Autonomous Mobile Robot

MSc. Software Engineering 2013/2014

University of St Andrews

By:

Adebayo Osipitan

Supervised by:

Michael Weir

Acknowledgement

I want to acknowledge GOD almighty for his Grace has kept me and seen me through this project and my entire study duration at the University of St Andrews. I would also like to thank my parents, Mr and Mrs A.I Osipitan and my brother and sisters for all their support and love.

I would like to acknowledge my supervisor Michael Weir, the Fixit and systems team – Jim Park and Stuart – as well as the MSc Coordinator Tom Kelsey for their continuous support, making sure I was continuously on track with this project. I would also like to acknowledge my friends Adeola Fabola and Oche J. Ejembi for their continuous encouragement

Declaration

I, Adebayo Osipitan, hereby declare that this report documentation which is 12185 words in length submitted for assessment is my own work, it is the record of work carried out by me and that it has not been submitted in any previous application for a higher degree, and credit is explicitly given to others by citation or acknowledgement.

This project was conducted by me at The University of St Andrews from June 2014 to August 2014 towards fulfilment of the requirements of the University of St Andrews for the degree of MSc under the supervision of Michael Weir.

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work.

Adebayo I. Osipitan

22/08/2014

[illegible]

Abstract

A robot is defined as "*a machine system that can carry out a series of complex actions automatically, especially one programmable by a computer*" and was coined from the word "*Robota*", which means forced labour. Automation and creating intelligence has always been a major research area for man, with precursors such as the *Unimate* paving the way for modern robotics, which has evolved with the advancements in artificial intelligence and sophisticated technology. The report analyses the process of developing a sophisticated low powered mobile robotic system using relatively low priced components – sensors, controllers, camera etc. Facilitating intelligence and some form of autonomy in the robotic system, analysing motion planning techniques and computer vision techniques with goal tracking and obstacle avoidance as well as constraints to consider- like holonomy - when developing for mobile robotics. In the case of this project, a user remote controlled robot with a webcam for autonomous goal tracking was facilitated.

Table of Contents

List of Figures

List of Tables

Abbreviations

- DOF – Degree of Freedom
- PET – Pulse Echo Technique
- PWM – Pulse Width Modulation
- GUI –Graphical User Interface
- USART – Universal Asynchronous Receiver/Transmitter
- SDA – Serial Data
- SCL – Serial Clock
- I^2C – Inter Integrated Circuit
- Opencv – Open source computer vision
- BGR – Blue-Green-Red
- RGB – Red-Green-Blue
- HSV – Hue Saturation Value
- WLAN - Wireless Local Area Network
- RF – Radio Frequency

(Word Count: 12031)

1 Introduction

This report serves as a form of documentation, for the development of an Autonomous Mobile Robotic Vehicle project, highlighting the development process cycle in terms of the project aims and objectives as well as the requirements for this system project. This project serves as research into robotics and their autonomy, with the development of a robotic vehicle interfaced with various sensors which enable it to be autonomous in its motion, or if the user desires can be remotely controlled via any computer with a COM port using wireless serial communication.

1.1 Project Overview

A robot is defined as "*a machine system that can carry out a series of complex actions automatically, especially one programmable by a computer*" and was coined from the word "*Robota*", which means forced labour (Oxford Dictionary 2014). The study and use of robots is called "*Robotics*" these words were created in 1941 by Isaac Asimov a scientist and writer who foresaw the era of robotics and even went further to coin the "*Laws of Robotics*" in one of his fictional novels "*Runaround*"(RobotShop 2008). Machine automation is a science that has been around for many years dating back to ancient Greece, with texts by Alexandria detailing automated movements (ThomasNet 2014). These automation precursors paved the way for modern robotics starting in the early 20th century, with *the* first industrial robot, the "*Unimate*" (meaning "*Universal Automation*"), invented in the 1950s by George Devol (RobotShop 2008). This engineering and computing pioneer later with Joseph Engelberger, formed the robotics company "*Unimation*" which led the early robotics era for decades. In the 21st century robotics has rapidly advanced and more so with the advancement of artificial intelligence in computing, and has thus graduated from being used for relatively simple industrial applications - assembly line operations that have pre-defined functions programmed into machines to speed up manufacturing processes -, to environments needing more intricate functionality, and also requiring *intelligence*, flexibility for a greater range of autonomy, and requiring less human interaction while carrying out complex tasks, while adapting to changing their environment and so on. Mobile robotics is the relatively new research area that has helped advance robotics from developing machines like simple manipulator systems to the complex systems that exhibit autonomy and higher intelligence than their predecessors. The topic of mobile robotics is the research area focused on by this project which involves the development of a robotic system, as well as developing algorithms for facilitating intelligence and autonomy. This report documents the processes followed during development of such a system, as well as recommends further improvements that can be made to the system, that may further enhance its autonomy in operation.

1.2 Project Motivation

Research on mobile robotics has evolved over the years such that they can be applied in a wider range, from complex systems such as those used in space exploration in the aerospace industry – Mars Rover 1996 till date, latest mars rover (Curiosity Rover) was launched by NASA in 2011 (NASA 2014) - to more socially commercial applications as with semi-autonomous drive-less cars, or fully autonomous drive-less car like the "*Google Car*", and even some house hold products like robot vacuums and cleaners (some having relatively autonomous operation) etc., these commercial systems operate using a variety of sensors and cameras, to visualize the world around their system, environmental perception by these systems serve as stimuli for real world responses. The purpose of this project is to build on that premise and develop low-powered design for an autonomous mobile robotic vehicle, by interfacing it with ultra-sonic sensors for obstacle detection and a camera for user visual perception of the robot's environment and also for tracking of a set goal or target. By detecting the distance of physical objects around the robot via the interfaced sensors, the mobile robot system will intelligently respond by planning what motion path will serve it best to take in reach its goal while approach its goal.

1.3 Project Aims and Objectives

The aim of this project is to develop an intelligent mobile robotic vehicle system, capable of tracking a goal (target) in free space while advancing towards the goal and also avoiding obstacles. To achieve such a system, the following objectives where laid out to be completed.

Project Objectives:

- Conduct research on mobile robotics.
- Conduct research on autonomy in mobile robot.
- Conduct extensive research on computer vision and image processing.
- Conduct extensive research on real time mobile robot motion planning.
- To identify and analyse components (hardware and software) best suited for the development of this project.
- To iteratively develop a target tracking system adaptable by a mobile robot using an interfaced webcam for computer vision.
- To iteratively developed a system of communication between the mobile robot and a controlling computer.

- To develop an intelligent motion planning algorithm that can be tested using Webot simulations.
- To create an graphical user interface that allows the user to visualise images received by the camera, that will be also be processed by the interface functionalities for target identification, as well as allowing user manipulator control of the mobile robotic system.
- To iteratively develop a real time mobile robot that adapts the motion planning algorithms created above for obstacle navigation in a real world environment.
- To interface the above mentioned mobile robot functionalities together.

1.4 Report Outline:

This report delivers a seven chapter review of the project progression, starting with this chapter, which defines the scope of this project with its high level deliverables. The second chapter gives a brief background summary on previous related work on the key areas focused on by this project. The third chapter describes the methodological approach undertaking in the development of this project. Chapter four is focused on stating the requirements specifications that govern the deliverables for this project. Chapters five and six give a retrospective view of the approach to the solution domain for this project. Chapter seven involves the critical evaluation of the project in terms its aims, objectives and requirements from an unbiased view. This focuses on the key achievements attained in terms of the objectives of this project, the drawbacks in the design and implementation process of this project, as well an overall summary of the report. This chapter introduces the project, detailing the aims, objective and overview of this project.

2 Literature Review

2.1 Computing in Mobile Robotics

Computation in robotic has always been "*The*" key factor when developing robots with any measure of autonomy, this is regardless of the actuators used or the sensors that get stimuli from a robot's environment. Programs written to control a robot's behaviour are significant such even the smallest of changes in a program function definition can lead to an overall behavioural change in a robotic system (Dudek & Jenkin 2010). Mobile robotics helps embody computational theories, representations and algorithms to be tested for feasibility in a physical environment, they also embody man's dream of creating physical agents that exhibit intelligence, a quest which is as old as ancient Greece – to create a machine that behaves like a man -. With the cost of developing robots decreasing by the decade, research on mobile robotics and its applications have gone further than it has since the conception of robotic theory, there is focus on computing architectures best adopted as controls system for mobile robots, which have yielded functional modules decomposition structures like;

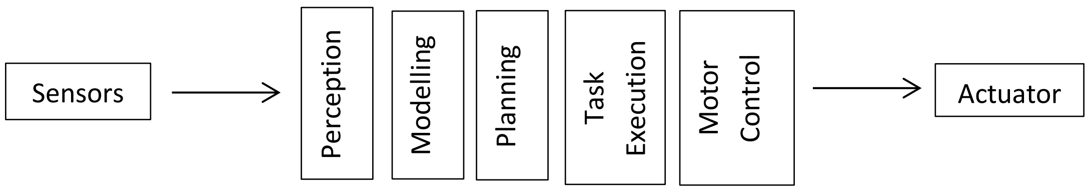


Figure 1: Decomposition of mobile robot control system into functional modules (R.A Brooks 1986)

To variants proposed by R.A Brooks (1986), based on "*Task-achieving Behaviour*"

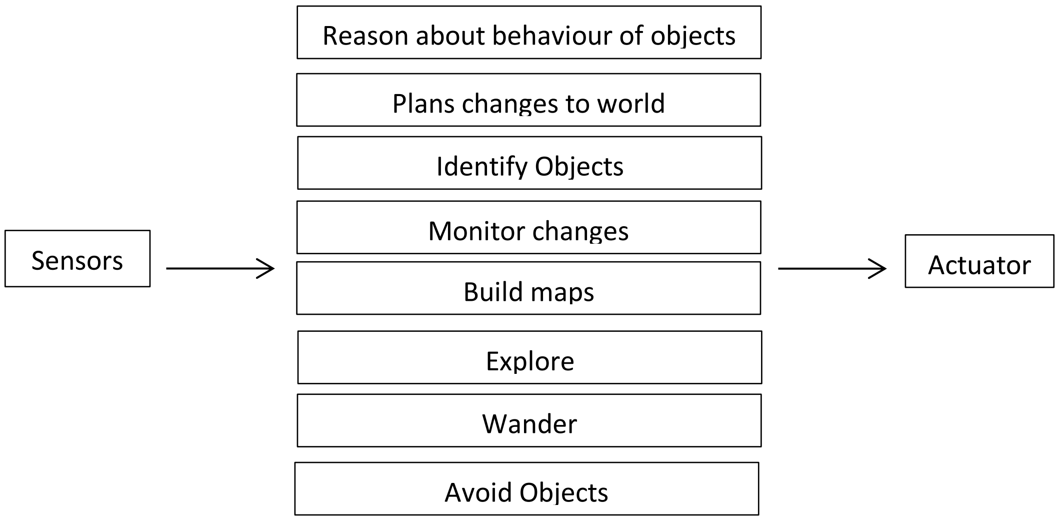


Figure 2: Task-achieving behaviour based decomposition proposed by R.A Brooks (1986).

To the "*distributed behaviour control*" system proposed by J.H. Reif and H. Wang (1998), to tackle control of large scaleswarm of mobile robots. The essence of a structured computing or control system is to enable mobile robots to efficiently coordinate their resources (both hardware and software) when completing complex tasks (that may be sometimes conflicting) in complex and dynamic environments. When developing a control system for an intelligent mobile robot for facilitating autonomy, certain requirements must be taken into consideration, these requirements include;

- **Prioritization of Multiple goals**: For most mobile robotic systems, single entity systems or swarm based systems, there are multiple goals these systems are trying to achieve and some of these goals often conflict in the real world, such as obstacle avoidance while tracking a goal or other members of a robotic swarm. Conflicting goals such as enabling long range direct communication using highly sophisticated hardware while maintaining power reserves, or even avoiding incoming moving obstacles while ensuring dedicated path continuation as with train systems. Control systems must be able to prioritize these goals so as to respond appropriately to real world complexities, responding and servicing both high and low priority goals (Brooks 1986).
- **Interfaced Sensor(s) Error**: To enable autonomy in a mobile robot, it means that the robot must have a way of receiving stimuli from its environment, to receive such stimuli, these robots are interfaced with various kinds of sensors – distance ranging sensors (infrared or ultrasonic), odometer sensors, beacons, communication peripherals, actuator mechanisms etc. – Sensor reading are often accompanied by errors with sensor reading being inaccurate due to varying real world factors like component composition, sensor overlap, outside domain acceptability. The control system must be able to account for such errors while planning response (Brooks 1986).
- **Robustness and Extensibility**: The control system must enable the robot to adjust to changing environments and also adapt to dysfunctional components – sensors – while still performing its normal function. The controller must also allow for extension of sensing and actuating capabilities of the robot, with effect of the computing power (Brooks 1986).

Fulfilling these requirement go beyond any simple computation that bring about a more reflexive responses to environmental stimuli, which is mostly a due to direct relationships between sensors and actuators (or effectors as the case may be), but more of reactive response, which allow the robot to respond appropriately to the dynamics of its environment. There have been many Control and computing system architectures that have been proposed and developed over the years including the "*Task Control Architecture*" which has been used for control architecture in advanced robotics projects such as a six-legged Ambler robot used for space exploration, and expected to traverse unknown terrain autonomously, incorporating in real- time motion planning goals, as well as error recovery and some sensory functionalities (perceptual, monitoring), mobile excavation robotic projects like a coal mining robot, a development and maintenance mobile robot systems by NASA, such systems to inspect space shuttles and assemble space stations (Simmons 1994). These varying control forms perform the task of creating an information flow from the environment, via the sensors, through a robotic system applied to back into the environment, via effectors in a closed loop (Brooks 1986). With the advancement in computing, robots that can have more intelligent reactions and more precise interpretation of stimuli from their environment has also evolved, allowing the applications of mobile robotics in exploration systems and also relatively low budget social systems – drive-less cars (Self driving cars), robot vacuum cleaners etc. – The subchapter below discusses how computing has help develop sophisticated control systems for these kinds of mobile robot.

2.2 Motion Planning

Motion planning in mobile robotics have been tackled in a varying number of ways, from line and track following robot path planning that guide these robots from one point to another to more evolved planning algorithms. The problem domain for motion planning research in robotics is based on obstacle avoidance during navigation, and was first (and still is) illustrated as "*Piano mover's problem*", a mathematical paradigm for determining the certainty of a given space (Livneh 2006) deriving a solution by investigating the space configuration of the piano where by it does not come in contact with obstacles in its path environment (Murray 1993).

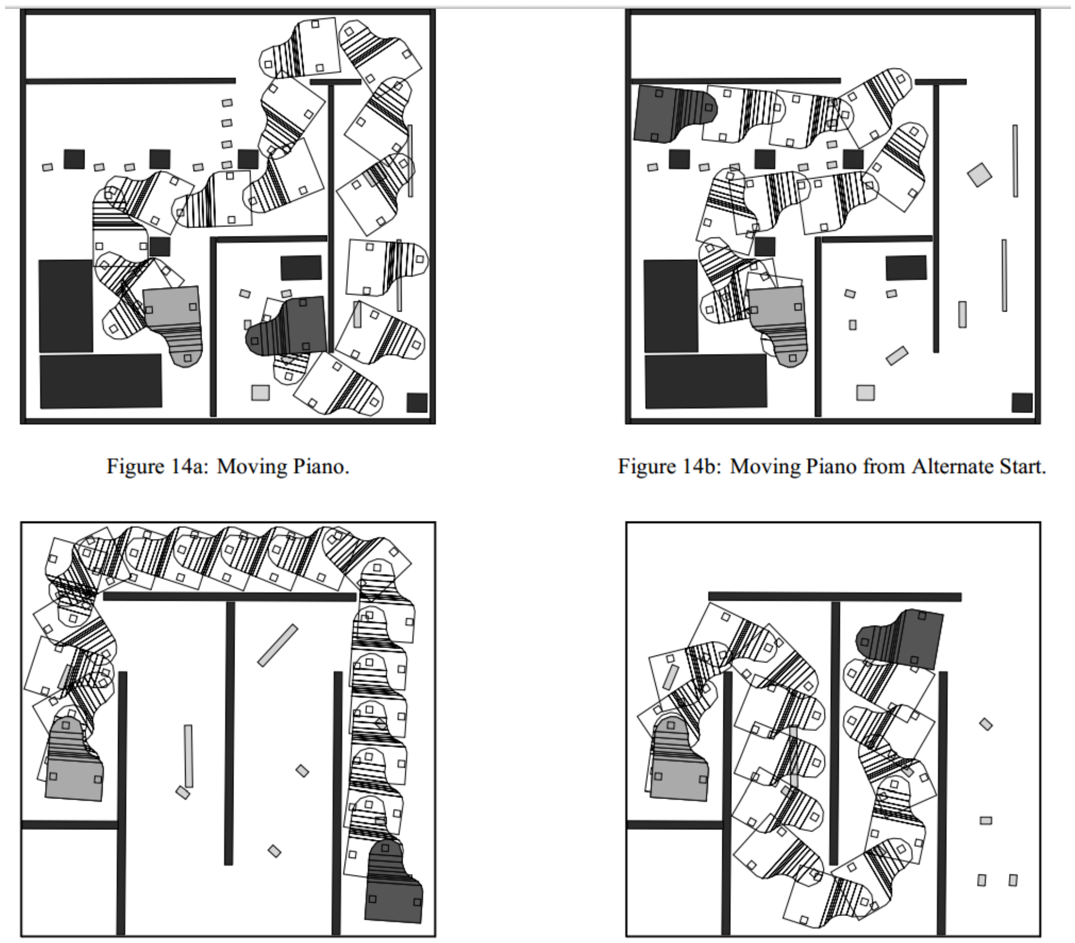


Figure 14a: Moving Piano.
Figure 14b: Moving Piano from Alternate Start.

