21	All data will be anonymised prior to analysis. Please state your method of anonymisation: The questionnaires are filled out anonymously, names are not taken at any point.	X		

FOR THE STUDENT TO COMPLETE:

Please complete and sign the following section and submit to your supervisor alongside any supporting documentation (this includes consent forms, information sheets and questionnaires where necessary).

Provide a brief summary of the participants and procedures of your project (max 100 words)

1. Short voluntary online questionnaire regarding the participant's opinions on the proposed system and its feature set. Participants may optionally include their email address if they wish to be contacted regarding the later evaluation sessions, but this information is not used as part of the study and is purely for practical purposes. Stored in a password protected manner.
2. Observed voluntary user testing sessions, where participants use the finished system to interact with a robot swarm. Data is collected through observation, and a short questionnaire following the testing session. No personal data collected, and all collected data remains anonymous.

I have considered the ethical implications of this project and have identified no significant ethical implications requiring a full ethics submission to the Physical Sciences Ethics Committee [X]

I have included all relevant paperwork (e.g. consent form, information sheet, questionnaire/interview schedules) with this application [X]

Signed…… Print name: Alistair Jewers Date: 03/05/2017
(Student)

FOR THE SUPERVISOR TO COMPLETE:
By signing this form you are taking responsibility for the ethical conduct of this project

The student has taken all reasonable steps to ensure ethical practice in this study and I can identify no significant ethical implications requiring a full ethics submission to the Physical Sciences Ethics Committee [ ]

I have checked and approved all relevant paperwork required for this proposal [ ]

STATEMENT OF ETHICAL APPROVAL

This project has been considered using the Physical Sciences Ethics Committee Fast-track ethical approval procedure, agreed by the Physical Sciences Ethics Committee of the University of York, and is now approved.

Signed…….. Print name…….. Date……
(Supervisor/Module leader)

OR

The details on this form indicate a need for a full application to PSEC. The practical aspects of this project will not proceed until this has application has been approved.

Signed…….. Print name…….. Date……
(Supervisor/Module leader)

A.2 User Evaluation Session Consent Form

Augmented Reality Debugging System for Swarm Robotics –
User Evaluation Sessions

Participant ID: _____

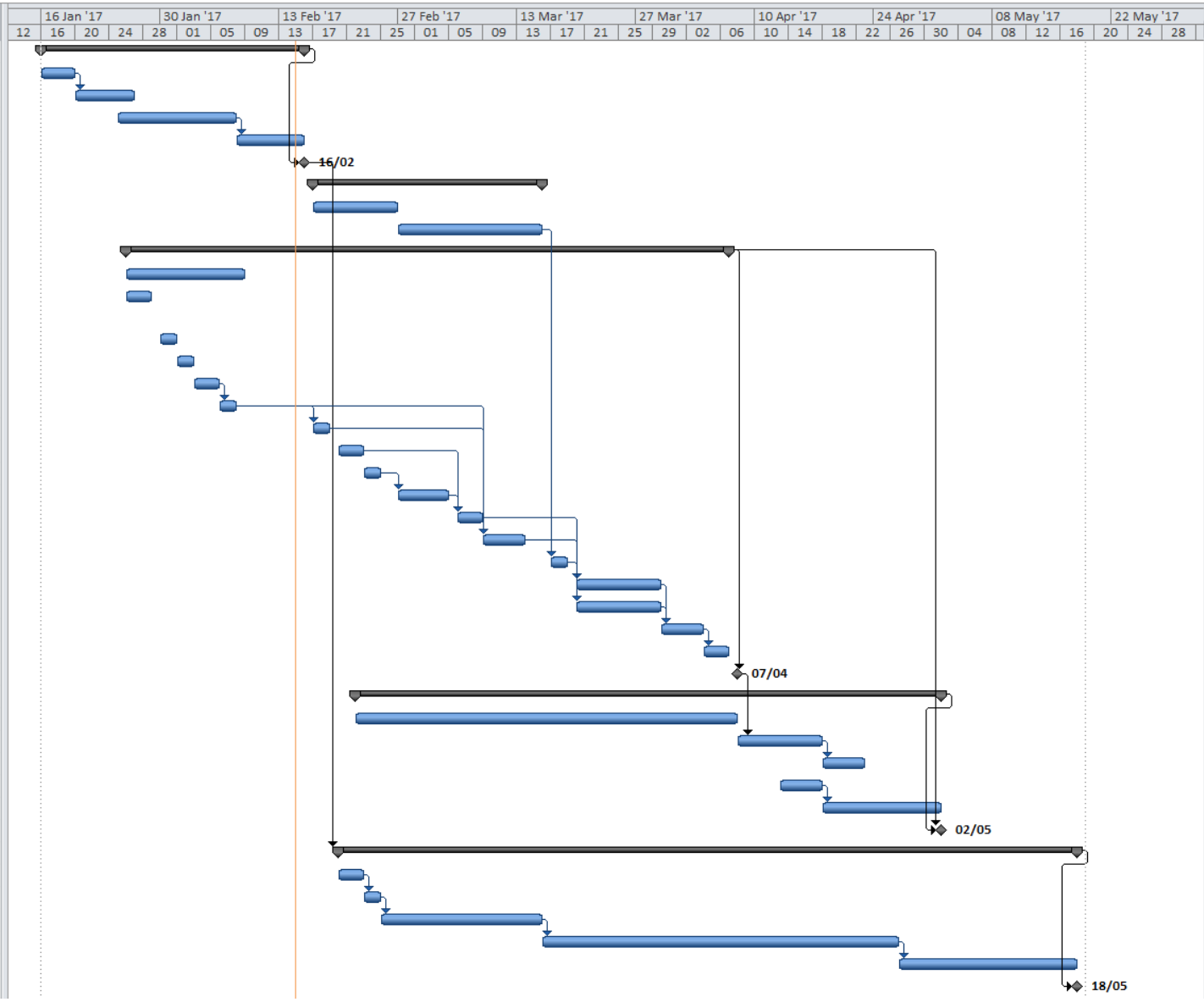
I confirm that I am taking part in this study voluntarily, and agree to the observation of my session, and the use of my data in an anonymised fashion, for the purposes of system testing and evaluation.