Figure 3: Piano mover's problem (Lengyel et al 1990)

"*Given an arbitrary rigid polyhedral object, P, and polyhedral environment, find a continuous collision free path taking P from some initial configuration to a desired goal configuration.*" (Lengyel et al 1990)

The piano in this problem represents a mobile robot in an environment with obstacles (thick dark lines) navigating its way towards arbitrary goal points. There have been a number of algorithms developed as solutions to this problem including; configuration space and potential fields.

It is important to note that motion planning in mobile robotics is dependent on several kinematic constraints – including, the type of robotic system is being developed (Humanoid or vehicular), the degree of freedom (DOF) of actuators, positioning of the robot and control (Batlle and Barjau 2008).-. The scope of this project only will strictly cover motion planning for vehicular mobile robot systems, but even within this scope factors like the holonomy of the vehicular mobile robot system needs to be considered before developing any type of motion planning algorithms. The holonomy of a mobile robot system is the comparison of degrees of freedom for robotic movement. Depending on the wheel family driving a robotic vehicle – conventional and Omni-directional -, the degree of freedom of movement as well as the holonomy of such a vehicle is defined; that is to say for a given configuration space with x amount of DOF, the holonomy of a vehicle whose path is within such a space is dependent on the DOF allowed by the nature of the controls that can be applied as well as the conservation laws that apply to such a system. These constraints are important to note, because although the conventional implication of planned motion assumes that "*arbitrary motion in the configuration space is allowed as long as obstacles are avoided…*", there are limitations that may occur if a system contains non-holonomic constraints, and as a result many of these planners cannot be directly applied as paths generated maybe unfeasible to a non-holonomic system (Murray 1993). Considering the wheel design for this project (See Chapter five), this report is constrained to motion planning for conventional non-holonomic (configuration space with non-algebraic constraints) fixed driving wheels with only two DOF with a steering wheel. This type of system has two DOF types to consider, the DOF of the system ($\mathbf{DOF}_{\mathbf{system}}$) and the DOF of the frame ($\mathbf{DOF}_{\mathbf{frame}}$) and the difference between them is the DOF used to steer the wheels (Batlle and Barjau 2008). For such system kinematics it will be easier to consider potential fields method, in planning system motion amongst others such as elimination of local minima, and graph searching techniques, this is because potential fields combines both path finding and trajectory generation together in a closed loop although one method is most times insufficient in path planning, a paper by Tanner et al (2003) proposes using a variation of potential fields (*dipolar*) in combination with navigation functionality methodologies to form another variant of potential field methods they call "*dipolar inverse Lyapunov functions (DILFs)*", with guaranteed claims for effectively avoiding obstacles in non-holonomic systems, this eliminated the local minima weakness from using potential fields on its own.

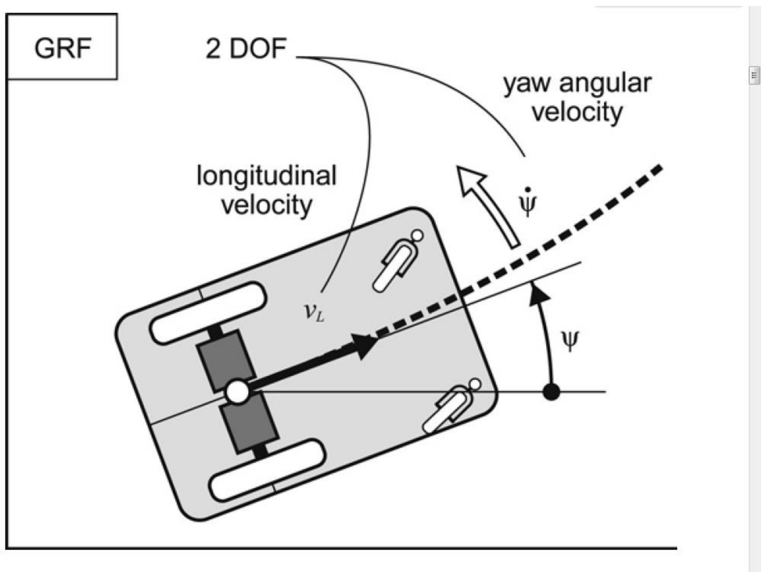


Figure 4: Mobile robot with 2 DOF using differential motors (Batlle and Barjau 2008)

2.2.1 Potential Fields Method

This method of path planning in robots was pioneered by Khatib and LeMaitre (Lengyel 1990), it involves creating mathematical models that represent both the robot and its environment (Tanner 2003), where obstacles can be represented as scalar analytic functions such;

$$f(x,y,z) = 0$$

Each obstacle is given its own a potential field that is inversely proportional in strength to the squared distance from such obstacle, (this theory is much like gravitational force theory for space-time continuation, where by any mas in space is said to have its own gravitational force that attracts other object to it), if at a maximum distance D_{max} where each obstacle's assigned potential no longer has influence. The Potential field for such an obstacle can be mathematically represented as:

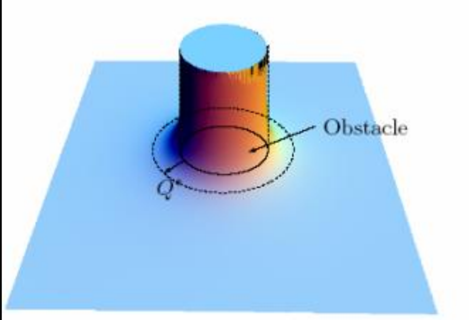
$$P_f(x,y,z) = \frac{\propto}{f(x,y,z)}$$

And by comparing f(x, y, z) with the maximum distance D_{max}:

$$f(x,y,z) \leq D_{max}\ ,\ P_f(x,y,z) > 0$$

$$f(x,y,z) > \ D_{max},\ P_f(x,y,z) = 0$$

Where by adding up potential fields of many such obstacles results in a single function in which a particle travelling within their potential field in accordance to normal laws of motion will not collide with any of these obstacles (Tanner 2003). By identifying peculiar points on real-life obstacles, assigning potential to these points, then linearly combining them an overall potential for a rigid body that gives position and orientation in a space constraint can be created. This can be addressed as a repulsion field of such an object.



$$U_{\text{rep}}(q) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{D(q)} - \frac{1}{Q^*}\right)^2, & D(q) \leq Q^*, \\ 0, & D(q) > Q^*, \end{cases}$$

whose gradient is

$$\nabla U_{\text{rep}}(q) = \begin{cases} \eta\left(\frac{1}{Q^*} - \frac{1}{D(q)}\right)\frac{1}{D^2(q)}\nabla D(q), & D(q) \leq Q^*, \\ 0, & D(q) > Q^*, \end{cases}$$

Figure 5: Repulsion potential of as rigid body (Howie Choset 2014)

2.3 Target Tracking

In most mobile robotics applications, the system will have goal it is trying to reach, or if acting as a part of a swarm of robots, there will be a need to keep track of the other members of the robot colony. There have been various research work done for tracking systems in mobile robotics, especially real-time tracking of known goals or acquired goals in its environment (be it a static, dynamic, structured or unstructured environment). Goal or Target tracking require sophisticated sensors that are efficient at acquiring accurate environmental data for control processing and motion planning for obstacle avoidance, odometry information about the mobile robot etc. (Li et al 2004). Sensors used for tracking, goals and targets span from infra-red sensors, position determining sensors for large scale environments , to video sensors for relatively short range targets.

2.3.1 Image processing for robot vision

With the increase in video sensors and their sophistication possibilities is using video in enhancing solutions for target tracking dynamics have increased, as previously, if images were used to locate targets only a single or static image was used. Now the use of video streams is being considered the norm in identification and tracking of a target (Lipton et al 1998). There are various techniques is processing an image. First to describe an image, an image in computing is usually expressed in the form of a matrix that represents properties such as intensity and colour.

With The sophistication of hardware technologies, software technologies have also been developed to aid image processing especially in computer vision, most notably Opencv and MATLAB. For this project the software package limitation is on Opencv. Opencv means Open Source Computer Vision and was developed by IBM for research on computer vision.

It has a BSD license so it free for academic use, it supports platforms such as C++, C, Python and Java (Android) interfaces and supports Windows, Linux, Android, iOS and Mac OS. It was launched in 1999, and since then over 500 algorithms have been developed for use with opencv including (OpenCV 2014):

- Tracking
- Image Pyramids
- Camera calibration, stereo and 3-D
- Matrix math
- Machine learning
- Transforms
- Fitting

Two of the most common place techniques used for almost all image processing applications as well as for tracking using the Opencv package are as follows:

Gray-Scale conversion

This is when an image is converted from its true-colour space to grey-scale, using a noninvertible simple image transform, this essentially reducing the information in the image yet maintaining feature related information (Solomon and Breckon, 2011) (E.I, AKANGSON. 2013).

A grayscale of a colour spaced image is calculated as an average of all three components arrays i.e. Grayscale, I = Average (R, G, B) = ((R + G + B)/ 3)

Edge Detection

The outline of a figure or object is a prominent features in any object and are key in human vision, as a figure can be completed recreated from a few of its prominent lines (Burger and Burge 2009). This is why image filtering processes are highly exploited for edge detection as features can easily be extracted by segmentation; simply stated "*edge detection is a method of segmenting an image into regions based on discontinuities, therefore if the edges of an image can be found effectively it has been segmented*" (Solomon and Breckon, 2011) (E.I, AKANGSON. 2013)..

3 Project Methodology

3.1 Project Management

For the project in its entirety, an IT engineering project management approach was employed for the project, with the project decomposed into stages as discussed briefly below:

- **Project Conception**: Here, the idea for the project to develop an autonomous mobile robot was created and approved, and the aim, objectives and scope were clearly identified.
- **Project planning**: This stage involved developing a plan and outline for the project, which were documented and made available to the supervisors, milestone identification were also discussed at this stage. The project plan document outlined the project deliverables, task breakdown, schedule and milestones as identified these served as inputs into the next stage of the project.
- **Project execution and monitoring**: After the planning stage, execution of the project development (i.e. building the mobile robot) cycle. There was monitoring and feedback – in form of emails and progress reports – took place at this stage to ensure that the execution was going according to plan, in line with the objectives and scope of the project.
- **Project closing**: After testing of the developed mobile robotic system, a finalised design and implementation documentation for the mobile robot will be submitted for analysis by supervisors. The diagram below summarises the stages that the project passed through from initiation to closure.

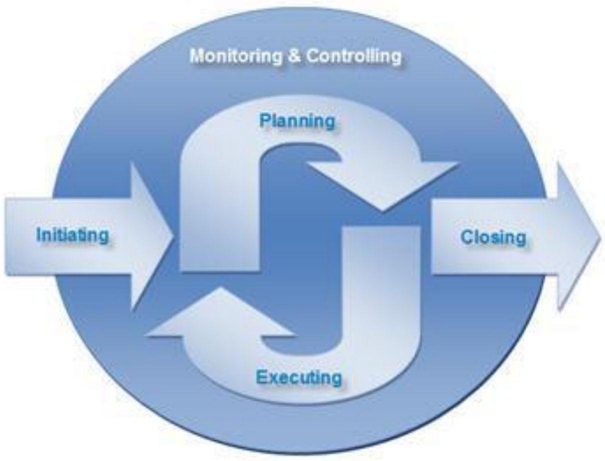


Figure 6 *IT project management approach. (McGill University 2013)*

3.2 Development Approach

For developing the system itself (Hardware and Software), a more systems engineering approach was employed, this involved a top-down development process:

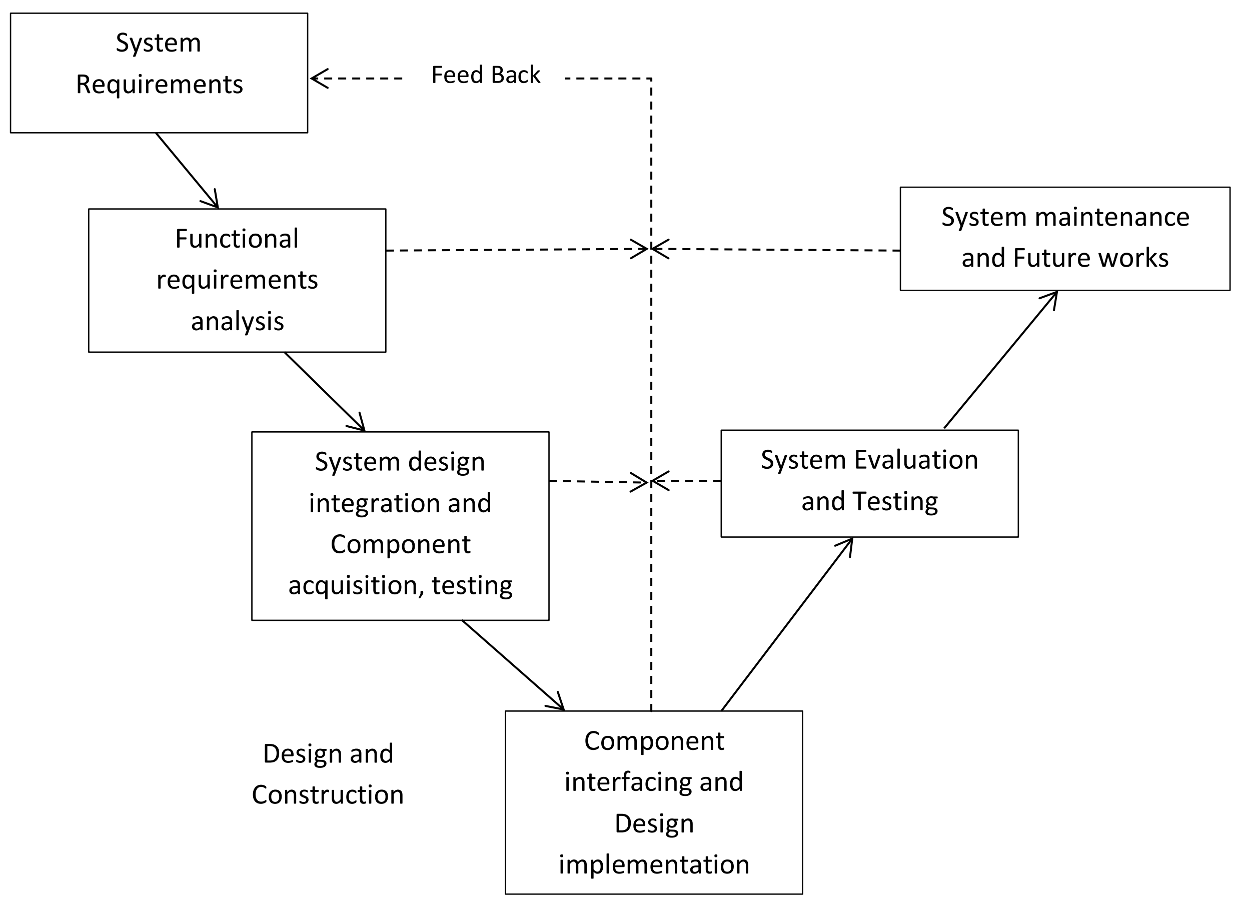

System Requirements
Feed Back
Functional requirements analysis
System maintenance and Future works
System design integration and Component acquisition, testing
System Evaluation and Testing
Design and Construction
Component interfacing and Design implementation

Figure 7: Top-Down/Bottom-up system development process (Blanchard 2004)

As shown above the approach employed for development of the system is an iterative one, with the stages briefly explained below;

- **System Requirements**: The requirements for the system where captured in this stage, that is in terms, of what actually makes up a mobile robot system – Sensors, effector, actuators, control unit etc. - , this stage is required for component identification, and also system design.
- **Functional Requirements Analysis**: The requirements for the mobile robotic system were captured after discussions with supervisor, and analysing previous work done on this project. This enabled the development of a plan (and outline) for the project.
- **Design integration and Component acquisition, Testing and Evaluation**: After studying the captured requirements and previous work, decisions were made concerning

the components to be used, along with their acquisition (and testing), the most suitable development environment, system architecture and the hardware and software requirements for mobile robot system. Comprehensive designs (see Chapter five) were developed based on these decisions.

- **Component interfacing and Design implementation**: Once the desired components have been acquired and tested, they need to be interfaced together as in the system design, and then the system functionalities can then be implemented. This implementation was split into four major parts namely;
    - Locomotion
    - Target/Goal Tracking
    - Communication
    - Motion Planning

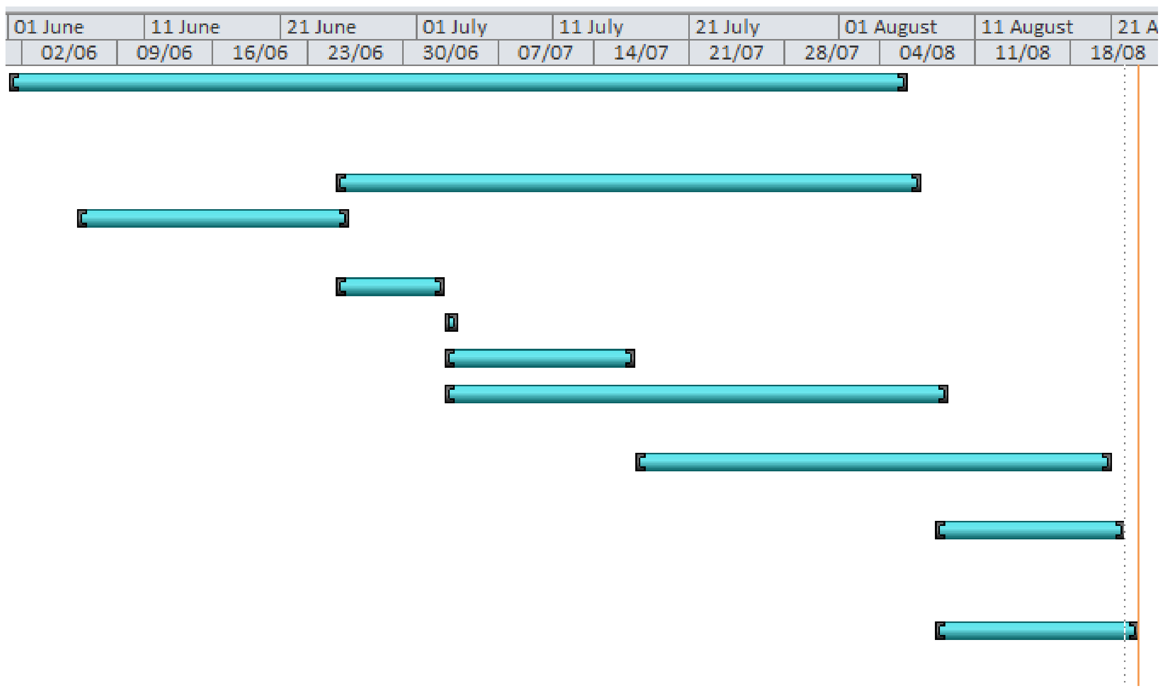
  Comprehensive detailing on the implementation of these functionalities is discussed in Chapter six.
- **System Evaluation and Testing**: The functionalities developed in the previous stage were then interfaced together into a mobile robot system. This overall setup was then tested to ensure adequate and consistent interaction between its comprising functionalities.
- **System maintenance and Future works**: The way the system was designed and implemented was such that it allows for easy maintenance and future implementations to be added to further needed research in the projects problem domain.

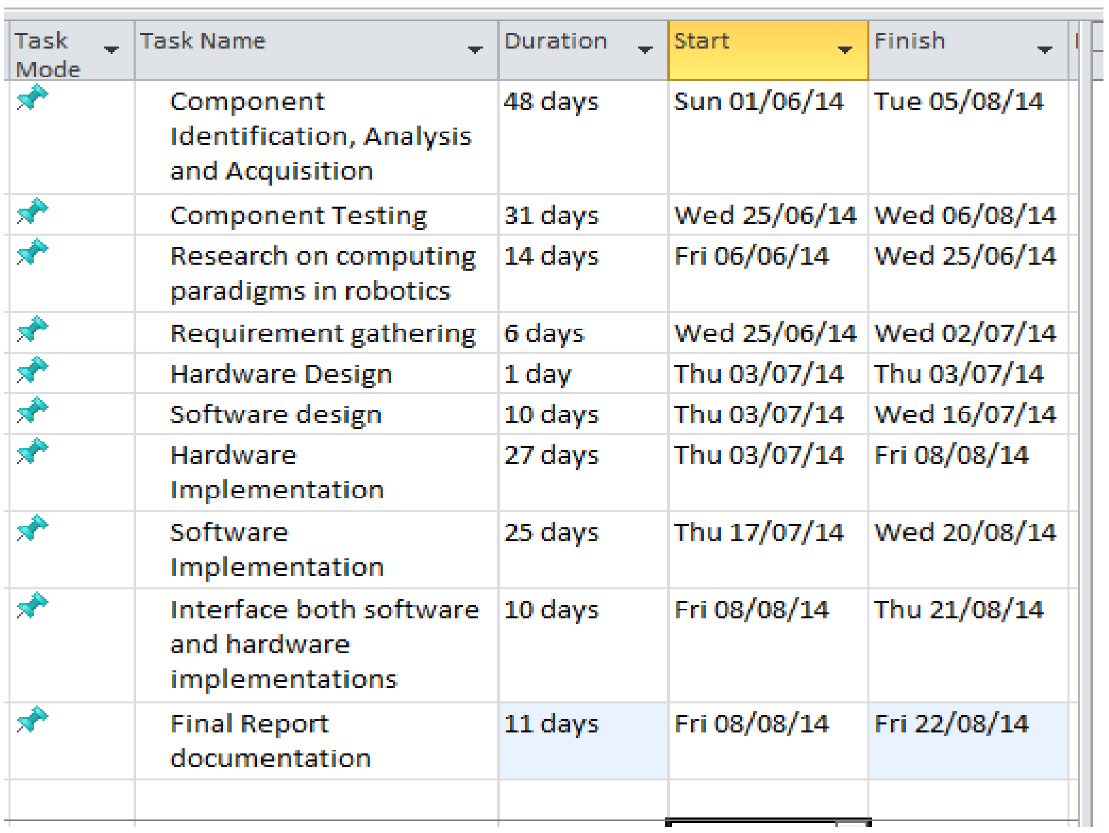
3.3 Component Identification and Acquisition

Since this project has electronics as well as computing as components of its overall make up, hardware peripherals needed for the physical implementation of the project needed to be identified, approved and acquired for the project to move forward (Analysis of these peripherals are discussed in subchapter 5.1). Some of these peripherals were already provided by the school, while others had to be acquired before and during the development cycle.

3.4 Project Schedule

The Gantt chart below features the time line of this project





Task Mode	Task Name	Duration	Start	Finish
📌	Component Identification, Analysis and Acquisition	48 days	Sun 01/06/14	Tue 05/08/14
📌	Component Testing	31 days	Wed 25/06/14	Wed 06/08/14
📌	Research on computing paradigms in robotics	14 days	Fri 06/06/14	Wed 25/06/14
📌	Requirement gathering	6 days	Wed 25/06/14	Wed 02/07/14
📌	Hardware Design	1 day	Thu 03/07/14	Thu 03/07/14
📌	Software design	10 days	Thu 03/07/14	Wed 16/07/14
📌	Hardware Implementation	27 days	Thu 03/07/14	Fri 08/08/14
📌	Software Implementation	25 days	Thu 17/07/14	Wed 20/08/14
📌	Interface both software and hardware implementations	10 days	Fri 08/08/14	Thu 21/08/14
📌	Final Report documentation	11 days	Fri 08/08/14	Fri 22/08/14

Figure 8: Project Gantt chart and Schedule

3.5 Risk and Ethics Management

There were no ethical concerns identified during project conception and approval, due to the fact that this project does not involve collection and use of user data or impact users or third parties.

Regardless of no ethical concerns involved for this project, as the project requires interaction with electrical and electronic components, precautionary methods had to be taken to ensure developer safety. These are listed below.

Precautionary Measures:

- When charging the battery pack, it must not be left unattended nor should it be done on a flammable surface.
- Power supply to the electrical components were from a constant supply source (DC supply) and the required voltage and current supply needed for each component was administered respectively to avoid overload and damage.
- Short circuit connections also need to be avoided, to ensure overloading of components and circuitries do not occur.

4 System Requirements Specification

This chapter describes the requirements specifications for designing an autonomous mobile robot developed for this project; it highlights the requirement that will affect the hardware functionality design as well as the software architecture to develop for this robotics project. For the sake of this chapter the mobile robot system will just be referred to as "The system" from now on.

4.1 Functional Requirements

Specification number	Requirement	Impact on System Design
FR001	The system must have full range of motion within its DOF constraints	High
FR002	The system circuitry must allow for relative low power supplication	High
FR003	The system must have 180 degree range motion for webcam movement	High
FR004	The system must be able to capture real time images of its environment	High
FR005	The system must be able to stream image capture feed from an attached webcam to a controlling computer	High
FR006	The system must be able to create a private wireless communication network between the robot and the controlling PC	High
FR007	The system must be able read interfaced sensor values in a readable format.	High
FR008	The system must have a user interface that allows the user to view camera feed from the robot	High
FR009	The system must be able to autonomously track a given target in real time.	High
FR010	The system should allow the user to choose the goal to be tracked	Medium

FR011	The system must allow wireless communication between the controlling pc and receiving device	High
FR012	The system's interface should allow the user to choose which serial port from a PC to use for communication	Medium
FR013	The system's interface must allow the user to choose the Baud rate for serial communication	Medium
FR014	The system's interface should allow the user to set the speed of movement of the robot	Low
FR015	The system must have an autonomous mode of operation	Medium
FR016	The system must have a manual mode of operation	Medium
FR017	The system must use intelligent motion planning algorithms for obstacle navigation during autonomous mode	High
FR018	The system must allow the user complete control of its robot subsystem's overall motion during its manual mode	High
FR019	The system must allow the user to switch between operation modes	Medium
FR020	The System must allow the user to remotely control the movement of mobile robot part system	High
FR021	The system must operate on an efficient low-power supply system.	High

Table 1 Functional Requirements of the system

4.2 User Interface Requirements

Specification Number	Requirements	Impact on System Design
IR001	The GUI must be easy to use	High
IR002	The GUI should be reasonably responsive	Medium
IR003	The different GUI icons labels must be presentable	Low
IR004	Words should be clearly visible on the GUI	Low

Table 2: User Interface requirement of the system

4.3 PC System Requirements

Specification Number	Requirement	Impact on System Design
SR001	The controlling PC must have functioning USB ports for connecting Serial Communication devices	High
SR002	The controlling PC must have OpenCV 2.4.8 installed for image processing	High
SR003	The controlling PC must have WIFI connectivity	High
SR004	The controlling PC must have a fast enough processing unit that will allow for quick image processing functionality while generating a GUI interface among other functionalities.	Medium

Table 3: PC system requirement to run the system GUI

5 System Design

5.1 Hardware Design

As mentioned earlier, one of the objectives of this project is to develop a physical mobile robot system; this includes designing a hardware solution whose implementation can be used to embody software algorithms (motion planning, target tracking etc.) devised for the project system. To develop any mobile robotic system physically, control systems, sensors as well as actuators among other peripherals (depending on the objective and application of the system being developed) will need to be interfaced together in other to achieve system functionalities. As shown in the figure below the mobile robot system developed for this project has a three layered frame each layer houses hardware for key functionality groups – first layer holds hardware for voltage regulation as well as hardware for main motor control, the second layer houses hardware for system control (microprocessor), the third layer is where all the sensors (ranger finder and webcam) and communication router.

The main motor configuration of the system is that of two independently powered differential motors (though they are both electronically controlled together) with conventional wheels that allow for only two DOF (DOF_{System} and DOF_{Frame}), and has two corner steering (just like a normal car). It also has a mechanical steering wheel at the centre so that the rotation axis of the system is located at the centre of that wheel. The electrical sub-system architecture is as shown in figure 8. The components that make up the physical controller, obstacle sensors, actuators, computer vision etc., and their properties are detailed in the Table 4 below.

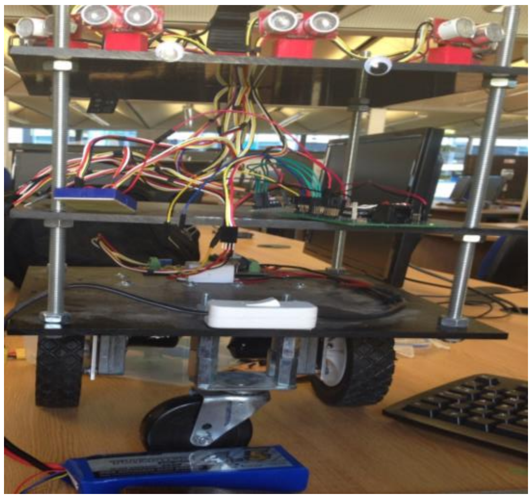
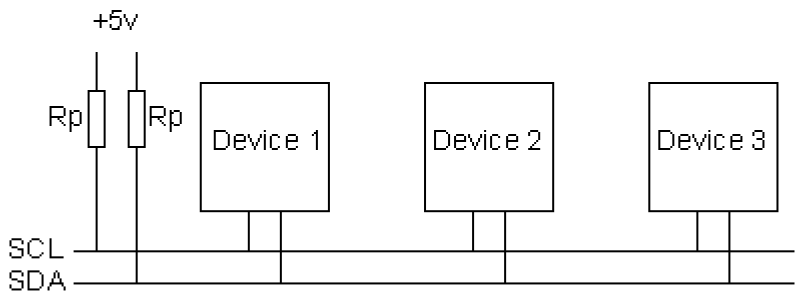


Figure 9: Mobile Robot System

The system generates various voltage levels (12V, 5V and 3.3V) as needed for each component, with the differential DC motors being the largest user of power. The sensor suit comprises of ultrasonic range finders and a webcam with wireless LAN connection. It also has a radio frequency router for wireless communication with a computer. Both the motor driver and the set of ultrasonic range finders use I^2C protocols in communicating with a microprocessor, using controller's I^2C bus (The MD22 also has four analogue control setting using Pulse Width Modulation to control the DC motors). Pull up resistors within the range 1.8K – 10K ohms are connected between controller and the devices in order to drive the Serial Data (SDA) and Serial Clock (SCL) signals, else a near null voltage signal is driven. This protocol uses a master-slave communication sequence, for this project the hardware controller (PIC) is the master device while the motor driver and the sensor sets act as slave devices.



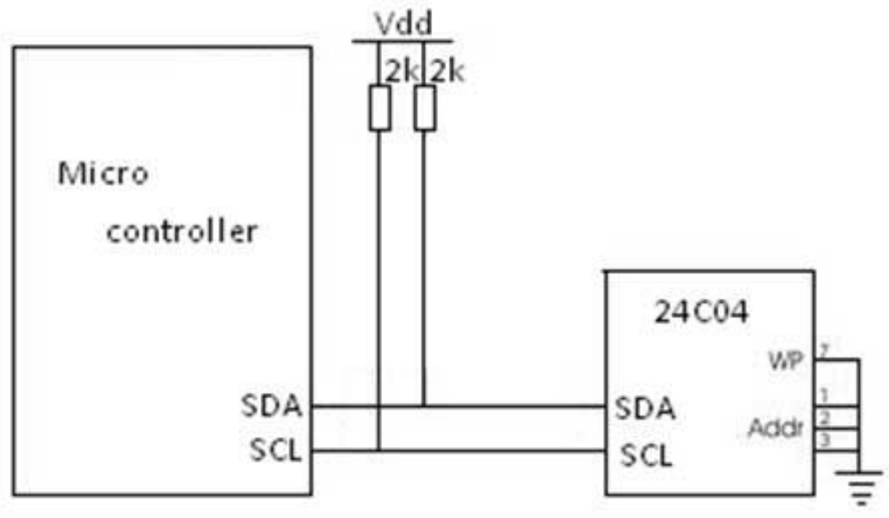


Figure 10: Interfacing I^2C devices with a micro controller (Robot Electronics 2014) (PanTech Solutions 2014)

The physical robot uses a PIC18f4520 chip as its hardware controller, the chip is interfaced with a multi-purpose design development board with high extensibility, for varying I/O port connections including Analogue to digital conversion I/Os (ADC), it is programmed via a

Pickit3 programmer and debugger mini ICD, using MPLAB IDE software. The PCB schematic for the development board is shown in the figure below.