Signed: _____

Appendix B

Gantt Chart

Task Name	Duration	Start	Finish	Predec
Initial Report	**31 days**	**Mon 16/01/17**	**Wed 15/02/17**	
Initial Reading	4 days	Mon 16/01/17	Thu 19/01/17	
Literature Survey	7 days	Fri 20/01/17	Thu 26/01/17	2
Initial Report Draft	14 days	Wed 25/01/17	Tue 07/02/17	
Intial Report Editing	8 days	Wed 08/02/17	Wed 15/02/17	4
Intial Report Deadline	0 days	Thu 16/02/17	Thu 16/02/17	1
User Survey	**27 days**	**Fri 17/02/17**	**Wed 15/03/17**	
Create Survey for Swarm Robotics Researchers	10 days	Fri 17/02/17	Sun 26/02/17	
Distribute Survey and Collate Results	17 days	Mon 27/02/17	Wed 15/03/17	
Development	**71 days**	**Thu 26/01/17**	**Thu 06/04/17**	
Read and Understand Relevant Existing Code	14 days	Thu 26/01/17	Wed 08/02/17	
Establish Development Environment and Toolchain	3 days	Thu 26/01/17	Sat 28/01/17	
Learn to Re-Program E-Puck Robots	2 days	Mon 30/01/17	Tue 31/01/17	
Design Software Architecture	2 days	Wed 01/02/17	Thu 02/02/17	
Design General User Interface	3 days	Fri 03/02/17	Sun 05/02/17	
Incorporate Tracking Camera Code	2 days	Mon 06/02/17	Tue 07/02/17	15
Implement Tracking Camera Controller	2 days	Fri 17/02/17	Sat 18/02/17	16
Implement Wireless Data Receive	3 days	Mon 20/02/17	Wed 22/02/17	
Determine Robot Data Types	2 days	Thu 23/02/17	Fri 24/02/17	
Design and Implement Data Model	6 days	Mon 27/02/17	Sat 04/03/17	19
Implement Mapping Received Data to Model	3 days	Mon 06/03/17	Wed 08/03/17	20,18
Implement Basic Visualiser	5 days	Thu 09/03/17	Mon 13/03/17	16,17
Design UI Data Representation	2 days	Fri 17/03/17	Sat 18/03/17	9
Implement Graphical Data Representation	10 days	Mon 20/03/17	Wed 29/03/17	21,23,22
Implement Textual Data Representation	10 days	Mon 20/03/17	Wed 29/03/17	21,23
Implement Data Representation Filtering	5 days	Thu 30/03/17	Mon 03/04/17	24,25
Implement Robot Data Comparison	3 days	Tue 04/04/17	Thu 06/04/17	26
Development Complete	0 days	Fri 07/04/17	Fri 07/04/17	10
Testing	**69 days**	**Wed 22/02/17**	**Mon 01/05/17**	
Continuous Integration Testing	45 days	Wed 22/02/17	Fri 07/04/17	
Manual Verification Testing	10 days	Sat 08/04/17	Mon 17/04/17	28
Verification Fixes and Changes	5 days	Tue 18/04/17	Sat 22/04/17	31
User Testing	5 days	Thu 13/04/17	Mon 17/04/17	
Final Fixes and Changes	14 days	Tue 18/04/17	Mon 01/05/17	33
Demonstration	0 days	Tue 02/05/17	Tue 02/05/17	29,10
Final Report	**87 days**	**Mon 20/02/17**	**Wed 17/05/17**	**6**
Create a Report Plan	3 days	Mon 20/02/17	Wed 22/02/17	
Create a Skeleton Document	2 days	Thu 23/02/17	Fri 24/02/17	37
Literature Review	19 days	Sat 25/02/17	Wed 15/03/17	38
Draft Report	42 days	Thu 16/03/17	Wed 26/04/17	39
Editing and Proof Reading	21 days	Thu 27/04/17	Wed 17/05/17	40
Final Report Deadline	0 days	Thu 18/05/17	Thu 18/05/17	36



Appendix C

Source Code Files

TABLE C.1: Source code and tertiary files that make up the main application.

File	Purpose
main.cpp	The entry point for the application. Instantiates the MainWindow class.
mainwindow.cpp, .h	The core class, controls the set up and tear down processes, handles responding to UI events within the main window and routing data throughout the system.
mainwindow.ui	Describes the user interface layout in an XML-like format. Used by the Qt framework to construct the UI.
datamodel.cpp, .h	The top level class encapsulating the full data model. Maintains a list of *RobotData* objects.
robotdata.cpp, .h	A class encapsulating the data of a single robot, including ID, name, position, path, state, sensor data and all custom user data.
datathread.cpp, .h	This class contains all routines for receiving data from the robots via wifi, and is runs on a thread of its own.
cameracontroller.cpp, .h	The high level class encapsulating the routines and data related to reading the tracking camera.
machinevision.cpp, .h	A lower level class encapsulating routines for interfacing with the camera hardware.
visualiser.cpp, .h	This class encapsulates the visualiser GUI component, and is implemented to conform the Qt framework by extending the QWidget class. Contains routines for generating and applying the video augmentations based on the current robot data and visualiser settings.
viselement.h	Contains an abstract class definition for a single visualiser settings element. These elements are used to define how specific elements of the video augmentation are rendered, and also contain settings and variables relevant to this task.

vis*.cpp, .h	Classes beginning with the '*vis*' prefix derive from the *VisElement* abstract class and contain routines for rendering the visualisation for one type of data. The latter part of the class name identifies the relevant data type.
irdataview.cpp, .h	A custom GUI object, derived from QWidget, which displays the raw IR sensor data as a bar graph in the data window.
settings.cpp, .h	Encapsulates the general application settings and provides functions for changing their values. Implemented as a singleton class to allow access from anywhere in the application.
log.cpp, .h	Encapsulates the functionality for recording events and data in log files. Implemented as a singleton class to allow access from anywhere in the application.
util.cpp, .h	Contains static utility functions used in various places throughout the application code.
*settingsdialog.cpp, .h	Classes ending with the '*settingsdialog*' suffix describe dialog windows for adjusting the settings related to the visualisation of specific data types, identified by the first part of the class name.
robotinfodialog.cpp, .h	Defines a dialog window which displays meta information about a specific robot, and provides controls to change the robot's display colour and delete its data from the data model.
addidmappingdialog.cpp, .h	Defines a dialog window which can be used to add a non-standard ID mapping.
testingwindow.cpp, .h	Defines a dialog window for running and displaying the results of the data model unit tests.
appconfig.h	Contains pre-processor definitions for controlling the inclusion of specific code segments.
SwarmDebug.pro	Used by the Qt framework to build the application. Lists the necessary code files and libraries.

TABLE C.2: Source code files that make up the robot side API.

File	Purpose
debug_network.cpp	This file encapsulates networking functionality for communicating with the debugging system. Contains routines for sending data of specific types, as well as for sending raw packets.
debug_network.h	Header file for the debugging system network interface. Also contains definitions for data type identifiers.

Appendix D

Robot Side API

TABLE D.1: The contents of the *DebugNetwork* class that forms the robot side API.

Variables		
Name	Type	Purpose
socket_ready	Boolean	Indicates whether the socket has been created successfully, and it ready to be used.
sock_in	Struct	A structure defining the target socket.
sock_fd	Integer	Identifies the socket on the local machine (robot).
robot_id	Integer	The numerical ID of this robot. Set at initialisation time. Inserted to the header of each packet to identify the sender.
Functions		
Name	Return Type	Purpose
init	Void	Initialises the class by setting up the socket and setting the ID for this robot. Requires a port number, a target host IP address and the robot ID as arguments.
destroy	Void	Shuts down the socket.
sendData	Void	Sends a given string as a raw data packet.
sendWatchdogPacket	Void	Constructs and sends a watchdog packet. Requires the robot name to be provided as an argument.
sendStatePacket	Void	Constructs and sends a state packet. Requires the current state to be provided as an argument string.

sendIRDataPacket	Void	Constructs and sends an IR data packet. Requires the data be arranged into an array, and takes a pointer to the first element and a size value as parameters. Also takes an extra boolean parameter which can be set to true to indicate that this packet should be identified as backgroud IR data.
sendLogMessage	Void	Constructs and sends a message packet to display a message in the application console and logs. The message string is supplied as an argument.
sendCustomData	Void	Constructs and sends a custom data packet. The key value pair are supplied to this function as two argument strings.
getRobotID	Integer	Returns the numerical robot ID, as currently set.
setRobotID	Void	Sets the numerical robot ID based on an integer argument.

Appendix E

User Guide

E.1 Overview

This user guide is for the *SwarmDebug* swarm robotics debugging system. *SwarmDebug* is a tool for monitoring robot swarms and extracting data in real time, to aid with the development and debugging of swarm robotics behaviours. The system functions by allowing the individual robots within a swarm to report data over WiFi to a central application. The system also tracks the robots within a live video feed, and fuses the two data sources. This user guide describes how to incorporate the *SwarmDebug* system into existing robot behavioural code, and how to use the main application.