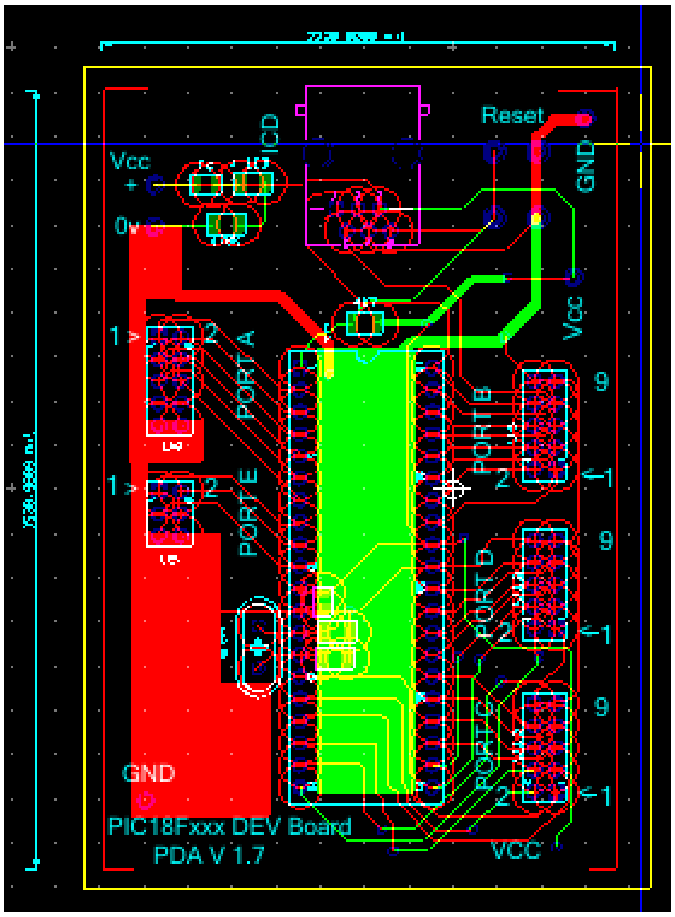


Figure 11: PCB schematic of PIC multipurpose development board with port extensibility

5.1.1 Architecture:

Before designing either of the software or hardware aspects of this project, an overall architecture was developed for the entire system interfacing; this architecture gives a high level abstraction of the control system as well as how the various elements in the system interact with each other. Figure 11 is a very high level abstraction of the overall system and element communication (⟷), while figure 12, is high level abstraction of the hardware architecture for the mobile robotic system.

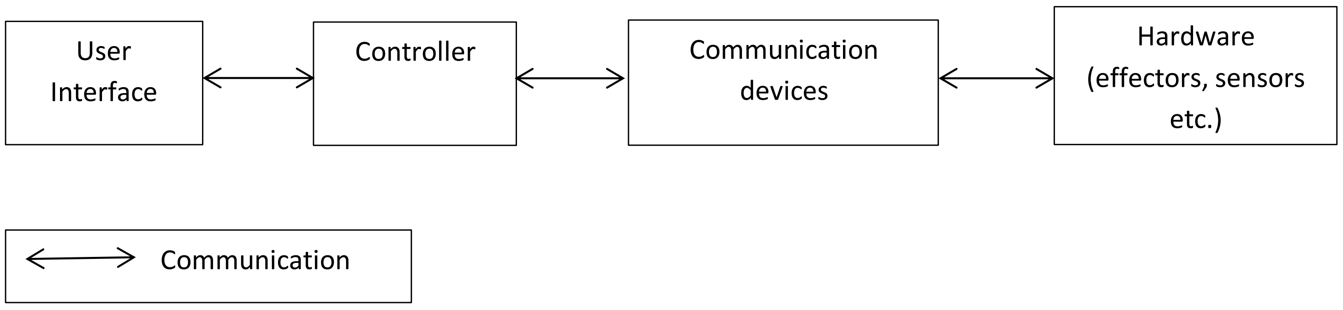


Figure 12: Highest level Controller (system architecture) abstraction

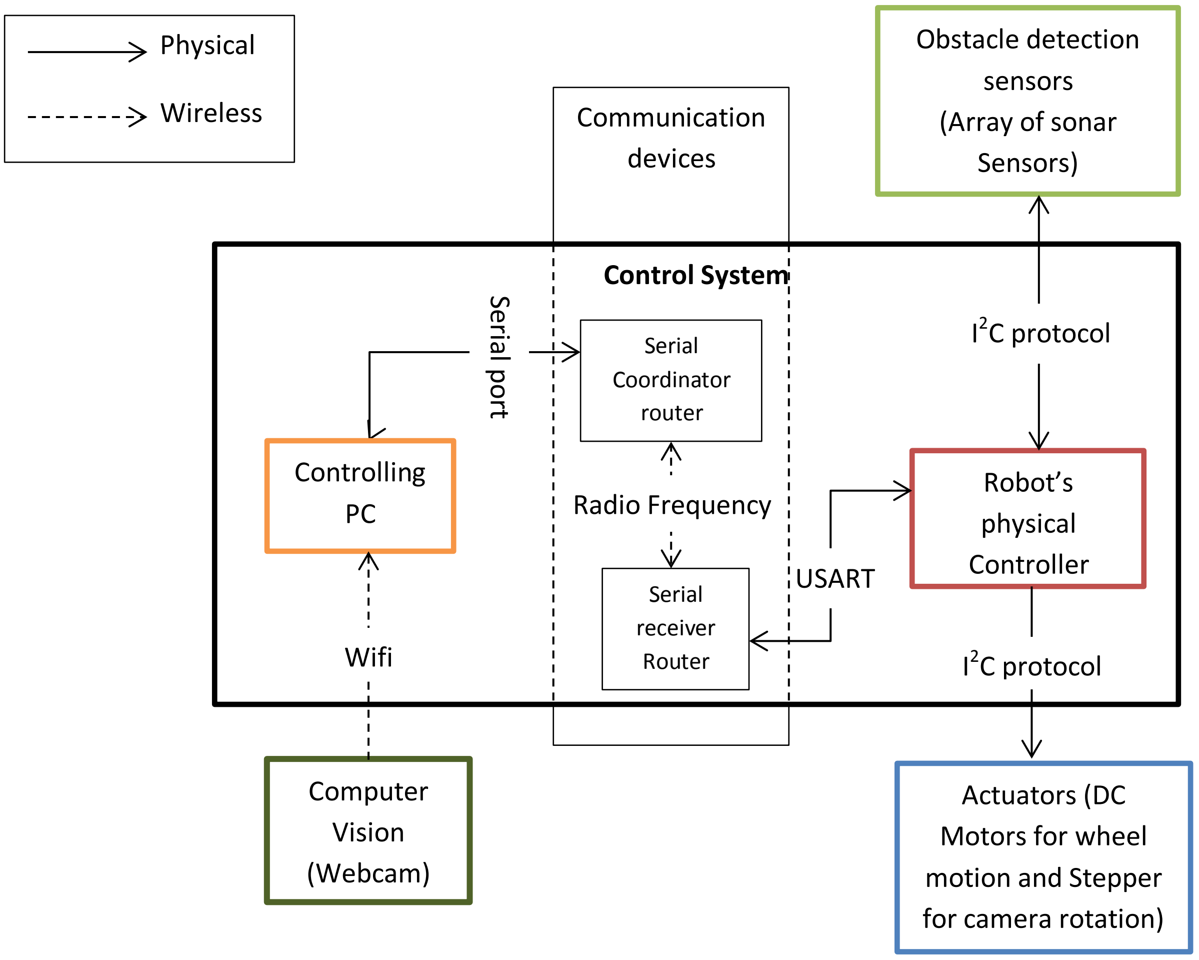

Physical
Wireless
Communication devices
Obstacle detection sensors (Array of sonar Sensors)
Control System
Serial port
Serial Coordinator router
I²C protocol
Controlling PC
Radio Frequency
Robot's physical Controller
USART
Serial receiver Router
Wifi
I²C protocol
Computer Vision (Webcam)
Actuators (DC Motors for wheel motion and Stepper for camera rotation)

Figure 13: Overall Physical architecture of mobile robotic system

5.1.2 Component Listing and Analysis

Item	**Model**	**Comment**		
CPU	PIC18F4520	(Microchip 2014)		
			Program Memory Type	Flash
			Program Memory (KB)	32
			CPU Speed (MIPS)	10
			RAM Bytes	1,536
			Data EEPROM (bytes)	256
			Digital Communication Peripherals	1-UART, 1-A/E/USART, 1-SPI, 1-I2C1-MSSP(SPI/I2C)
			Capture/Compare/PWM Peripherals	1 CCP, 1 ECCP
			Timers	1 x 8-bit, 3 x 16-bit
			ADC	13 ch, 10-bit
			Comparators	2
			Temperature Range (C)	-40 to 125
			Operating Voltage Range (V)	2 to 5.5
			Pin Count	40
Camera	Ai-Ball wireless webcam	(Trek 2010)		
			Camera Specifications	VGA 640x480, QVGA 320x240, QQVGA 160x120, up to 30fps F2.8, View Angle: 60 degree, Focusing: 20cm to Infinity
			Wireless Interface	IEEE 802.11b/g 2.4GHz ISM Band
			Wireless Security	WEP 64/128, WPA, WPA2
			Wireless Range	Infrastructure: 20m (Typical), Adhoc: 7.5m (Typical)
			Wireless performance*	Infrastructure: 54Mbps, max. connection rate, Adhoc: 11Mbps, max. connection rate,
			Antenna	Single, internal
			Dimension / Weight	30mm(Diameter) x 35mm(L), 100g
			Power Supply / Consumption	Battery operated Voltage: 3.0V, Power: 750mAH (CR2), Current consumption: 320mA (typical); 350mA (maximum)
Router	Xbee Pro Series 2	(Digi 2012)		
			Indoor/Urban Range	Up to 300 ft. (90 m), up to 200 ft (60 m) international variant
			Transmit Power Output	50mW (+17 dBm), 10mW (+10 dBm) for International variant
			RF Data Rate	250,000 bps
			Supply Voltage	3.0 - 3.4 V

		Operating Frequency Band	ISM 2.4 GHz
		Antenna Options	Integrated Whip Antenna, Embedded. PCB Antenna, RPSMA or U.FL Connector
		Supported Network Topologies	Point-to-point, point-to-multipoint, peer-to-peer and mesh
		Number of Channels	14 Direct Sequence Channels
		Channels	11 to 24
		Addressing Options	PAN ID and Addresses, Cluster IDs and Endpoints (optional)
Motor Controls	MD22	(Robot electronics 2014)	
		Control	Drives two motors with independent control
		Steering features	motors can be commanded to turn by I2C register or input (Analogue + Servo)
		Operation	5 modes of operation including steering
		DC- operation	Uses high current MOSFETs, making a very robust module
Motors	DC motors	12V power supply	
Rotor control	Darlington Driver	Eight Darlington transistors 5-12V power supply	
Chip Programmer	Pickit 3	(Microchip 2014)	
		PC interfacing	USB (Full speed 12 Mbits/s interface to host PC)
		Execution	Real-time execution
		IDE	MPLAB IDE compatible
		Hardware casing	Totally enclosed
		Status Indicator	Diagnostic LEDs (power, busy, error)
		Operation	Read/write program and data memory of microcontroller
		Voltage regulation	Supports low voltage to 2.0 volts (2.0v to 6.0v range), Built-in over-voltage/short circuit monitor
Camera rotator	Stepper motors		
		Steps	32 step, 11.25° per step
		Power supply	5-12V
Distance ranging sensors	SRF08	(Robot Electronics 2012)	

		Input Voltage	5V
		Current	15mA typ. - 3mA
		Range	3cm - 6m
		Frequency	40kHz
		Maximum analogue gain	Variable 94 to 1025 in 32 steps
		Connection	Standard I2C Bus.
		Light sensor	Front Facing light sensor.
		Timing	Fully timed echo, freeing host controller of task.
		Echo	Multiple echo - keeps looking after first echo.
		Units	Range reported in µS, cm or inches.
		Size	43mm x 20mm x 17mm
Batteries	Turnigy 4.5	LiPo 11.1V 1-1.4A, 3 cell	

Table 4: Component listing and their properties.

5.1.3 Overall Hardware System Design

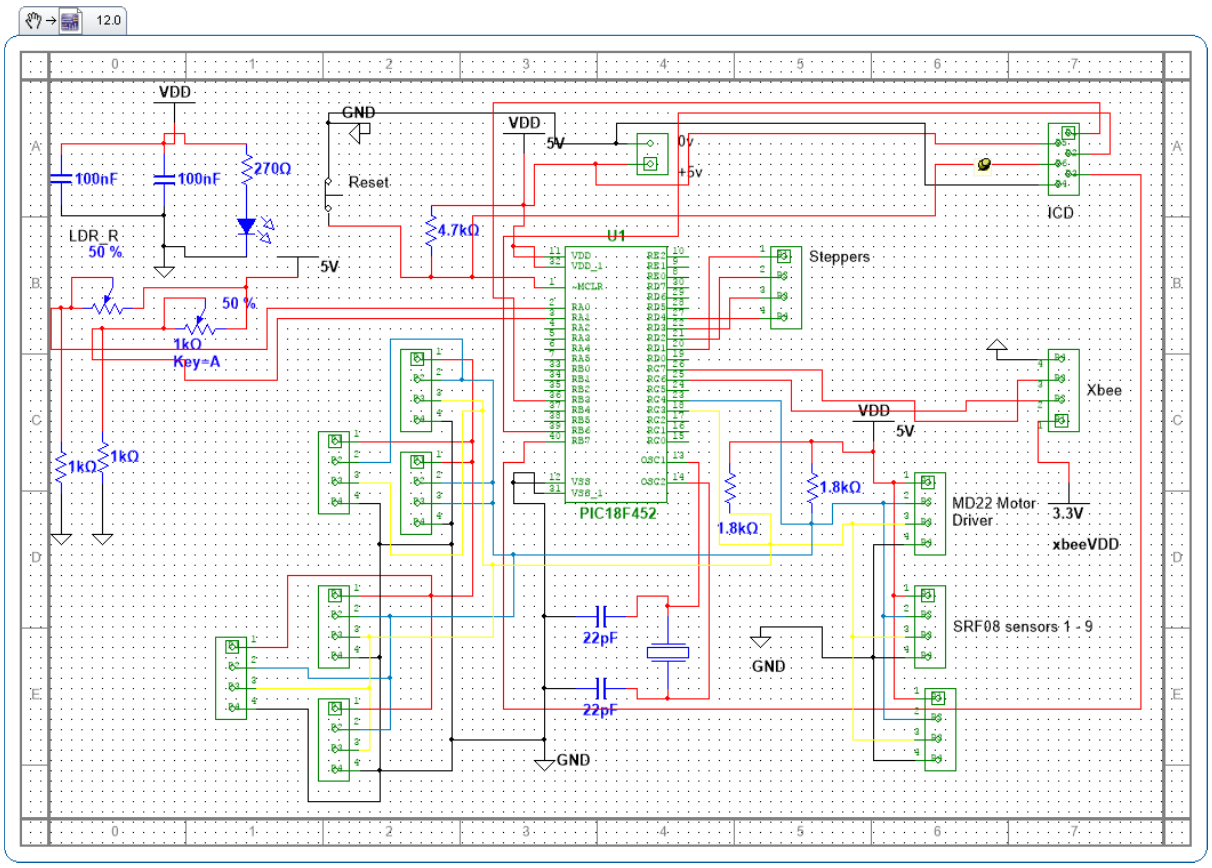


Figure 14: Electronic schematic for hardware component interfacing for the system

5.2 Software Design

5.2.1 Architecture

In order to make design decisions for developing the base software architecture for this project's system, prioritization of the parts of the captured requirements (see Chapter 4) that have some software orientation is needed, these requirements will define the core software architecture.

Specification number	Requirement	Impact on System Design
FR008	The system must have a user interface that allows the user to view camera feed from the robot	High
FR009	The system must be able to track a given target in real time from a camera feed	High
FR010	The system should allow the user to choose the goal to be tracked	Medium
FR011	The system must allow wireless communication between the controlling pc and receiving device	High
FR012	The system's interface should allow the user to choose which serial port from a PC to use for communication	Medium
FR013	The system's interface must allow the user to choose the Baud rate for serial communication	Medium

Table 5: Prioritized Requirement for the system software development

As mention previously, since this project has dome electronic components, the constraint for developing language will be C and C++, as these languages are best to for accessing functionalities from hardware components. The software architectural drivers chosen to design the system's software aspect is that of an object oriented architectural style in combination with Model-View-Controller design pattern, these drivers are best suited for developing the system because other styles like Client and server, will add unnecessary complexity to the system as a whole (This architecture was previously attempted for this project by Conrad de Kerckhove [2014], a critical error occurred in connecting with server and locked down the entire system).

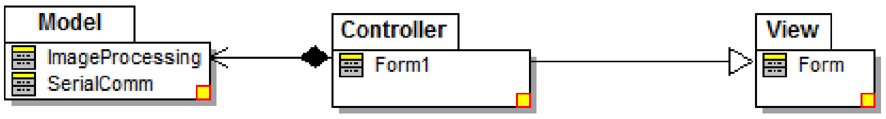


Figure 15: MVC design pattern for the Interface

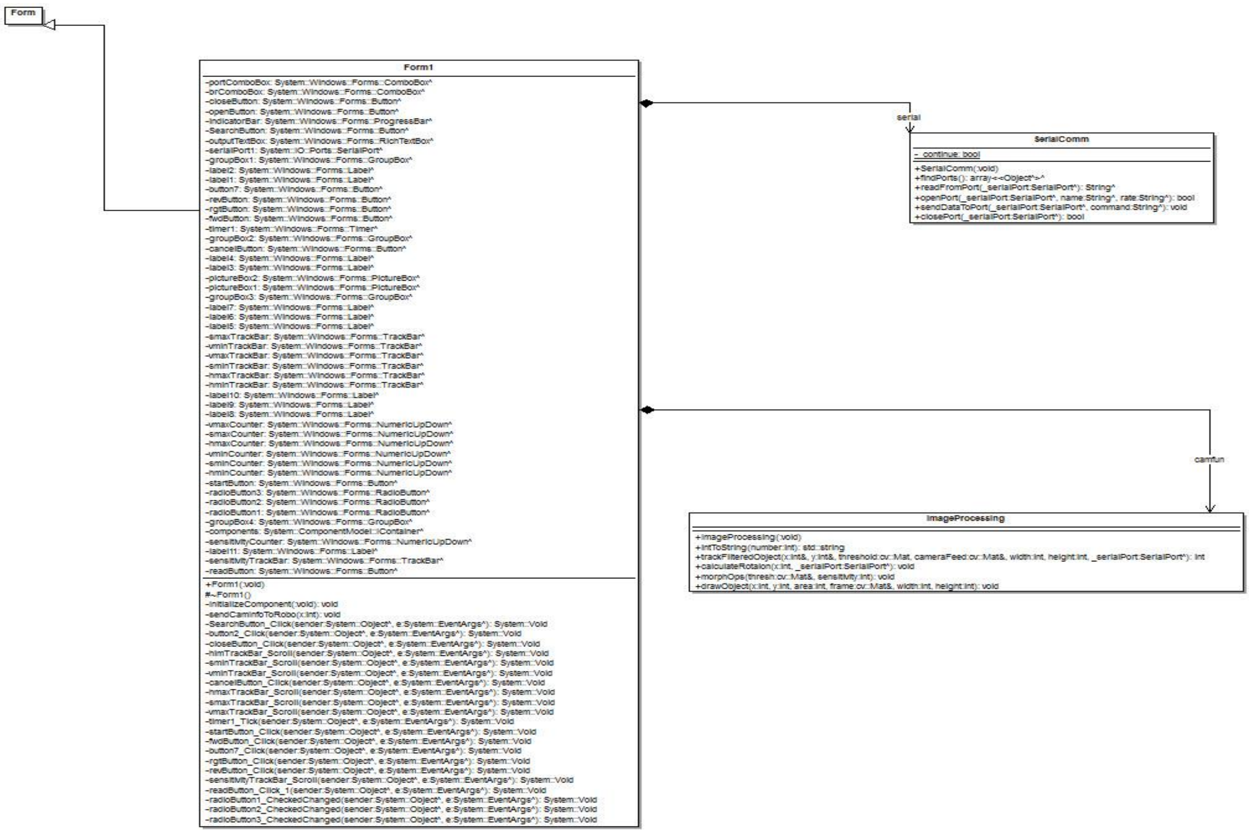

Form
Form1
-portComboBox: System::Windows::Forms::ComboBox^
-brComboBox: System::Windows::Forms::ComboBox^
-closeButton: System::Windows::Forms::Button^
-openButton: System::Windows::Forms::Button^
-indicatorBar: System::Windows::Forms::ProgressBar^
-SearchButton: System::Windows::Forms::Button^
-outputTextBox: System::Windows::Forms::RichTextBox^
-serialPort1: System::IO::Ports::SerialPort^
-groupBox1: System::Windows::Forms::GroupBox^
-label2: System::Windows::Forms::Label^
-label1: System::Windows::Forms::Label^
-button7: System::Windows::Forms::Button^
-revButton: System::Windows::Forms::Button^
-rgtButton: System::Windows::Forms::Button^
-fwdButton: System::Windows::Forms::Button^
-timer1: System::Windows::Forms::Timer^
-groupBox2: System::Windows::Forms::GroupBox^
-cancelButton: System::Windows::Forms::Button^
-label4: System::Windows::Forms::Label^
-label3: System::Windows::Forms::Label^
-pictureBox2: System::Windows::Forms::PictureBox^
-pictureBox1: System::Windows::Forms::PictureBox^
-groupBox3: System::Windows::Forms::GroupBox^
-label7: System::Windows::Forms::Label^
-label6: System::Windows::Forms::Label^
-label5: System::Windows::Forms::Label^
-smaxTrackBar: System::Windows::Forms::TrackBar^
-vminTrackBar: System::Windows::Forms::TrackBar^
-vmaxTrackBar: System::Windows::Forms::TrackBar^
-sminTrackBar: System::Windows::Forms::TrackBar^
-hmaxTrackBar: System::Windows::Forms::TrackBar^
-hminTrackBar: System::Windows::Forms::TrackBar^
-label10: System::Windows::Forms::Label^
-label9: System::Windows::Forms::Label^
-label8: System::Windows::Forms::Label^
-vmaxCounter: System::Windows::Forms::NumericUpDown^
-smaxCounter: System::Windows::Forms::NumericUpDown^
-hmaxCounter: System::Windows::Forms::NumericUpDown^
-vminCounter: System::Windows::Forms::NumericUpDown^
-sminCounter: System::Windows::Forms::NumericUpDown^
-hminCounter: System::Windows::Forms::NumericUpDown^
-startButton: System::Windows::Forms::Button^
-radioButton3: System::Windows::Forms::RadioButton^
-radioButton2: System::Windows::Forms::RadioButton^
-radioButton1: System::Windows::Forms::RadioButton^
-groupBox4: System::Windows::Forms::GroupBox^
-components: System::ComponentModel::IContainer^
-sensitivityCounter: System::Windows::Forms::NumericUpDown^
-label11: System::Windows::Forms::Label^
-sensitivityTrackBar: System::Windows::Forms::TrackBar^
-readButton: System::Windows::Forms::Button^
+Form1(:void)
#~Form1()
-InitializeComponent(:void): void
-sendCamInfoToRobo(x:int): void
-SearchButton_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-button2_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-closeButton_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-hlmTrackBar_Scroll(sender:System::Object^, e:System::EventArgs^): System::Void
-sminTrackBar_Scroll(sender:System::Object^, e:System::EventArgs^): System::Void
-vminTrackBar_Scroll(sender:System::Object^, e:System::EventArgs^): System::Void
-cancelButton_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-hmaxTrackBar_Scroll(sender:System::Object^, e:System::EventArgs^): System::Void
-smaxTrackBar_Scroll(sender:System::Object^, e:System::EventArgs^): System::Void
-vmaxTrackBar_Scroll(sender:System::Object^, e:System::EventArgs^): System::Void
-timer1_Tick(sender:System::Object^, e:System::EventArgs^): System::Void
-startButton_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-fwdButton_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-button7_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-rgtButton_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-revButton_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-sensitivityTrackBar_Scroll(sender:System::Object^, e:System::EventArgs^): System::Void
-readButton_Click_1(sender:System::Object^, e:System::EventArgs^): System::Void
-radioButton1_CheckedChanged(sender:System::Object^, e:System::EventArgs^): System::Void
-radioButton2_CheckedChanged(sender:System::Object^, e:System::EventArgs^): System::Void
-radioButton3_CheckedChanged(sender:System::Object^, e:System::EventArgs^): System::Void
serial
SerialComm
- continue: bool
+SerialComm(:void)
+findPorts(): array<<Object^>^
+readFromPort(_serialPort:SerialPort^): String^
+openPort(_serialPort:SerialPort^, name:String^, rate:String^): bool
+sendDataToPort(_serialPort:SerialPort^, command:String^): void
+closePort(_serialPort:SerialPort^): bool
camfun
ImageProcessing
+ImageProcessing(:void)
+intToString(number:int): std::string
+trackFilteredObject(x:int&, y:int&, threshold:cv::Mat, cameraFeed:cv::Mat&, width:int, height:int, _serialPort:SerialPort^): int
+calculateRotaion(x:int, _serialPort:SerialPort^): void
+morphOps(thresh:cv::Mat&, sensitivity:int): void
+drawObject(x:int, y:int, area:int, frame:cv::Mat&, width:int, height:int): void

Figure 16: The extracted interface design class diagram

The model comprises of the ImageProcessing class and the SerialComm class which serve as the back-end of the program where image processing and tracking functions as well as serial port communications functions are defined, they are then called using their initialized objects in the controller class.

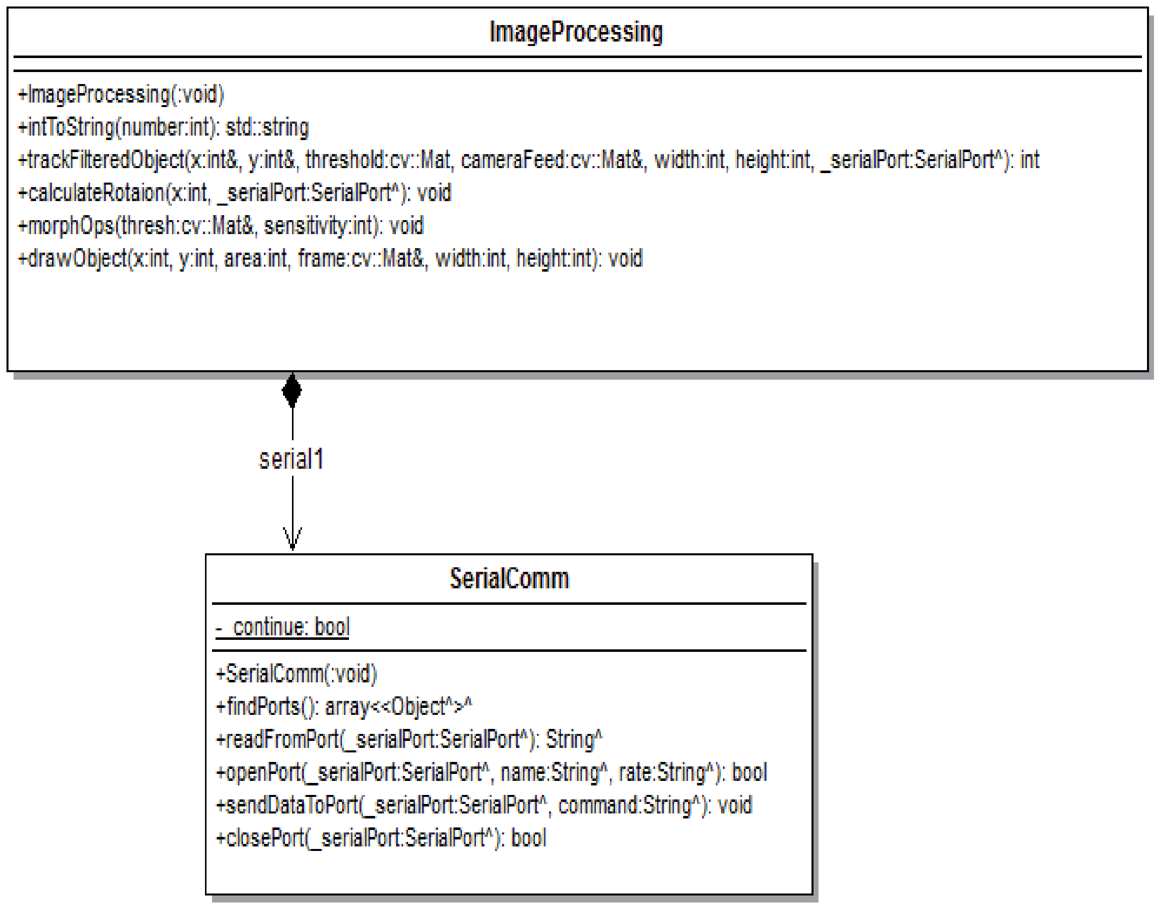
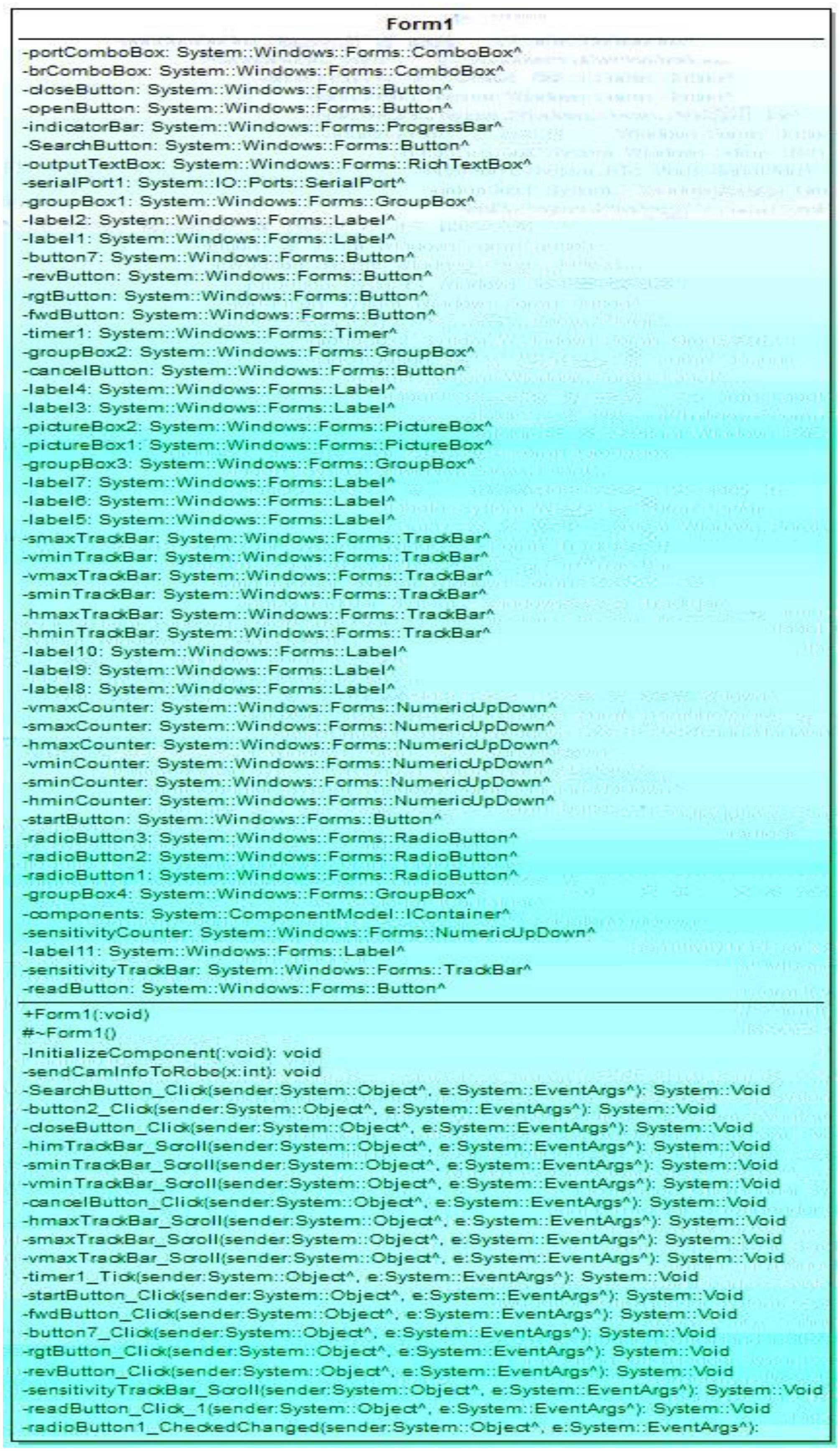

ImageProcessing
+ImageProcessing(:void)
+intToString(number:int): std::string
+trackFilteredObject(x:int&, y:int&, threshold:cv::Mat, cameraFeed:cv::Mat&, width:int, height:int, _serialPort:SerialPort^): int
+calculateRotaion(x:int, _serialPort:SerialPort^): void
+morphOps(thresh:cv::Mat&, sensitivity:int): void
+drawObject(x:int, y:int, area:int, frame:cv::Mat&, width:int, height:int): void
serial1
SerialComm
- continue: bool
+SerialComm(:void)
+findPorts(): array<<Object^>^
+readFromPort(_serialPort:SerialPort^): String^
+openPort(_serialPort:SerialPort^, name:String^, rate:String^): bool
+sendDataToPort(_serialPort:SerialPort^, command:String^): void
+closePort(_serialPort:SerialPort^): bool