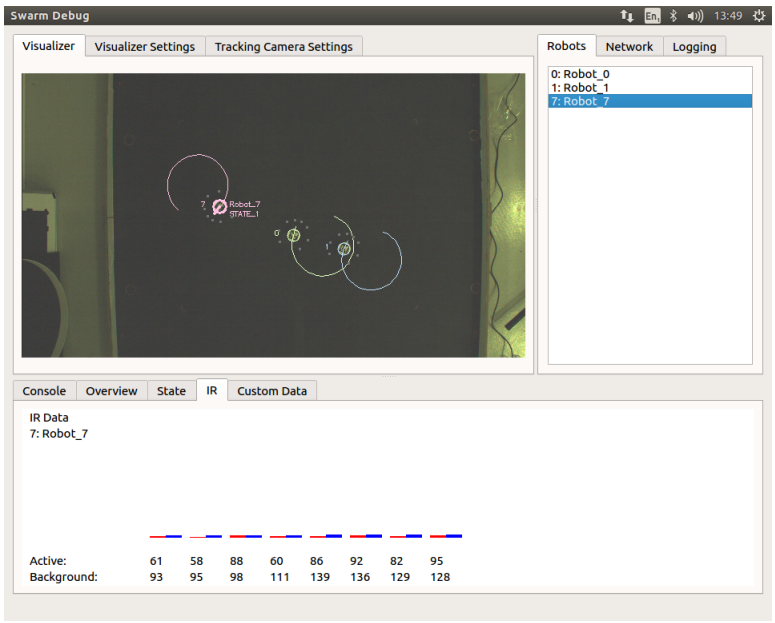


FIGURE E.1: A screenshot of the *SwarmDebug* application, examining a robot's IR sensor data.

E.2 Incorporating the *SwarmDebug* API

The *SwarmDebug* system gives a user full control over the data that each robot reports, and how often. On the robot side, an API is provided which includes functions for transmitting packets of data to the main application. This API is lightweight and simple, consisting of only 2 files, and 10 functions.

To incorporate the API into existing robot behavioural code, complete the following steps:

1. Begin by cloning or downloading the *SwarmDebug* repository, from `http://github.com/ajewers/SwarmDebug/`.
2. Add the files `RobotAPI/debug_network.cpp` and `RobotAPI/debug_network.h` to your build configuration.
3. Within your robot controller code, during initialisation, add a call to the `init(port, server_ip, robot_id)` function.
    - The *port* parameter should be the port number on which the main application will be listening for data. The application defaults to port 8888.
    - The *server_ip* parameter should be the IP address of the server or computer running the main application, in a string format. For example: "192.168.1.101".
    - The *robot_id* parameter should be the ID number of this robot. This should ideally match the ID number of the tracking tag attached to the robot, but this is not strictly necessary.
4. Throughout the robots behavioural code, whenever you wish the robot to report data to the debugging system, add a call to one of the following functions:
    - Call `sendWatchdogPacket(name)` to send a watch-dog packet, notifying the debugging system that this robot is still active. The *name* parameter should be a string containing the name you wish the robot to display.
    - Call `sendStatePacket(state)` to send a state packet, notifying the debugging system of the robot's current state. The *state* parameter should be a string containing the state the robot is currently in.
    - Call `sendIRDataPacket(data, count, background)` to send the robot's current IR sensor readings to the debugging system. The sensor data should be arranged into an array, where the *data* parameter is a pointer to the first element, the *count* parameter contains the number of values in the array, and the *background* parameter is a boolean, where

true indicates that this data is a passive background reading, and *false* indicates it is an active reading.

- Call `sendCustomData(key, value)` to send a custom data packet to the debugging system. This data should be formatted as a key/value pair of strings, supplied as the two parameters of the function respectively.
- Call `sendLogMessage(message)` to send a single message to the debugging system, to be displayed in the application console and recorded in the log, if running.

5. At the exit point of the behavioural code, add a call to the `destroy()` function to tear down the API, and stop the networking code.

The following is an example of how the system might be incorporated into some hypothetical robot behavioural code:

```
...
import debug_network.h

void behaviourInit(void) {
    ... Initialisation code ...

    DebugNetwork::init(8888, "192.168.1.101", this->id);
}

void behaviourControlStep(void) {
        ... Behavioural step code ...

        DebugNetwork::sendWatchdogPacket(this->name);
        DebugNetwork::sendStatePacket(this->state);

        ... Read IR sensors ...
        DebugNetwork::sendIRDataPacket(this->ir_data, IR_SENSOR_COUNT, false);
}

...

void behaviourEnd(void) {
        ... Tear-down code ...

        DebugNetwork::destroy();
}
```

E.3 Using the *SwarmDebug* Application

The *SwarmDebug* application requires a Linux operating system. Start the application by running the executable. To begin listening for data packets from the robots, navigate to the *network* tab within the right hand panel, enter the desired port (default 8888, this should match the port entered when initialising the API in the robot code), and press the *Start Listening* button. Navigate back to the *Robots* tab, and select a robot from the list to begin examining its data. The lower panel is used to display detailed information about the selected robot, with each of the tabs including information about a different data type.

The main panel in the top left of the application displays the visualiser. This should be showing a video feed of the robots, with an overlay highlighting the robot's positions and identifying them by ID, and by name if the robots have reported a name using a watch-dog packet. This visualiser component provides graphical displays for the various data regarding the robots. To configure this display, navigate to the *Visualiser Settings* tab within the same panel. The different graphical displays are listed here, and can be enabled and disabled by clicking their corresponding check boxes. More detailed settings for each display can be accessed by double clicking their list entries to open a detailed settings window.

If the IDs assigned to the robots do not match the IDs being reported by the tracking system, different mappings can be set in the *Tracking Camera Settings* tab, found in the main panel. Data from the system can be exported to a text file by enabling logging in the *Logging* tab, found in the right hand panel. Double clicking on a robot within the robot list will open a robot information window, where the robot's display colour can be set, and its data can be erased from the system if necessary.

E.4 Compiling the Application from Source

The *SwarmDebug* application is developed using the *Qt* application framework. In order to compile the application from the source the Qt libraries must be present. The easiest way to do this is to complete the following steps:

1. Install Qt, and the *Qt Creator* IDE from the following link: `https://info.qt.io/download-qt-for-application-development`
2. Clone the *SwarmDebug* source repository from `https://github.com/ajewers/SwarmDebug.git`.
3. Open the Qt Creator IDE, and import the SwarmDebug project.
4. Press build and run!

This process must be completed on a computer running the Linux operating system. The compilation relies on the presence of the *Common Vision Blocks* libraries on the compiling computer. If these libraries are not present, a version of the code can be compiled with the camera feed functionality disabled. In order to do this complete the following steps:

1. In the *appconfig.h* file, disable the `CVB_CAMERA_PRESENT` definition by converting line 4 to a comment using two forward slashes.
2. In the *SwarmDebug.pro* file, disable the inclusion of the `lCVCImg`, `lCVCDriver` and `lCVCUtilities` libraries by converting lines 94, 95 and 96 to comments by adding a hash character to the beginning of each.
3. Press build and run!

Appendix F

Test Results

F.1 Manual UI Testing Results

Visualiser Panel Tab System	
Purpose	Allows access to the different tabs within the visualiser panel.
Required Functionality	Must display the names of each different tab. Must allow the user to click on the tabs and thus change between them. The required tabs are the visualiser tab, the visualiser settings tab and the tracking camera settings tab.
1. Examine UI element visually. Verify that it appears correct. Verify that it contains all elements necessary to satisfy its purpose.	
The tab bar appears to be visually correct. All required tabs are present and visible.	
2. Examine all text within the element. Check for errors in both meaning and spelling.	
The spelling of each tab name is correct. The meaning of each tab name is relatively clear, although 'Settings' is ambiguous, and only related to the visualiser through position and context. Consider renaming this tab to clarify its purpose.	
3. Verify that all components within the element which perform actions in response to user input operate correctly.	
Tab selection operates correctly. Clicking any of the tabs changes the panel to display the contents of that tab. This satisfies the required functionality.	
4. Verify that all components respond quickly to user input.	
The tab changes immediately when clicked.	
5. Verify that component actions and functionality do not degrade with extreme use (sustained rapid input, large numbers of input changes, etc).	
Repeatedly and rapidly changing tabs does not have any detrimental affect on the application.	
6. Verify that all data displayed within the element is visible, readable, correctly arranged and correctly labelled.	
n/a.	