Figure 17: Model class diagram showing relationship between classes

The controller is the Form class that implements the Form interface, in this class; *button* events, *trackbar* event, and all other events are handled.



```
Form1
-portComboBox: System::Windows::Forms::ComboBox^
-brComboBox: System::Windows::Forms::ComboBox^
-closeButton: System::Windows::Forms::Button^
-openButton: System::Windows::Forms::Button^
-indicatorBar: System::Windows::Forms::ProgressBar^
-SearchButton: System::Windows::Forms::Button^
-outputTextBox: System::Windows::Forms::RichTextBox^
-serialPort1: System::IO::Ports::SerialPort^
-groupBox1: System::Windows::Forms::GroupBox^
-label2: System::Windows::Forms::Label^
-label1: System::Windows::Forms::Label^
-button7: System::Windows::Forms::Button^
-revButton: System::Windows::Forms::Button^
-rgtButton: System::Windows::Forms::Button^
-fwdButton: System::Windows::Forms::Button^
-timer1: System::Windows::Forms::Timer^
-groupBox2: System::Windows::Forms::GroupBox^
-cancelButton: System::Windows::Forms::Button^
-label4: System::Windows::Forms::Label^
-label3: System::Windows::Forms::Label^
-pictureBox2: System::Windows::Forms::PictureBox^
-pictureBox1: System::Windows::Forms::PictureBox^
-groupBox3: System::Windows::Forms::GroupBox^
-label7: System::Windows::Forms::Label^
-label6: System::Windows::Forms::Label^
-label5: System::Windows::Forms::Label^
-smaxTrackBar: System::Windows::Forms::TrackBar^
-vminTrackBar: System::Windows::Forms::TrackBar^
-vmaxTrackBar: System::Windows::Forms::TrackBar^
-sminTrackBar: System::Windows::Forms::TrackBar^
-hmaxTrackBar: System::Windows::Forms::TrackBar^
-hminTrackBar: System::Windows::Forms::TrackBar^
-label10: System::Windows::Forms::Label^
-label9: System::Windows::Forms::Label^
-label8: System::Windows::Forms::Label^
-vmaxCounter: System::Windows::Forms::NumericUpDown^
-smaxCounter: System::Windows::Forms::NumericUpDown^
-hmaxCounter: System::Windows::Forms::NumericUpDown^
-vminCounter: System::Windows::Forms::NumericUpDown^
-sminCounter: System::Windows::Forms::NumericUpDown^
-hminCounter: System::Windows::Forms::NumericUpDown^
-startButton: System::Windows::Forms::Button^
-radioButton3: System::Windows::Forms::RadioButton^
-radioButton2: System::Windows::Forms::RadioButton^
-radioButton1: System::Windows::Forms::RadioButton^
-groupBox4: System::Windows::Forms::GroupBox^
-components: System::ComponentModel::IContainer^
-sensitivityCounter: System::Windows::Forms::NumericUpDown^
-label11: System::Windows::Forms::Label^
-sensitivityTrackBar: System::Windows::Forms::TrackBar^
-readButton: System::Windows::Forms::Button^
+Form1(:void)
#~Form1()
-InitializeComponent(:void): void
-sendCamInfoToRobo(x:int): void
-SearchButton_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-button2_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-closeButton_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-himTrackBar_Scroll(sender:System::Object^, e:System::EventArgs^): System::Void
-sminTrackBar_Scroll(sender:System::Object^, e:System::EventArgs^): System::Void
-vminTrackBar_Scroll(sender:System::Object^, e:System::EventArgs^): System::Void
-cancelButton_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-hmaxTrackBar_Scroll(sender:System::Object^, e:System::EventArgs^): System::Void
-smaxTrackBar_Scroll(sender:System::Object^, e:System::EventArgs^): System::Void
-vmaxTrackBar_Scroll(sender:System::Object^, e:System::EventArgs^): System::Void
-timer1_Tick(sender:System::Object^, e:System::EventArgs^): System::Void
-startButton_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-fwdButton_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-button7_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-rgtButton_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-revButton_Click(sender:System::Object^, e:System::EventArgs^): System::Void
-sensitivityTrackBar_Scroll(sender:System::Object^, e:System::EventArgs^): System::Void
-readButton_Click_1(sender:System::Object^, e:System::EventArgs^): System::Void
-radioButton1_CheckedChanged(sender:System::Object^, e:System::EventArgs^):
```

Figure 18: Form class (controller)

For designing the software program for hardware controller operations that control the other hardware components, no real architecture was used, but functions were separated accordingly for debugging, maintenance and readability purposes.

5.2.2 Component Listing

Item	Model	Comment
Robot Controller code Compiler	MPLAB IDE v8.90 - MCC18 compiler	32-bit Windows GUI interface C language support in form of MCC18, Developed by Microchip for embedded programming And also assembly language support (MPASM)
Radio module programmer	X-CTU	Developed by Digi
Interface	MS Visual Studio 2010	Windows free Form projects
Image Processing	OpenCV 2.4.8	C++ libraries developed by IBM for image processing and research in computer vision Ported to other languages, such as, JavaCV for java, and also Opencv-Python, for python

Table 6: Software component list

5.2.3 Target/Goal Tracking

The design for enabling target tracking uses robot vision from an attached webcam. Images streamed in real-time from the webcam are captured, and then processed using openCV functions – Including filtering the HSV (Hue, Saturation, values – Blue-Green-Red [BGR] shades i.e. hue or tint) values in an image frame, morphing a threshold image based on the changing HSV values, drawing shapes and lines to identify the target currently being tracked etc.. Manipulation of the image processing part of the interface was developed to be user friendly, such that it allows the user to choose which target within the vision field of the robot's webcam. After the user has acquired a target in the frame, the (x,y) pixelated position of the centre of the target is tracked. As the is no tilt motion integrated into the webcam motion, but only horizontal translations, only the pixelated x-coordinate of the target is used in sending camera rotation control.

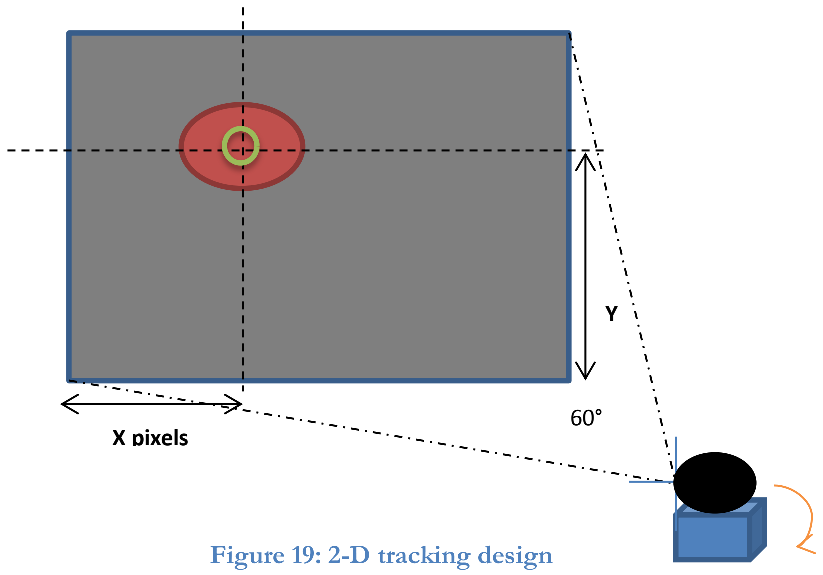


Figure 19: 2-D tracking design

Pixelated coordinates of the target in a captured frame are used because; there are hardware limitations - robot vision depth perception (for an unknown goal size) cannot be evaluated using a single camera only; but using triangulation techniques which use trigonometric methods to calculate distance and angle when locating a target in a space constraint (Jansen 2010). This can either be done using stereoscopic vision (using to aligned cameras to view and then compare frames), or using a visible IR emitter device attached to the camera to act as a distance sensor for the camera, this is because light travels in a straight line, making it easy to measure the distance of a target when the IR ray is reflection is captured on the camera.-

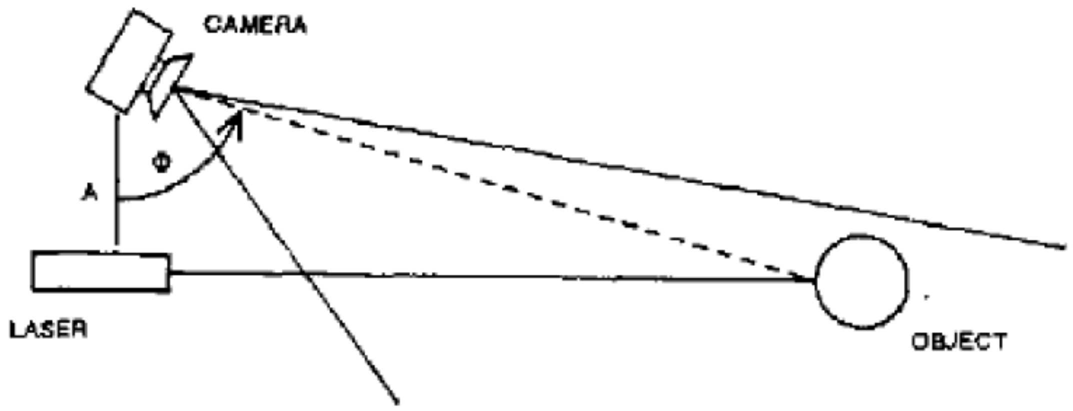


Figure 20: Depth perception by Triangulation using IR emitter with camera (Jensen 2010)

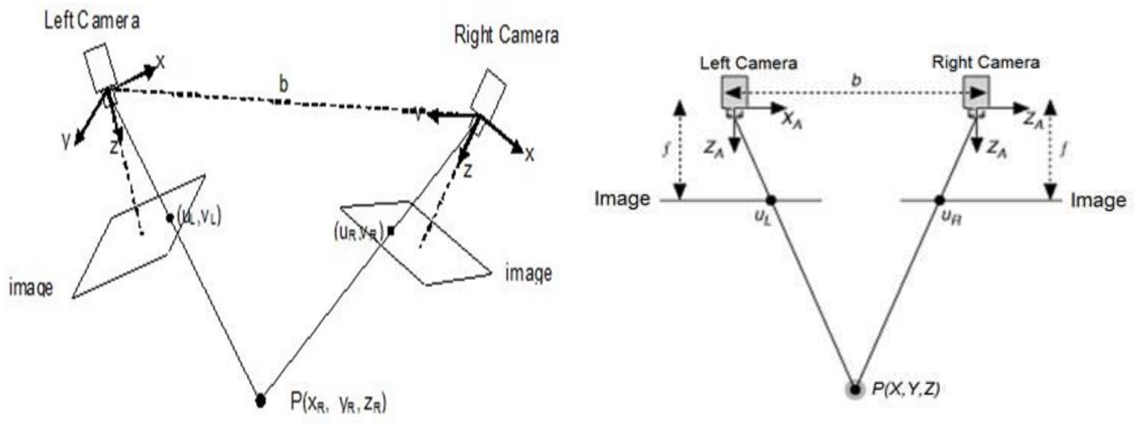


Figure 21 Stereo vision systems typically used in robotics (National Instrument 2014)

Also the lack of odometry components such as gyroscopes and accelerometer limit real world approximation of the targets position relative to that of the robot in a space constraint.

5.2.4 Motion Planning

Motion during Manual Operation: The design for manual motion was very simple; using the interface, the user should be able to control the direction to the robot moves, by clicking on the button for the corresponding direction. If a directional button is clicked, a button click event will be called to handle a function that sends a directional command to be transmitted via the connect RF module to the receiver module mounted on the robot, figures (…) the activity diagram actions performed by the hardware controller software program.

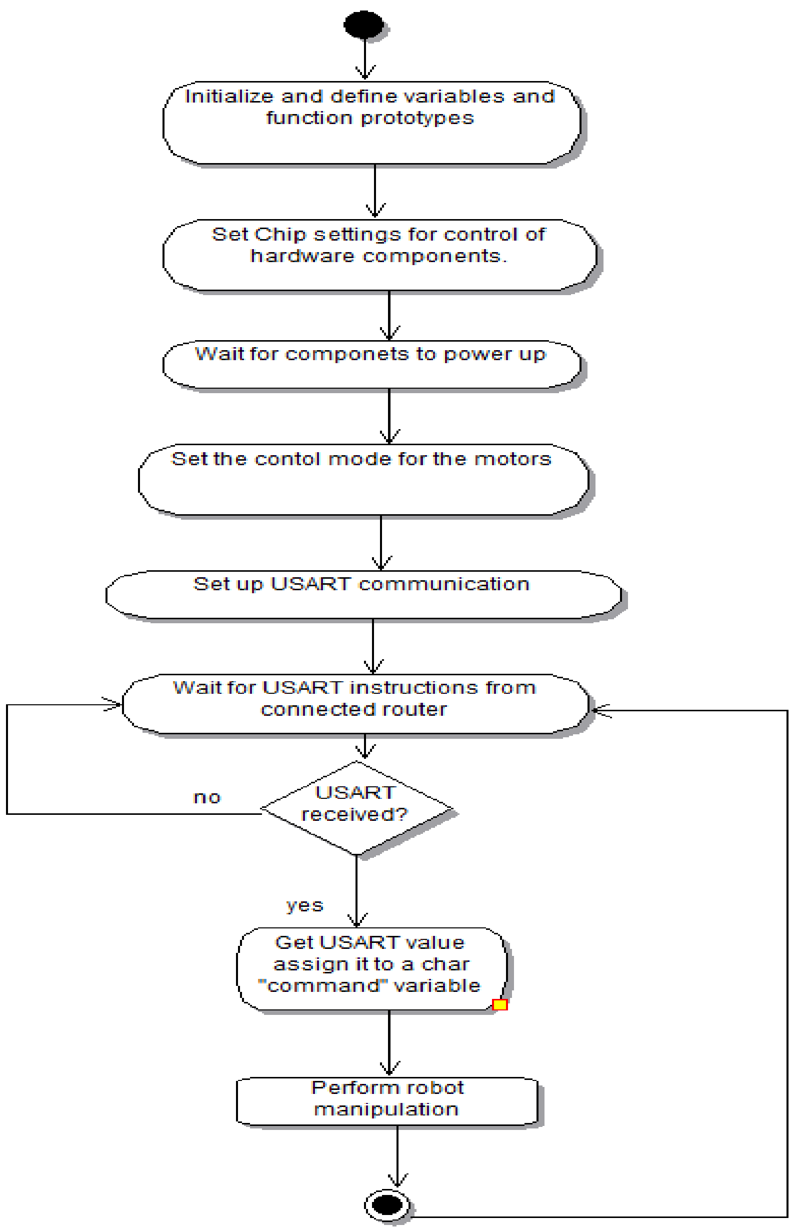

Initialize and define variables and function prototypes
Set Chip settings for control of hardware components.
Wait for componets to power up
Set the contol mode for the motors
Set up USART communication
Wait for USART instructions from connected router
USART received?
no
yes
Get USART value assign it to a char "command" variable
Perform robot manipulation

Figure 22: Mid-level design for controller program for the hardware controller

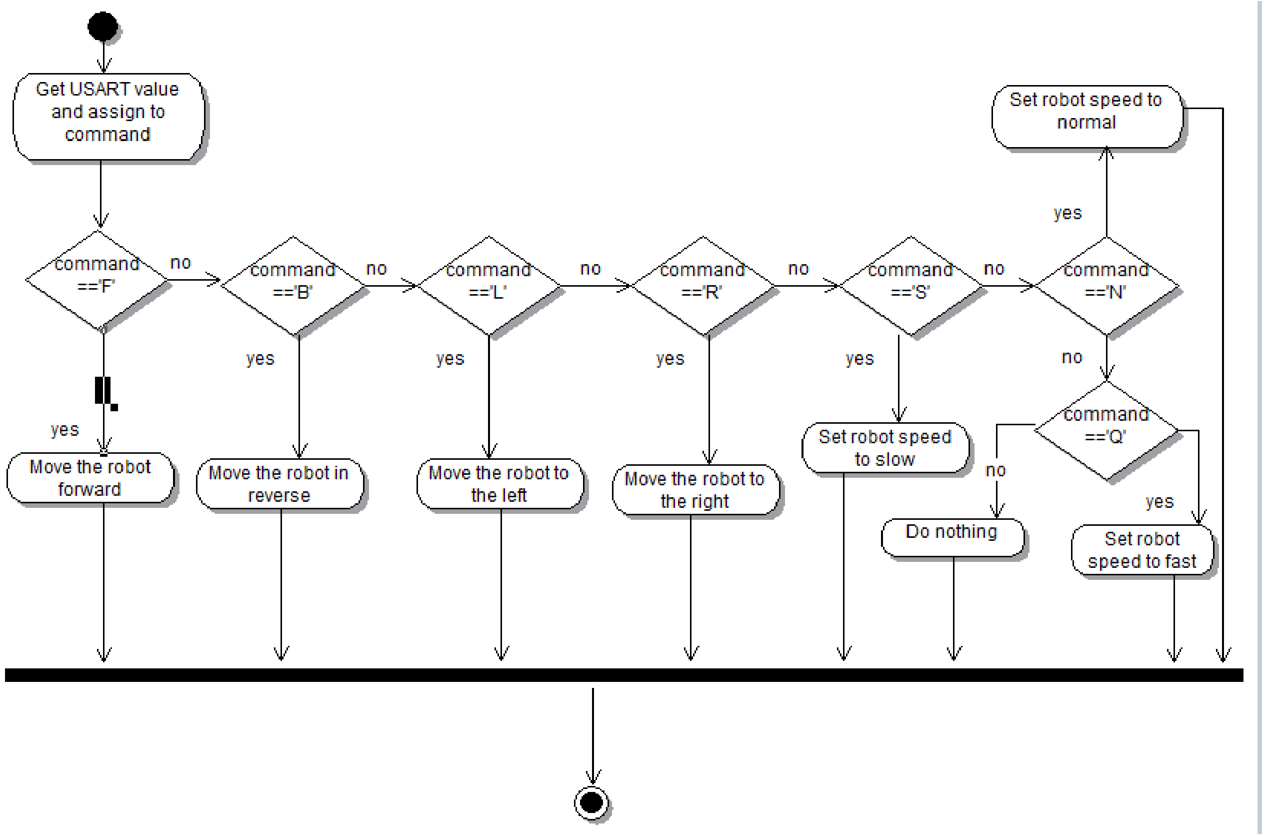


Figure 23: Low level program design for hardware response to user input

Object Detection and avoidance:

Though an actual implementation of obstacle navigation was not achieved during the course of this project because of certain drawbacks (see Chapter 7), an algorithmic design was developed, and that can realistically be embodied by the current hardware facilities available. To detect objects within a certain proximity range about the robot system while in motion, nine ultrasonic range sensors were to be connected to the robot's hardware controller (PIC). The figure below shows both a mid-level and low level design for object detection and avoidance.

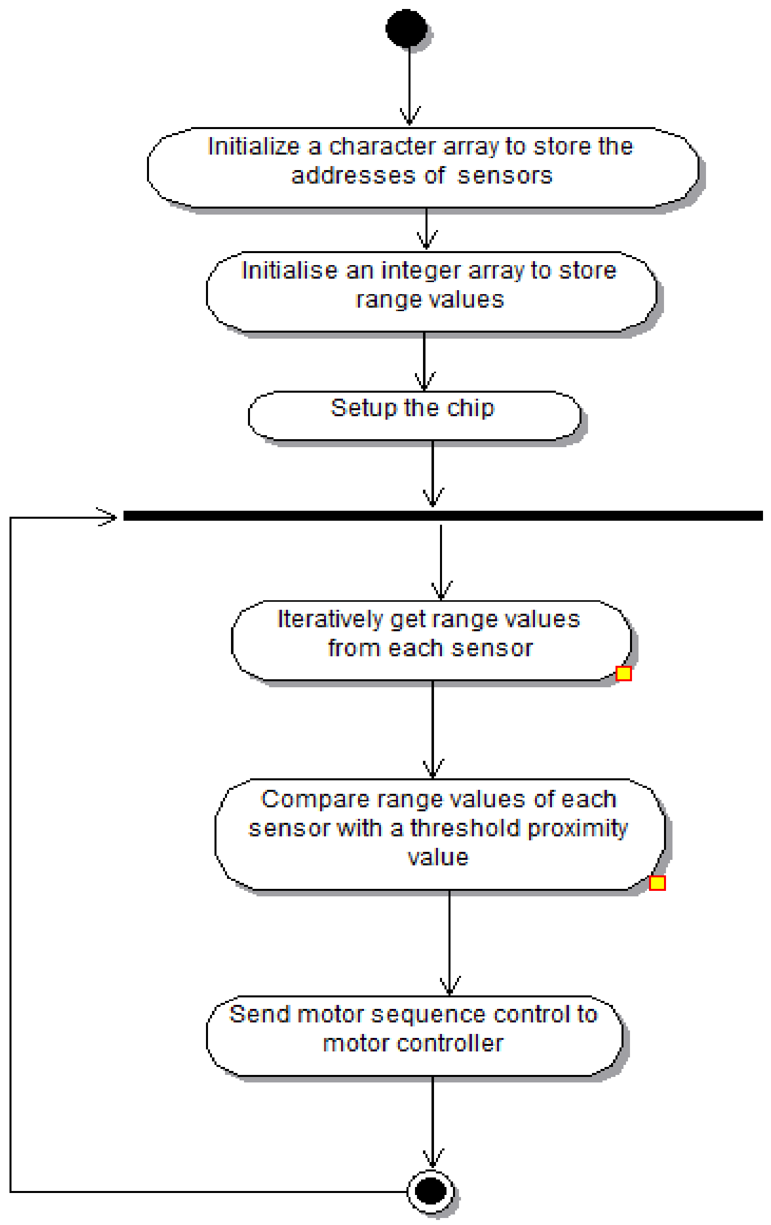
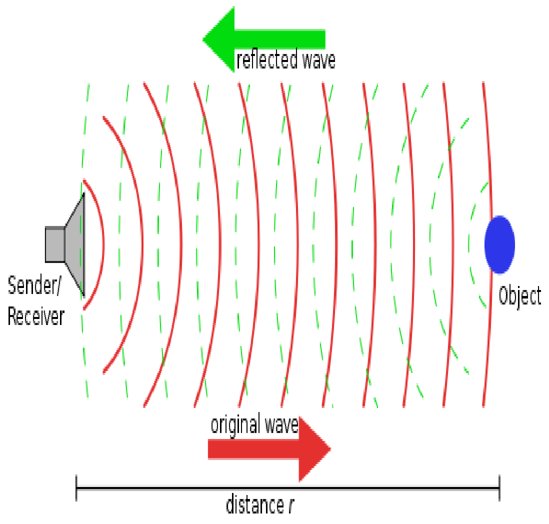

Initialize a character array to store the addresses of sensors
Initialise an integer array to store range values
Setup the chip
Iteratively get range values from each sensor
Compare range values of each sensor with a threshold proximity value
Send motor sequence control to motor controller

Figure 24: Mid-level activity diagram for illustrating obstacle detection and avoidance

As mentioned before there are nine sensors interfaced with the robot, such that there are three sensors on each side of the robot apart from the back (see figure below). The ultrasonic range sensors are such that they have a broad detection pattern as seen in the figure below. They work act as transceivers that emit high frequency sound waves and receive the echo of such sound waves when they come in contact with the surface of an object (pulse echo technique – referred to in this paper as PET from now on). The srf08 sensor used uses this ranging technique (PET), by emitting ultrasonic sound waves every 65μS (This can be changed for slower pulse time), It also has a maximum range of 600cm (6m), this maximum can also be changed to suit user preference.



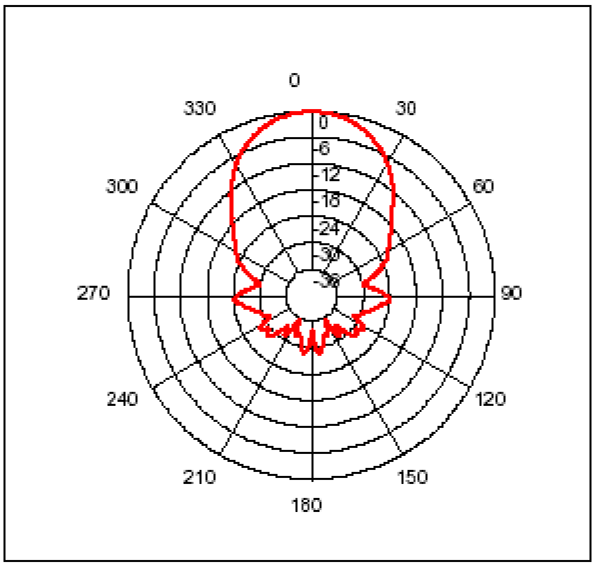


Figure 25: Illustration of pulse echo technique [left], Srf08 broadcast spectrum [right] (Dadachanji, 2013) (Robot Electronics 2014)

The srf08 range finder sensor uses gain settings to boost sensitivity for receiving echoes from relatively large distances (since sound wave disperses and attenuate relative to increasing distance), coupled with relatively broad detection spectrum, this makes it more effective at distance ranging than optical and infrared sensors. The drawback to this sensor and its operating property is that, though it has this beam angle pattern for proximity sensing only the relative distance to the object is provided as well as light readings, but not the angle of detection, this inability was taken into account for developing an algorithm.

The first low level activity is to get the sensor values for each sensor, the figure below illustrates this.

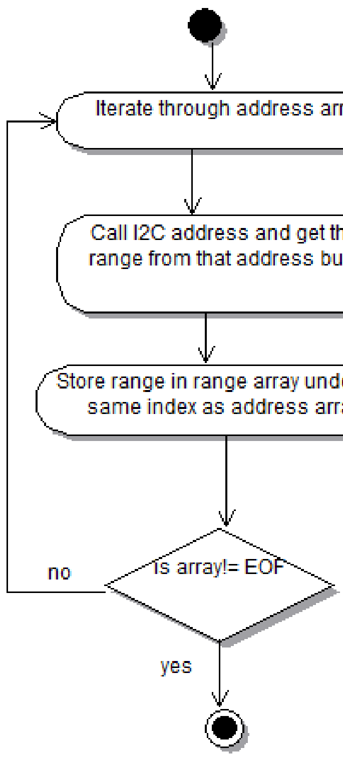

Iterate through address array
Call I2C address and get the range from that address bus.
Store range in range array under the same index as address array
is array!= EOF
no
yes

Figure 26: Iteration loop for getting sensor values.

To analyse the sensor values, first a proximity threshold parameter "d" must establish, this threshold acts as the minimum value be which any obstacle encountered in an environment can be proximity to the physical robot, by which objects detected inside these parameter i.e. closer than distance "d" in any direction detectable by the sensors, the robot will display a reactive response. Also as mentioned previously, the srf08 sensors have a beam transmission angle by which objects detected with the spectrum the sensors return the small distance (i.e. distance of the closest object). The figure below illustrates object detection using these sensors.

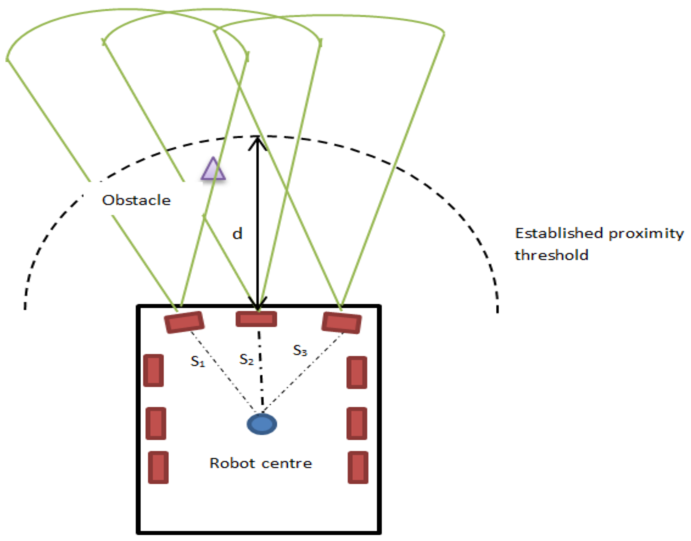


Figure 27: Possible object detection scenario

Having the sensors so close to each other, as shown in the figure above, there is a high probability of overlapping detection i.e. several sensors detecting the same object that may sometimes be at unequal distances. With these sensors inability to tell which direction (angular respective to each sensor), this may cause some ambiguity about the exact location of an obstacle. To account for this, a suggested algorithm as seen in the figure below could be used;

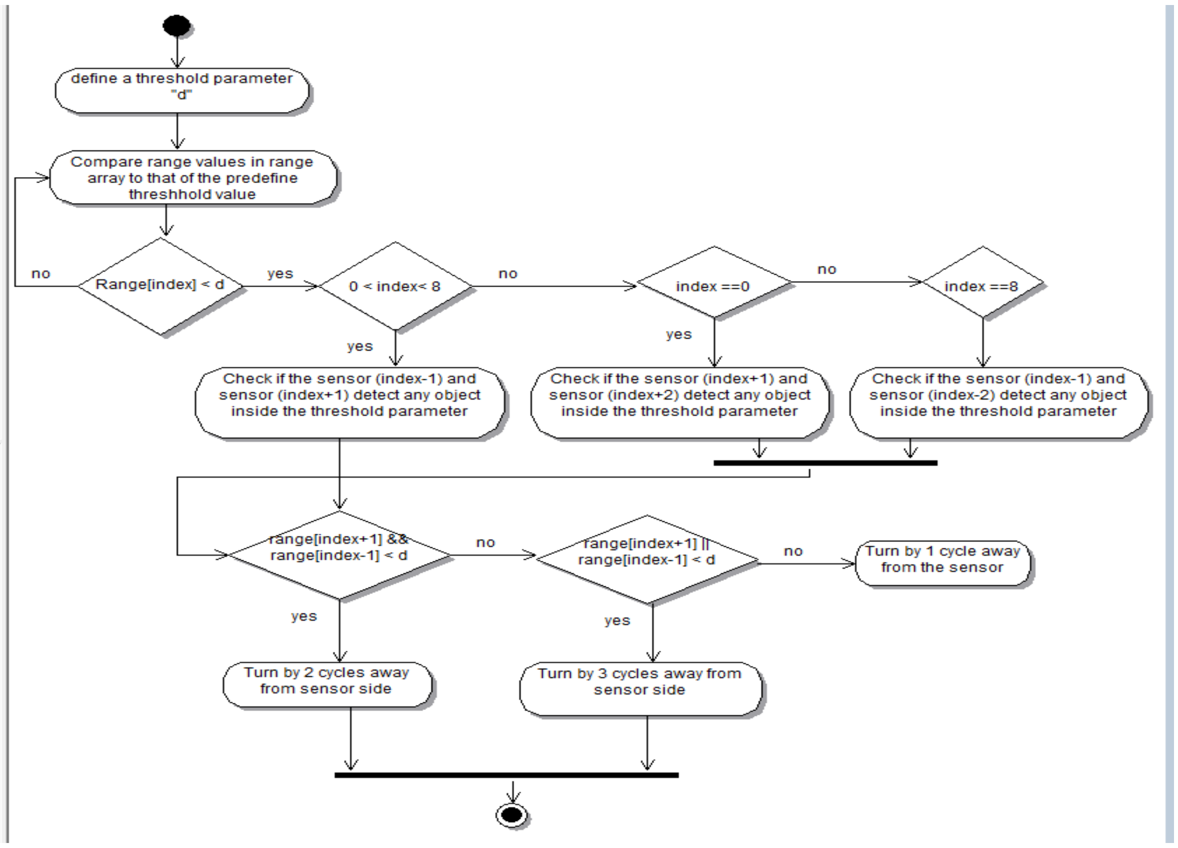

define a threshold parameter "d"
Compare range values in range array to that of the predefine threshhold value
no
Range[index] < d
yes
0 < index< 8
no
index ==0
no
index ==8
yes
yes
Check if the sensor (index-1) and sensor (index+1) detect any object inside the threshold parameter
Check if the sensor (index+1) and sensor (index+2) detect any object inside the threshold parameter
Check if the sensor (index-1) and sensor (index-2) detect any object inside the threshold parameter
range[index+1] && range[index-1] < d
no
range[index+1] || range[index-1] < d
no
Turn by 1 cycle away from the sensor
yes
yes
Turn by 2 cycles away from sensor side
Turn by 3 cycles away from sensor side

Figure 28: Low-level loop for comparing sensor ranges and responding

This algorithm suggests that when a sensor detects an object within the established threshold parameter, to get a sense of region location of the obstacle, neighbouring sensor range values should also be checked if they have also detected an obstacle(s), within the threshold, depend of the results of such evaluation the robot controller will send turn cycle commands (see Chapter 6) to the motor driver.

5.2.5 Locomotion

The software design for the system locomotion was simply to enable the robot to move in four directions- Forward, Reverse, Left, Right -. The degree of turn for each movement cycle for the left and right direction was chosen based on the position the sensors are situated on. This was designed for compatibility with the implementation of the obstacle navigation sequence proposed above and also because of the angular position in which the sensors are situated on the frame of the robot.

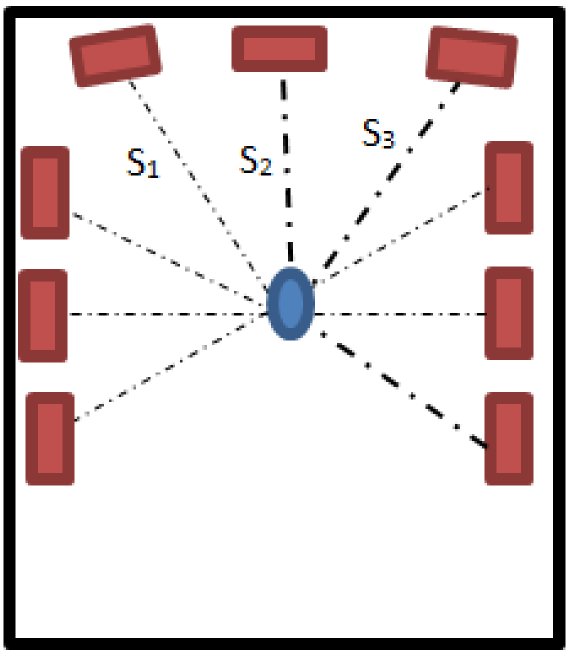


Figure 29: Sensor arrangement on robot frame

The sensors arranged in such a way that, they are about 30° away from each other, so the turn degree chosen was 30° also.

5.2.6 Graphic User Interface

To design GUI for controlling the system a use case scenario diagram was developed that captures the requirements for the GUI.