7. Verify that the element behaves sensibly when window resizing occurs, and that it remains usable and data remains visible whenever possible. The tabs are not affected by window resizing. All of the tabs fit within the minimum size of the panel. If the panel is reduced below this size it is minimized, and the tabs are hidden as intended.
8. Verify that the element updates promptly when responding to changes in data. n/a.

Fixes Required	Consider changing the text on the visualiser settings tab from 'Settings' to something more informative to improve clarity.

Visualiser Settings Tab	
Purpose	Displays the current settings for the visualiser and allows the user to change them.
Required Functionality	Must correctly display the current visualiser settings. Must allow the user to adjust the general visualiser settings, and access dialog windows for changing the settings of specific visualisations.
1. Examine UI element visually. Verify that it appears correct. Verify that it contains all elements necessary to satisfy its purpose. Panel layout appears to be correct. Controls are included to modify all of the required settings.	
2. Examine all text within the element. Check for errors in both meaning and spelling. All spelling is correct, and the meaning of all text is clear.	
3. Verify that all components within the element which perform actions in response to user input operate correctly. All the settings controls work correctly. Double clicking on any of the visualiser config elements in the list displays the appropriate settings dialog, if one exists.	
4. Verify that all components respond quickly to user input. All controls respond immediately to input.	
5. Verify that component actions and functionality do not degrade with extreme use (sustained rapid input, large numbers of input changes, etc). Rapidly enabling and disabling settings multiple times has no detrimental affect. Disabling and re-enabling the robot colours setting causes the robots to be re-assigned colours randomly, which might be undesired behaviour.	
6. Verify that all data displayed within the element is visible, readable, correctly arranged and correctly labelled. Information regarding the current settings is readable, correctly arranged and clearly labelled. Detailed settings for each visualiser element are described in text next to the element name.	
7. Verify that the element behaves sensibly when window resizing occurs, and that it remains usable and data remains visible whenever possible. All text fits within the minimum panel width, and scroll bars are presented when the height becomes too small to display the full visualiser config element list. Resizing is handled gracefully.	
8. Verify that the element updates promptly when responding to changes in data. Changes to the general settings and the visualiser settings are reflected in the panel immediately.	
Fixes Required	Investigate alternative methods for assigning robot colours.

Camera Settings Tab	
Purpose	Displays the current settings for the tracking camera and allows the user to change them.
Required Functionality	Must correctly display the current camera settings. Must allow the user to adjust the camera settings. Must allow the user to adjust the tracking system settings, and apply/remove mappings between specific tracking tag IDs and robot IDs.
1. Examine UI element visually. Verify that it appears correct. Verify that it contains all elements necessary to satisfy its purpose. Panel layout appears to be correct. Controls are included to modify all of the required settings. A table and associated controls are included to allow the ID mapping to be modified.	
2. Examine all text within the element. Check for errors in both meaning and spelling. All spelling is correct, and the meaning of all text is clear.	
3. Verify that all components within the element which perform actions in response to user input operate correctly. All the settings controls and the mapping table work correctly.	
4. Verify that all components respond quickly to user input. All controls respond immediately to input.	
5. Verify that component actions and functionality do not degrade with extreme use (sustained rapid input, large numbers of input changes, etc). Camera resolution input boxes are input validated to only accept numbers in the range 1 to 10,000. Rapid use of controls has no detrimental affect.	
6. Verify that all data displayed within the element is visible, readable, correctly arranged and correctly labelled. Current settings are all correctly displayed.	
7. Verify that the element behaves sensibly when window resizing occurs, and that it remains usable and data remains visible whenever possible. All components fit within the panel's minimum dimensions, and are hidden when the panel is minimized.	
8. Verify that the element updates promptly when responding to changes in data. n/a.	
Fixes Required	None.

Robot List Panel Tab System	
Purpose	Allows access to the different tabs within the robot list panel.
Required Functionality	Must display the names of each different tab. Must allow the user to click on the tabs and thus change between them. The required tabs are the robot list tab, the network settings tab and the logging tab.
1. Examine UI element visually. Verify that it appears correct. Verify that it contains all elements necessary to satisfy its purpose.	
The tab bar appears to be visually correct. All required tabs are present and visible.	
2. Examine all text within the element. Check for errors in both meaning and spelling.	
The spelling of each tab name is correct. The meaning of each tab name is clear.	
3. Verify that all components within the element which perform actions in response to user input operate correctly.	
Tab selection operates correctly. Clicking any of the tabs changes the panel to display the contents of that tab. This satisfies the required functionality.	
4. Verify that all components respond quickly to user input.	
The tab changes immediately when clicked.	
5. Verify that component actions and functionality do not degrade with extreme use (sustained rapid input, large numbers of input changes, etc).	
Repeatedly and rapidly changing tabs does not have any detrimental affect on the application.	
6. Verify that all data displayed within the element is visible, readable, correctly arranged and correctly labelled.	
n/a.	
7. Verify that the element behaves sensibly when window resizing occurs, and that it remains usable and data remains visible whenever possible.	
When the panel width is reduced such that the tabs do not fit, small scroll arrows are added to allow the user to scroll along the tabs, therefore remaining accessible even when the window or panel size is reduced.	
8. Verify that the element updates promptly when responding to changes in data.	
n/a.	
Fixes Required	None.

Robot List	
Purpose	Displays a list of all the robots for which the system has data, allowing the user to select them.
Required Functionality	Must display a list of all the known robots, identifying them by both ID and name. Must allow the user to select any of the robots, causing the other parts of the interface to target the newly selected robot.
1. Examine UI element visually. Verify that it appears correct. Verify that it contains all elements necessary to satisfy its purpose. The list displays correctly. Contains all know robots. Clearly shows current selection.	
2. Examine all text within the element. Check for errors in both meaning and spelling. n/a.	
3. Verify that all components within the element which perform actions in response to user input operate correctly. Robots can be selected correctly by clicking, and changing the selection causes the rest of the application to update to the new focus.	
4. Verify that all components respond quickly to user input. The application responds to a new selection immediately.	
5. Verify that component actions and functionality do not degrade with extreme use (sustained rapid input, large numbers of input changes, etc). Rapidly changing the selected robot has no detrimental effect on the application.	
6. Verify that all data displayed within the element is visible, readable, correctly arranged and correctly labelled. For each robot the correct ID number and name are displayed, as defined in the data model. The the number of robots exceeds the available space in the list a scroll bar is added.	
7. Verify that the element behaves sensibly when window resizing occurs, and that it remains usable and data remains visible whenever possible. The list is unaffected by window resizing, adding a scroll bar when necessary at small sizes.	
8. Verify that the element updates promptly when responding to changes in data. Changes in the data model are reflected immediately, including robots being added, removed and changing name.	
Fixes Required	None.

Network Settings Tab	
Purpose	Allows the user to configure settings related to the network communication with the robots.
Required Functionality	Must display the current network settings. Must allow the user to change the network settings. Must allow the user to start and stop the data thread which listens for packets.
1. Examine UI element visually. Verify that it appears correct. Verify that it contains all elements necessary to satisfy its purpose.	
The panel appears visually correct, with the necessary components visible.	
2. Examine all text within the element. Check for errors in both meaning and spelling.	
All spelling correct and all meanings clear.	
3. Verify that all components within the element which perform actions in response to user input operate correctly.	
The port number can be changed correctly, and is input-validated to reject non-numerical input. The button for starting and stopping the data thread operates correctly.	
4. Verify that all components respond quickly to user input.	
All components respond immediately.	
5. Verify that component actions and functionality do not degrade with extreme use (sustained rapid input, large numbers of input changes, etc).	
Rapidly and repeatedly pressing the start/stop listening button appears to have no detrimental effect. The port number entry box rejects numbers below 1, but does not have a maximum value, as some computers support extremely high port numbers.	
6. Verify that all data displayed within the element is visible, readable, correctly arranged and correctly labelled.	
n/a.	
7. Verify that the element behaves sensibly when window resizing occurs, and that it remains usable and data remains visible whenever possible.	
Controls remain usable at the full range of interface sizes.	
8. Verify that the element updates promptly when responding to changes in data.	
n/a.	
Fixes Required	None.

Data Logging Tab	
Purpose	Allows the user to configure the data logging functionality.
Required Functionality	Must display the current data logging settings. Must allow the user to change the data logging settings. Must allow the user to start and stop the data logging.
1. Examine UI element visually. Verify that it appears correct. Verify that it contains all elements necessary to satisfy its purpose. The panel appears correct, with all required elements present.	
2. Examine all text within the element. Check for errors in both meaning and spelling. Spellings correct and meanings clear. The log file directory path can be extremely long, and might overrun the available width of the panel. Long paths should be truncated.	
3. Verify that all components within the element which perform actions in response to user input operate correctly. All components operate correctly. The log file path can be set using a file dialog.	
4. Verify that all components respond quickly to user input. All components respond immediately.	
5. Verify that component actions and functionality do not degrade with extreme use (sustained rapid input, large numbers of input changes, etc). Rapid use of the start/stop button has no detrimental effect. Invalid file selections are rejected by the dialog.	
6. Verify that all data displayed within the element is visible, readable, correctly arranged and correctly labelled. n/a.	
7. Verify that the element behaves sensibly when window resizing occurs, and that it remains usable and data remains visible whenever possible. All controls fit within the minimum size of the panel. See 2. for note on the log file path length.	
8. Verify that the element updates promptly when responding to changes in data. n/a.	
Fixes Required	Add code to handle long directory paths gracefully.

Data Panel Tab System	
Purpose	Allows access to the different tabs within the data panel.
Required Functionality	Must display the names of each different tab. Must allow the user to click on the tabs and thus change between them. The required tabs are: Console, Overview, State, IR data, Custom data.
1. Examine UI element visually. Verify that it appears correct. Verify that it contains all elements necessary to satisfy its purpose.	
The tab bar appears to be visually correct. All required tabs are present and visible.	
2. Examine all text within the element. Check for errors in both meaning and spelling.	
The spelling of each tab name is correct. The meaning of each tab name is clear.	
3. Verify that all components within the element which perform actions in response to user input operate correctly.	
Tab selection operates correctly. Clicking any of the tabs changes the panel to display the contents of that tab. This satisfies the required functionality.	
4. Verify that all components respond quickly to user input.	
The tab changes immediately when clicked.	
5. Verify that component actions and functionality do not degrade with extreme use (sustained rapid input, large numbers of input changes, etc).	
Repeatedly and rapidly changing tabs does not have any detrimental affect on the application.	
6. Verify that all data displayed within the element is visible, readable, correctly arranged and correctly labelled.	
n/a.	
7. Verify that the element behaves sensibly when window resizing occurs, and that it remains usable and data remains visible whenever possible.	
All tabs fit within the minimum window width.	
8. Verify that the element updates promptly when responding to changes in data.	
n/a.	
Fixes Required	None.

Console Tab	
Purpose	Displays a text based console which reports messages regarding application events and messages from the robots.
Required Functionality	Must contain a text console. Must display messages regarding the application and messages from the robots. Must display messages in order. Must identify the source of all robot messages. Must update immediately when a message is received from either source.
1. Examine UI element visually. Verify that it appears correct. Verify that it contains all elements necessary to satisfy its purpose. The console is visually correct, however the most recent message is displayed on the top line, counter-intuitively.	
2. Examine all text within the element. Check for errors in both meaning and spelling. The text within the element comes from the messages, hence cannot be checked here.	
3. Verify that all components within the element which perform actions in response to user input operate correctly. Messages are presented correctly and in order. Robot messages are identified correctly.	
4. Verify that all components respond quickly to user input. n/a.	
5. Verify that component actions and functionality do not degrade with extreme use (sustained rapid input, large numbers of input changes, etc). Long messages and large numbers of messages are handled gracefully, and remain readable using the automatic scroll bars.	
6. Verify that all data displayed within the element is visible, readable, correctly arranged and correctly labelled. All messages are readable and clearly labelled.	
7. Verify that the element behaves sensibly when window resizing occurs, and that it remains usable and data remains visible whenever possible. Resizing teh window has no detrimental effect on the console.	
8. Verify that the element updates promptly when responding to changes in data. New messages appear immediately.	
Fixes Required	Adjust the ordering so that the most recent message appears on the bottom line, as is the standard for text consoles.

Overview Tab	
Purpose	Displays a summary of the data related to the selected robot.
Required Functionality	Must display data related to the selected robot, including ID, name, state and position/orientation values. Must update immediately when new data is received.
1. Examine UI element visually. Verify that it appears correct. Verify that it contains all elements necessary to satisfy its purpose.	
The tab appears visually correct. The heading text appears redundant as the tab itself already contains the heading.	
2. Examine all text within the element. Check for errors in both meaning and spelling.	
All spelling correct and all meanings clear.	
3. Verify that all components within the element which perform actions in response to user input operate correctly.	
n/a.	
4. Verify that all components respond quickly to user input.	
n/a.	
5. Verify that component actions and functionality do not degrade with extreme use (sustained rapid input, large numbers of input changes, etc).	
n/a.	
6. Verify that all data displayed within the element is visible, readable, correctly arranged and correctly labelled.	
All of the data points are visible, readable, clear, correctly arranged and labelled.	
7. Verify that the element behaves sensibly when window resizing occurs, and that it remains usable and data remains visible whenever possible.	
All data is visible at the full range of window size.	
8. Verify that the element updates promptly when responding to changes in data.	
Updates to the data are reflected immediately.	
Fixes Required	Remove the overview heading from inside the tab, as it is redundant.

State Tab	
Purpose	Displays information related to the internal state of the robot.
Required Functionality	Must display a list of the selected robot's known states. Must display a list of the selected robots recent state changes, including timing information. Must update immediately when a state change occurs.
1. Examine UI element visually. Verify that it appears correct. Verify that it contains all elements necessary to satisfy its purpose.	
Appears visually correct. Both lists are present.	
2. Examine all text within the element. Check for errors in both meaning and spelling.	
All spelling correct and meanings clear.	
3. Verify that all components within the element which perform actions in response to user input operate correctly.	
Lists operate correctly and ignore selection events.	
4. Verify that all components respond quickly to user input.	
n/a.	
5. Verify that component actions and functionality do not degrade with extreme use (sustained rapid input, large numbers of input changes, etc).	
Large numbers of states and transition entries remain readable using scroll bars.	
6. Verify that all data displayed within the element is visible, readable, correctly arranged and correctly labelled.	
Known states are readable and clear. Transitions are readable but no always clear, due to the layout of the time-stamp. The clarity could be improved with better formatting.	
7. Verify that the element behaves sensibly when window resizing occurs, and that it remains usable and data remains visible whenever possible.	
resizing has no detrimental effect on the tab. Scroll bars are available when necessary.	
8. Verify that the element updates promptly when responding to changes in data.	
New states and transitions are displayed immediately.	
Fixes Required	Re-format the state transition list entries to improve clarity.

IR Data Tab	
Purpose	Displays the IR sensor data for the selected robot.
Required Functionality	Must provide a visual display of the selected robot's IR sensor values. Must provide a numerical display of the robot's IR sensor values. Must support both active and background values. Must update immediately when new values arrive.
1. Examine UI element visually. Verify that it appears correct. Verify that it contains all elements necessary to satisfy its purpose. The element is lacking clear headings for some of the data points. The bar graph is visually correct. The numerical displays are correct but their current format takes up a lot of space.	
2. Examine all text within the element. Check for errors in both meaning and spelling. All spelling correct. Some headings have unclear meanings.	
3. Verify that all components within the element which perform actions in response to user input operate correctly. n/a.	
4. Verify that all components respond quickly to user input. n/a.	
5. Verify that component actions and functionality do not degrade with extreme use (sustained rapid input, large numbers of input changes, etc). Sensor values that are out of range are not displayed.	
6. Verify that all data displayed within the element is visible, readable, correctly arranged and correctly labelled. All data is readable and correctly arranged. The IR data is not clearly labelled.	
7. Verify that the element behaves sensibly when window resizing occurs, and that it remains usable and data remains visible whenever possible. Some of the data becomes hidden if the window width is reduced.	
8. Verify that the element updates promptly when responding to changes in data. New data is reflected immediately.	
Fixes Required	Add clear headings and reformat the numerical data to improve clarity. Rearrange the layout so that the bar graph fits within the minimum window width.

Custom Data Tab	
Purpose	Displays custom data reported by the selected robot.
Required Functionality	Must display each custom data point in, including both key and value strings. Must update immediately when new data is received.
1. Examine UI element visually. Verify that it appears correct. Verify that it contains all elements necessary to satisfy its purpose. The tab is visually correct, including a table to display the custom data key/value pairs.	
2. Examine all text within the element. Check for errors in both meaning and spelling. All spelling correct and meanings clear.	
3. Verify that all components within the element which perform actions in response to user input operate correctly. The table cannot be modified using the mouse or keyboard, as intended.	
4. Verify that all components respond quickly to user input. n/a.	
5. Verify that component actions and functionality do not degrade with extreme use (sustained rapid input, large numbers of input changes, etc). Long keys and values and large numbers of key/value pairs are handled gracefully with scroll bars.	
6. Verify that all data displayed within the element is visible, readable, correctly arranged and correctly labelled. All data is displayed clearly. The columns of the table are clearly labelled.	
7. Verify that the element behaves sensibly when window resizing occurs, and that it remains usable and data remains visible whenever possible. The table scales to fit the available space, providing scroll bars whenever necessary.	
8. Verify that the element updates promptly when responding to changes in data. New and updated custom data is displayed immediately.	
Fixes Required	None.

Individual Visualisation Settings Dialogs	
Purpose	Allow the user to change settings for the data visualisations.
Required Functionality	Must display the settings for the selected visualisation type in a pop-up, modal window. Must allow the user to change these settings. Must provide the user with options for applying or cancelling the changes.
1. Examine UI element visually. Verify that it appears correct. Verify that it contains all elements necessary to satisfy its purpose.	
All settings dialogs appear correctly, containing all the necessary controls to adjust the available settings.	
2. Examine all text within the element. Check for errors in both meaning and spelling.	
All spelling correct and all meanings clear.	
3. Verify that all components within the element which perform actions in response to user input operate correctly.	
All settings controls working correctly.	
4. Verify that all components respond quickly to user input.	
Settings changes take effect immediately after pressing the apply button.	
5. Verify that component actions and functionality do not degrade with extreme use (sustained rapid input, large numbers of input changes, etc).	
All components appear robust to repeated, rapid use. Input fields are validated to reject out of range values.	
6. Verify that all data displayed within the element is visible, readable, correctly arranged and correctly labelled.	
All settings are clearly displayed, labelled and arranged.	
7. Verify that the element behaves sensibly when window resizing occurs, and that it remains usable and data remains visible whenever possible.	
Dialog windows cannot be resized.	
8. Verify that the element updates promptly when responding to changes in data.	
n/a.	
Fixes Required	None.

General Visualiser Functionality	
Purpose	Displays the live video feed and overlays the the data visualisations.
Required Functionality	Must display the video feed. Must render the data visualisations based on the current data. Must allow the user to select a robot by clicking on it within the image.
1. Verify that the video image is displayed correctly.	
The video feed image is displayed correctly, and updates at a decent rate. Due to the mounting orientation of the camera the image appears reversed when compared to the real world view. Settings could be added to allow the user to flip the image.	
2. Verify that user clicks within the visualiser space are located correctly, at a number of different window sizes.	
User clicks are correctly located as shown by the cross-hairs showing the latest click location. Tested for a number of different sizes and image dimensions / aspect ratios, and worked correctly each time.	
3. Verify that robots can be selected by clicking on their location in the visualiser image.	
Robots can be selected by clicking, with a reasonable tolerance. Robots positioned directly adjacent to one another can still be selected correctly. Clicking a robot with ID 10 caused a robot with ID 1 to be selected erroneously.	
Fixes Required	Add setting to allow the user to flip the video image. Robot selection works incorrectly for robots with IDs beginning with the same digit; determine the cause and fix.

Robot ID Visualisation	
Purpose	Overlay the numerical ID of the robot on the video feed.
Required Functionality	Display the numerical ID as a text string adjacent to the related robot.
Settings	On / off (Toggle). Display for selected robot only or all robots (Toggle).
1. Define input data and expected representation.	
Five active robots using ID's 0, 1, 5, 7 and 8. expect to see the ID numbers rendered to the left of each robot, slightly above their center.	
3. Verify the representation is as expected.	
Numerical ID numbers appear next to each robot. The ID is positioned slightly too far to the left of each robot, sometimes overlapping with other visualisations in crowded areas. Settings apply correctly.	
4. Check that the visualisation is clear and any text is legible.	
ID numbers are legible.	
5. Repeat for multiple sets of input data.	
Tested with five robots.	

6. Verify that integrity is maintained with extreme data, corner cases and zero data, wherever possible. Robot IDs higher than 999 result in long strings, which overlap with the robot itself and its position/direction visualisation. Robot swarm of this size are not anticipated however.	
7. Verify that integrity is maintained at a range of window sizes, within reasonable limits. IDs display at the same size for all visualiser sizes, therefore remaining legible but occupying excessive space at small sizes. However the visualiser is not usable at such small sizes, so this is not anticipated to cause issues.	
Fixes Required	Reposition IDs slightly closer to the robot. Potentially allow the user to configure the positioning.

Robot Name Visualisation	
Purpose	Overlay the name of the robot on the video feed.
Required Functionality	Display the name of the robot as a text string adjacent to the related robot.
Settings	On / off (Toggle). Display for selected robot only or all robots (Toggle).
1. Define input data and expected representation.	
Five active robots reporting the names Robot_0, Robot_1, Robot_5, Robot_7 and Robot_8 in watchdog packets. Expect to see the names rendered to the right of each robot, slightly above their center.	
3. Verify the representation is as expected.	
Names appear next to each robot. The text is positioned correctly to the right of each robot. Settings apply correctly.	
4. Check that the visualisation is clear and any text is legible.	
Names are legible.	
5. Repeat for multiple sets of input data.	
Tested with five robots.	
6. Verify that integrity is maintained with extreme data, corner cases and zero data, wherever possible.	
Extremely long names are displayed up to the edge of the image.	
7. Verify that integrity is maintained at a range of window sizes, within reasonable limits.	
Names remain legible for all usable sizes of the visualiser.	
Fixes Required	None.

Robot State Visualisation	
Purpose	Overlay the current state of the robot on the video feed.
Required Functionality	Display the state of the robot as a text string adjacent to the related robot. Update this display whenever the state changes.
Settings	On / off (Toggle). Display for selected robot only or all robots (Toggle).
1. Define input data and expected representation.	
Five active robots oscillating between STATE1 and STATE2. Expect to see the states rendered to the right of each robot, below the name visualisation.	
3. Verify the representation is as expected.	
States appear next to each robot. The text is positioned correctly to the right of each robot and below the name text. Settings apply correctly.	
4. Check that the visualisation is clear and any text is legible.	
States are legible.	
5. Repeat for multiple sets of input data.	
Tested with five robots, each changing state at different times.	
6. Verify that integrity is maintained with extreme data, corner cases and zero data, wherever possible.	
Extremely long states are displayed up to the edge of the image.	
7. Verify that integrity is maintained at a range of window sizes, within reasonable limits.	
States remain legible for all usable sizes of the visualiser.	
Fixes Required	None.

Robot Position Visualisation	
Purpose	Overlay a small circle around the robot's current position.
Required Functionality	Render a circle outline around the robots current position. Use a thicker line to highlight the robot if it is currently selected.
Settings	On / off (Toggle).
1. Define input data and expected representation. Five active robots moving around the arena in different paths. Expect to see a circle rendered around the center of each robot, updating as their positions change.	
3. Verify the representation is as expected. Circles are correctly rendered for all robots, and update correctly over time. Can be correctly toggled on and off.	
4. Check that the visualisation is clear and any text is legible. Circles are clear.	
5. Repeat for multiple sets of input data. Tested with five robots.	
6. Verify that integrity is maintained with extreme data, corner cases and zero data, wherever possible. Circles display correctly for all positions within the image.	
7. Verify that integrity is maintained at a range of window sizes, within reasonable limits. Circles render correctly at all sizes.	
Fixes Required	None.

	Robot Direction Visualisation
Purpose	Overlay a small line indicating the direction the robot is facing.
Required Functionality	Render a line from the center of the robot outwards, in the direction it is facing. Use a thicker line if the robot is selected.
Settings	On / off (Toggle).
1. Define input data and expected representation. Five active robots moving around the arena in different paths. Expect to see a line rendered from the center of each robot outward in the direction it is facing, updating as its orientation changes.	
3. Verify the representation is as expected. Lines are correctly rendered for all robots, and update correctly over time. Can be correctly toggled on and off.	
4. Check that the visualisation is clear and any text is legible. Lines are clear.	
5. Repeat for multiple sets of input data. Tested with five robots.	
6. Verify that integrity is maintained with extreme data, corner cases and zero data, wherever possible. Lines display correctly for all orientations, at all positions within the image.	
7. Verify that integrity is maintained at a range of window sizes, within reasonable limits. Lines render correctly at all sizes.	
Fixes Required	None.

IR Data Visualisation	
Purpose	Overlay a graphical representation of the robots infra-red sensor data.
Required Functionality	IR sensor data represented in one of two modes. In proximity mode, display a line in the direction of each relevant sensor with a length inversely related to the sensors value, indicating an approximation of proximity. In 'heat' mode, display a small box for each sensor adjacent to the robot and positioned to match the sensor layout, that changes colour as the sensor value changes. Uses the position and orientation of the robot to render with the correct position and rotation.
Settings	On / off (Toggle). Display for selected robot only or all robots (Toggle). Proximity or heat mode (Toggle). Angle for each sensor in degrees (Numerical).
1. Define input data and expected representation. Five active robots reporting their IR sensor data whilst an object is placed at varying distances from each of their sensors. Expect to see the graphical representations change to reflect the changing value.	
3. Verify the representation is as expected. IR data is displayed in both modes. The display varies correctly as the sensor values change. Proximity lines change length correctly, but extend much further than necessary for low values. The boxes in heat mode change colour correctly, but are coloured close to black for low values, which does not display well on a dark background.	
4. Check that the visualisation is clear and any text is legible. Both visualisations are clear.	
5. Repeat for multiple sets of input data. Tested with 5 robots across a range of sensor values.	
6. Verify that integrity is maintained with extreme data, corner cases and zero data, wherever possible. Out of range sensor data is not displayed.	
7. Verify that integrity is maintained at a range of window sizes, within reasonable limits. The size of the visualisation does not change with the size of the image. However the proximity lines can only be representative, so their actual size is not important, only the relative variation in that size.	
Fixes Required	Set a more sensible maximum length for the proximity lines. Improve mapping between sensor values and line length (non-linear). Adjust heat mode colour scheme for clarity.

Robot Path Visualisation	
Purpose	Display the robots recent movement in the form of a trail.
Required Functionality	Render the robot's position history as a sequence of line segments. support an adjustable sampling interval.
Settings	On / off (Toggle). Display for selected robot only or all robots (Toggle). Sampling interval (Numerical).
1. Define input data and expected representation. Five active robots moving around the arena along different paths. Expect to see a trail line behind each robot showing the path it has taken.	
3. Verify the representation is as expected. Trails are correctly drawn for all robots, accurate to their approximate path. Varying the sample interval allows for longer, lower resolution and shorter, higher resolution paths. Can be correctly toggled on and off, and set to only display for the selected robot.	
4. Check that the visualisation is clear and any text is legible. Trails are clearly drawn.	
5. Repeat for multiple sets of input data. Tested with 5 robots moving along a number of different paths.	
6. Verify that integrity is maintained with extreme data, corner cases and zero data, wherever possible. Setting an excessively large sampling interval leads to a very long, jagged path, as expected.	
7. Verify that integrity is maintained at a range of window sizes, within reasonable limits. Paths remain accurate for all visualiser sizes, as path point positions are stored as proportional coordinates.	
Fixes Required	Add an upper limit to the sampling interval setting.

Custom Data Visualisation	
Purpose	Display a specific element of the robot's custom data as text.
Required Functionality	Must display the key and the current value for the target data point as text adjacent to the robot, below the state text. Must update whenever the value for that key changes. The user must be able to set the target key.
Settings	On / off (Toggle). Display for selected robot only or all robots (Toggle). Set the target data point (Text input).
1. Define input data and expected representation. Five robots, each reporting custom data for two keys, with the values varying over time. Expect to see the target data key and value displayed as text below the state text.	
3. Verify the representation is as expected. Custom data for the target key is correctly displayed, and updates immediately when new data arrives. The target key can be changed and the visualisation updates to reflect this. Can be toggled on and off correctly.	
4. Check that the visualisation is clear and any text is legible. Text is clear and legible.	
5. Repeat for multiple sets of input data. Tested with 5 robots reporting data for two different keys.	
6. Verify that integrity is maintained with extreme data, corner cases and zero data, wherever possible. If no key is set, no data is displayed. Very long data values are displayed up to the edge of the image. If no data exists for the target key and the selected robot no data is displayed.	
7. Verify that integrity is maintained at a range of window sizes, within reasonable limits. Remains visible at all usable visualiser sizes.	
Fixes Required	None.

Bibliography

[1] M. Dorigo, M. Birattari, and M. Brambilla, "Swarm robotics", *Scholarpedia*, vol. 9, no. 1, p. 1463, 2014.

[2] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli, "The e-puck, a robot designed for education in engineering", *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, vol. 1, no. 1, pp. 59–65, 2009.

[3] E. Sahin, "Swarm robotics: From sources of inspiration to domains of application", in *Swarm Robotics WS 2004*, E. S. amd W. M. Spears, Ed., Berlin: Springer, 2005, pp. 10–20.

[4] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm intelligence: From natural to artificial systems*. Oxford University Press, 1999, ISBN: 0-19-513159-2.

[5] M. Kelly, M. Lutz, and C. Reid, *Army ants' 'living' bridges span collective intelligence, 'swarm' robotics (PNAS)*, `https://blogs.princeton.edu/research/2015/11/24/army-ants-living-bridges-span-collective-intelligence-swarm-robotics-pnas/`, Research at Princeton Blog, 2014.

[6] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: A review from the swarm engineering perspective", in *Swarm Intelligence*, 1, vol. 7, Springer, 2013, pp. 1–41.

[7] A. Kolling, P. Walker, N. Chakraborty, K. Sycara, and M. Lewis, "Human interaction with robot swarms: A survey", *IEEE Transactions on Human-Machine Systems*, vol. 46, no. 1, pp. 9–26, 2016.

[8] A. Rule and J. Forlizzi, "Designing interfaces for multi-user, multi-robot systems", *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pp. 97–104, 2012.

[9] R. E. Christ, "Research for evaluating visual display codes: An emphasis on colour coding", *Information design: The design and evaluation of signs and printed materials*, pp. 209–228, 1984.

[10] C. Carney and J. L. Campbell, "In vehicle display icons and other information elements: Literature review", U.S. Department of Transportation, Federal Highway Administration, Tech. Rep., 1998, pp. 209–228.

[11] G. Podevijn, R. O'Grady, and M. Dorigo, "Self-organised feedback in human swarm interaction", in *Proceedings of the workshop on robot feedback in human-robot interaction: How to make a robot readable for a human interaction partner (Ro-Man 2012)*, 2012.

[12] C. Q. Nguyen, B.-C. Min, E. T. Matson, A. H. Smith, J. E. Dietz, and D. Kim, "Using mobile robots to establish mobile wireless mesh networks and increase network throughput", *International Journal of Distributed Sensor Networks*, vol. 8, no. 8, 2012.

[13] J. McLurkin, J. Smith, J. Frankel, D. Sotkowitz, D. Blau, and B. Schmidt, "Speaking swarmish: Human-robot interface design for large swarms of autonomous mobile robots.", in *AAAI spring symposium: To boldly go where no human-robot team has gone before*, 2006, pp. 72–75.

[14] T. H. J. Collet and B. A. MacDonald, "An augmented reality debugging system for mobile robot software engineers", *Proceedings 2006 IEEE International Conference on Robotics and Automation*, pp. 3954–3959, 2006.

[15] L. Gumbley and B. A. MacDonald, "Realtime debugging for robotics software", *Australasian Conference on Robotics and Automation (ACRA)*, 2009.

[16] B. Annable, D. Budden, and A. Mendes, "Nubugger: A visual real-time robot debugging system", in *RoboCup 2013: Robot World Cup XVII*, S. Behnke, M. Veloso, A. Visser, and R. Xiong, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 544–551.

[17] R. T. Azuma, "A survey of augmented reality", *Foundations and Trends nin Human-Computer Interaction*, vol. 8, no. 2-3, pp. 73–272, 1997.

[18] Sayduck Ltd. (2016). Sayduck - make your space beautiful. `https://sayduck.com/`, (visited on 05/15/2017).

[19] Microsoft Corporation. (2017). Microsoft hololens. `https://www.microsoft.com/en-gb/hololens`, (visited on 05/15/2017).

[20] G. S. Pandher, *Microsoft hololens preorders: Price, specs of the augmented reality headset*, `https://www.thebitbag.com/microsoft-hololens-preorders-price-specs-of-the-augmented-reality-headset/137410`, Accessed: 2017-05-03.

[21] M. B. A. Clark and G. Lee, "A survey of augmented reality", *Presence: Teleoperators and Virtual Environments*, vol. 6, no. 4, pp. 355–385, 2014.

[22] P. Milgram, D. Zhai S. Drascic, and J. Grodski, "Applications of augmented reality for human-robot communication", *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems '93, IROS '93.*, vol. 3, pp. 1467–1472, 1993.

[23] M. Daily, Y. Cho, K. Martin, and D. Payton, "World embedded interfaces for human-robot interaction", in *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*, 2003.

[24] F. Ghirighelli, J. Guzzi, G. A. Di Caro, V. Caglioti, L. M. Gambardella, and A Giusti, "Interactive augmented reality for understanding and analyzing multi-robot systems", *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1195–1201, 2014.

[25] S. Garrido-Jurado, R. Munos-Salinas, F. J. Madrid-Cuevas, and M. J. Marin-Jimenez, "Automatic generation and detection of highly reliable fiducial markers under occlusion", *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014.

[26] W. Liu and A. F. T Winfield, "Open-hardware e-puck linux extension board for experimental swarm robotics research", *Microprocessors and Microsystems*, vol. 35, no. 1, pp. 60–67, 2011.

[27] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo, "ARGoS: A modular, parallel, multi-engine simulator for multi-robot systems", *Swarm Intelligence*, vol. 6, no. 4, pp. 271–295, 2012.

[28] Stemmer Imaging Ltd. (2014). JAI Go machine vision camera, [Online]. Available: `https://www.stemmer-imaging.co.uk/en/products/series/jai-go/` (visited on 04/29/2017).

[29] Stemmer Imaging. (2011). GIGE Vision in practice. `https://www.stemmer-imaging.co.uk/media/uploads/websites/documents/products/en-GigE-Vision-in-Practice-TTUe3-201312.pdf`, (visited on 05/16/2017).

[30] B. O'Sullivan, "Making sense of revision-control systems", *Commun. ACM*, vol. 52, no. 9, pp. 56–62, 2009.

[31] L. Allen, G. Fernandez, K. Kane, D. Leblang, D. Minard, and G. McLean, *Version control system for geographically distributed software development*, US Patent 5,675,802, 1997. [Online]. Available: `https://www.google.com/patents/US5675802`.

[32] S. Chacon, *Pro git*, 1st. Berkely, CA, USA: Apress, 2009, ISBN: 1430218339, 9781430218333.

[33] Github Inc. (2017). Github: About, [Online]. Available: `https://github.com/about` (visited on 05/11/2017).

[34] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato, "Version control with subversion", 2002.

[35] B. O'Sullivan, *Mercurial: The definitive guide: The definitive guide*, ser. Animal Guide. O'Reilly Media, 2009, ISBN: 9780596555474.

[36] The Qt Company Ltd. (2017). Qt creator manual. `http://doc.qt.io/qtcreator/`, (visited on 05/11/2017).

[37] JetBrains s.r.o. (2017). CLion; a cross platform ide for C and C++. `https://www.jetbrains.com/clion/`, (visited on 05/11/2017).

[38] J. Simmer. (2017). Sublime Text: The text editor you'll fall in love with. `https://www.sublimetext.com/`, (visited on 05/11/2017).

[39] The Agile Alliance. (2017). Agile 101: What is agile software development? `https://www.agilealliance.org/agile101/`, (visited on 05/14/2017).

[40] D. J. Armstrong, "The quarks of object-oriented development", *Communications of the ACM - Next-generation cyber forensics*, vol. 49, no. 2, pp. 123–128, Feb. 2006.

[41] S. Burbeck, "Applications programming in smalltalk-80: How to use model-view-controller (mvc)", *Smalltalk-80*, vol. 2, no. 5, 1992.

[42] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov, "Real-time computer vision with OpenCV", *Commun. ACM*, vol. 55, no. 6, pp. 61–69, 2012.

[43] C. Meinel and H. Sack, "The foundation of the internet: TCP/IP reference model", in *Internetworking: Technological Foundations and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 29–61.

[44] J. Postel, "User datagram protocol", RFC 768, 1980. [Online]. Available: `https://tools.ietf.org/html/rfc768`.

[45] T. Bray, "The JavaScript object notation (JSON) data interchange format", RFC 7159, 2014. [Online]. Available: `https://tools.ietf.org/html/rfc7159`.

[46] D. Huizinga and A. Kolawa, *Automated defect prevention: Best practices in software management*, ser. Wiley - IEEE. Wiley, 2007.

[47] B. Boehm, "Software risk management", in *ESEC '89: 2nd European Software Engineering Conference University of Warwick, Coventry, UK September 11–15, 1989 Proceedings*, C. Ghezzi and J. A. McDermid, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 1–19.

[48] The Qt Company Ltd. (2017). Qt reference pages. `http://doc.qt.io/qt-5/reference-overview.html`, (visited on 05/13/2017).