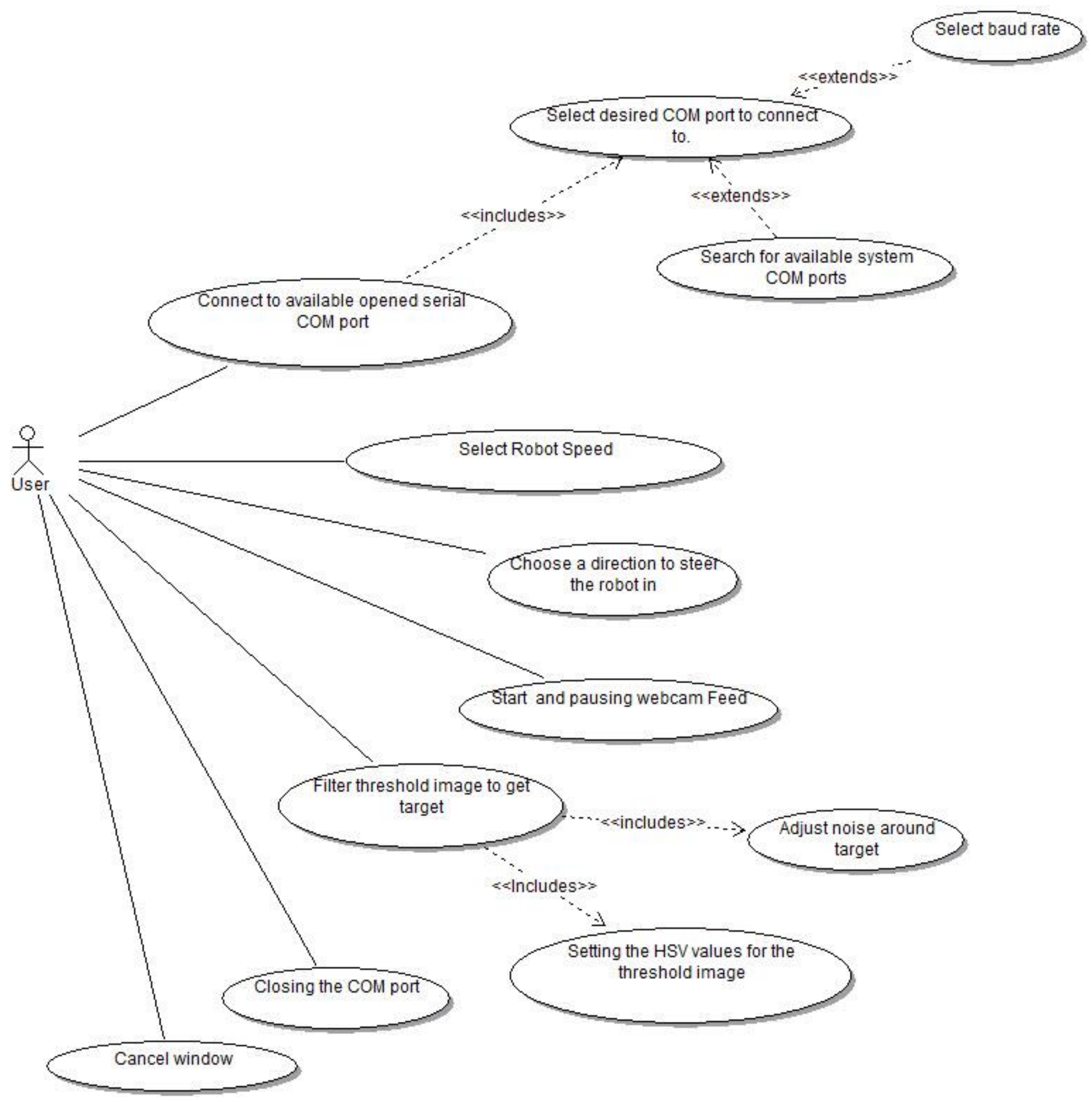

Select baud rate
<<extends>>
Select desired COM port to connect to.
<<includes>>
<<extends>>
Search for available system COM ports
Connect to available opened serial COM port
User
Select Robot Speed
Choose a direction to steer the robot in
Start and pausing webcam Feed
Filter threshold image to get target
<<includes>>
Adjust noise around target
<<Includes>>
Setting the HSV values for the threshold image
Closing the COM port
Cancel window

Figure 30: Use case scenario design for the GUI

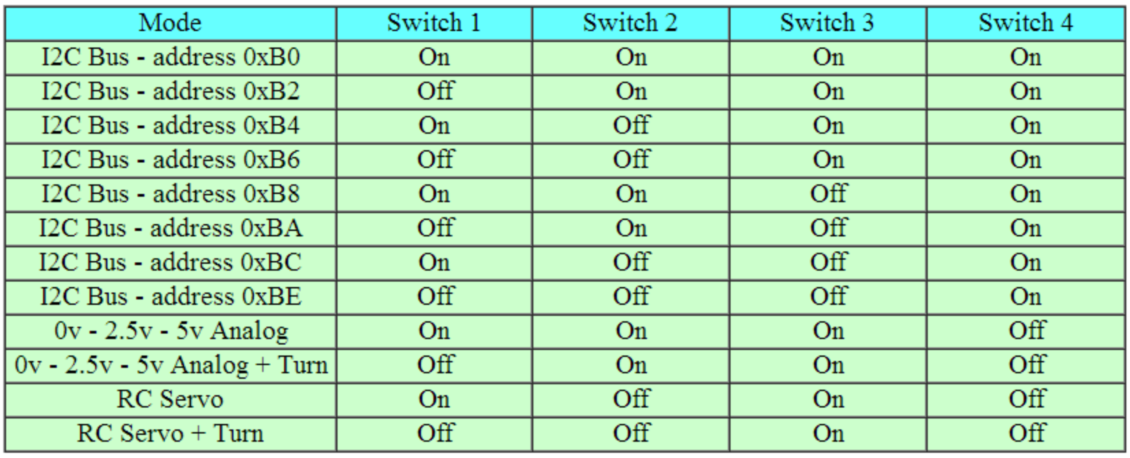
6 System Implementation

6.1 Hardware Implementation

6.1.1 Locomotion

Motor Control:

As previously stated (see Chapter 5), the motor driver (See Chapter 5 for component listing) has several modes operation, which are basically analogue and digital settings. However for connectivity sake as well more accurate control, the digital configuration mode was used (The operating modes of the motor driver can be easily changed using the mode switched - it is to be noted that operation mode change cannot be done while the driver is powered on, it must first be disconnected from the power supply before change the operating mode [Robot Electronics 2006]-). The digital operating mode has several addresses for master-slave I^2C communication with a hardware controller; these addresses are configured by various combinations of mode switches, the Table below shows which combination accesses modes and addresses.



Mode	Switch 1	Switch 2	Switch 3	Switch 4
I2C Bus - address 0xB0	On	On	On	On
I2C Bus - address 0xB2	Off	On	On	On
I2C Bus - address 0xB4	On	Off	On	On
I2C Bus - address 0xB6	Off	Off	On	On
I2C Bus - address 0xB8	On	On	Off	On
I2C Bus - address 0xBA	Off	On	Off	On
I2C Bus - address 0xBC	On	Off	Off	On
I2C Bus - address 0xBE	Off	Off	Off	On
0v - 2.5v - 5v Analog	On	On	On	Off
0v - 2.5v - 5v Analog + Turn	Off	On	On	Off
RC Servo	On	Off	On	Off
RC Servo + Turn	Off	Off	On	Off

Table 7: Mode switch selection combination (Robot Electronics 2014).

The driver requires two power supplies; an 11 - 12V power supply for powering the circuit board for the motor driver and also the differential wheel it drives one side and a 5V to drive the logic gates that control the signals received to the several H-bridge transducers that drive the differential wheels. These logic gates communicate with the robot controller via I^2C communication protocol and are the slave devices in this case. The SDA and SCL ports of the driver are connected to the SDA and SCL pins of the robot controller respectively (See Appendix B for pin layout of PIC micro controller).

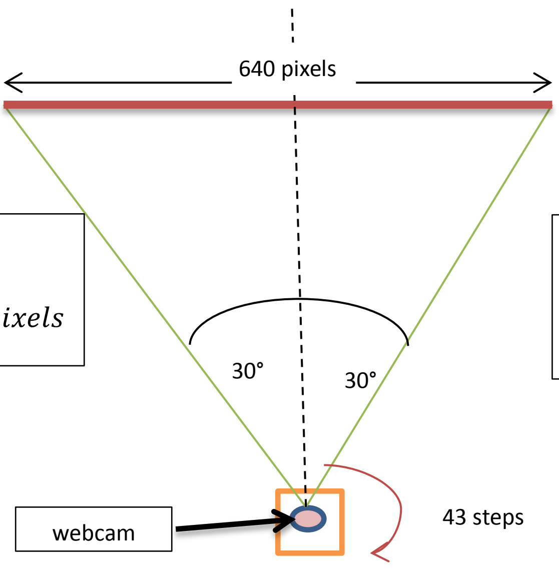
6.1.2 Target/Goal Tracking

Goal tracking was carried out using a wireless mini webcam (see Component list in Chapter 5) - which is powered separately by a 3.0V battery -. This camera broadcasts its own WLAN signal that can be picked up via ad-hoc connection to another wireless device or the camera can be connected to an infrastructural network for remote viewing over the internet (For this project to demonstrate extensibility it the webcam was connect to the cs-network via an independent WIFI router). The webcam has a 60° view angle and a streams 640 by 480 pixel area video. The webcam is mounted on a stepper motor controlled platform – the stepper motor rotates according the step sequence relayed by the robot controller (a half step sequence was used for stepper rotation). The stepper motor used is a 32-step motor; this means that for each step the stepper rotates by 11.25° in a given direction, with a gear reduction set of 1/16, which means it needs about 512 steps to complete a full cycle (360° rotation) (Adafruit 2014). In order words

$$360° = 512\ steps$$

$$11.25° = \frac{512}{360} \times 11.25 = 16\ steps$$

So for a view angle of 60°,



$$60° = 640\ pixels$$

$$1° = \frac{640}{60} = 10.67 \approx 11 pixels$$

To rotate the camera by half its view angle, it will take $\frac{512}{360} \times 30 \approx 43 steps$

Figure 31: Camera target tracking setup

With this knowledge it was easy calculating the steps needed for small changes in the frame by translating change in pixel coordinate points to angles, then to step sequences to serve to the stepper motor for rotating the camera accordingly.

$$Steps \approx \frac{512}{360} \times x°$$

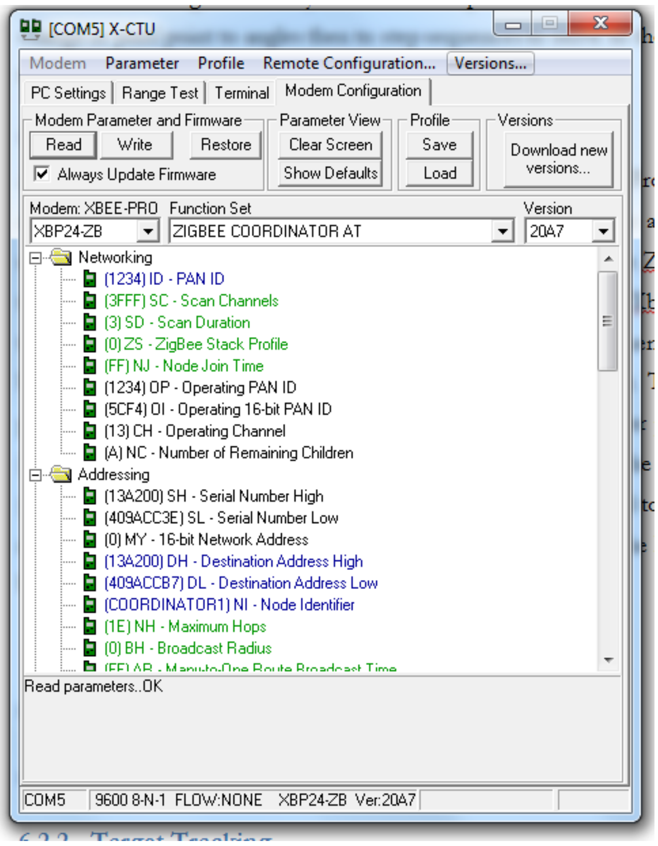
The step sequence used is that of a half-step sequence, for smoother more accurate control.

Step	Sequence
0	0x0C
1	0x04
2	0x06
3	0x02
4	0x03
5	0x01
6	0x09
7	0x08

Table 8: Half step sequence for stepper rotation

6.1.3 Communication

A communication network was established using two Zigbee Pro – Series 2 radio frequency modules for communication between the controlling computer and the robot controller to conveying messages across each other. The Zigbee module is connected to the controlling computer (via USB serial port connection using a Xbee explorer module), this is designated as the "COORDINATOR" module that broadcasts the radio frequency network to be picked up by the "ROUTER" module connected to the robot controller. The "ROUTER" module is powered by a 3.3V regulated supply and connects its transmitter pin (Dout) and receiver pin (Din) to the USART receiver pin (rx) and transmitter pin (tx) on the robot controller respectively (please see Appendix B for pin layout). Each Zigbee module had to be configured individually, matching destination addresses as well as the network ID, using the X-CTU software developed by Digi.


[COM5] X-CTU
Modem Parameter Profile Remote Configuration... Versions...
PC Settings Range Test Terminal Modem Configuration
Modem Parameter and Firmware
Read Write Restore
Always Update Firmware
Parameter View
Clear Screen
Show Defaults
Profile
Save
Load
Versions
Download new versions...

Modem: XBEE-PRO XBP24-ZB
Function Set ZIGBEE COORDINATOR AT
Version 20A7

```
Networking
    (1234) ID - PAN ID
    (3FFF) SC - Scan Channels
    (3) SD - Scan Duration
    (0) ZS - ZigBee Stack Profile
    (FF) NJ - Node Join Time
    (1234) OP - Operating PAN ID
    (5CF4) OI - Operating 16-bit PAN ID
    (13) CH - Operating Channel
    (A) NC - Number of Remaining Children
Addressing
    (13A200) SH - Serial Number High
    (409ACC3E) SL - Serial Number Low
    (0) MY - 16-bit Network Address
    (13A200) DH - Destination Address High
    (409ACCB7) DL - Destination Address Low
    (COORDINATOR1) NI - Node Identifier
    (1E) NH - Maximum Hops
    (0) BH - Broadcast Radius
    (FF) AR - Many-to-One Route Broadcast Time
Read parameters..OK
COM5  9600 8-N-1 FLOW:NONE  XBP24-ZB Ver:20A7
```

Figure 32: X-CTU setup for configuring Zigbee

Instructions for setting up the zigbees can be found in reference below (Ober 2012).

6.2 Software Implementation

6.2.1 Locomotion

Motor Control:

As stated earlier, the MD22 motor driver can be programmed for digital operation. Using I^{2}C protocol, the robot controller is the designated master device and therefore drives the clock-line SCL, and the motor driver is the slave device. By transmitting data via the I^{2}C bus of the robot controller, motion step sequences can programmed into the driver. Data transfer is in a byte sequence (8 bits) and are carried by the SDA line (Robot Electronics 2014).

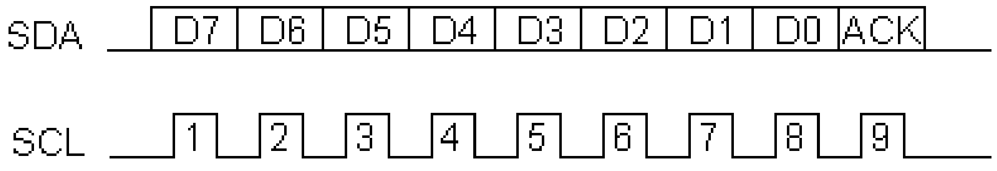
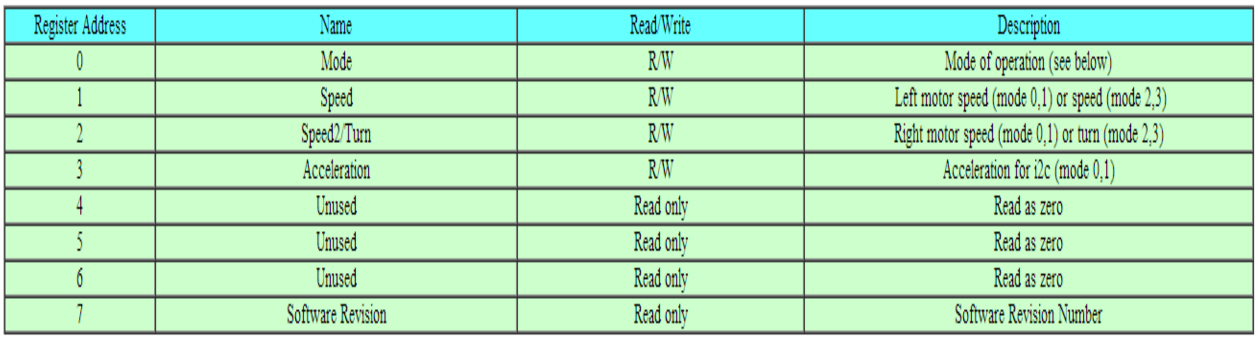


Figure 33: Bit representation of the SDA and SCL signals (Robot Electronics 2014)

The digital configuration for driving the motor has several operational modes that can be chosen simply via writing a mode address to the mode register of the MD22.



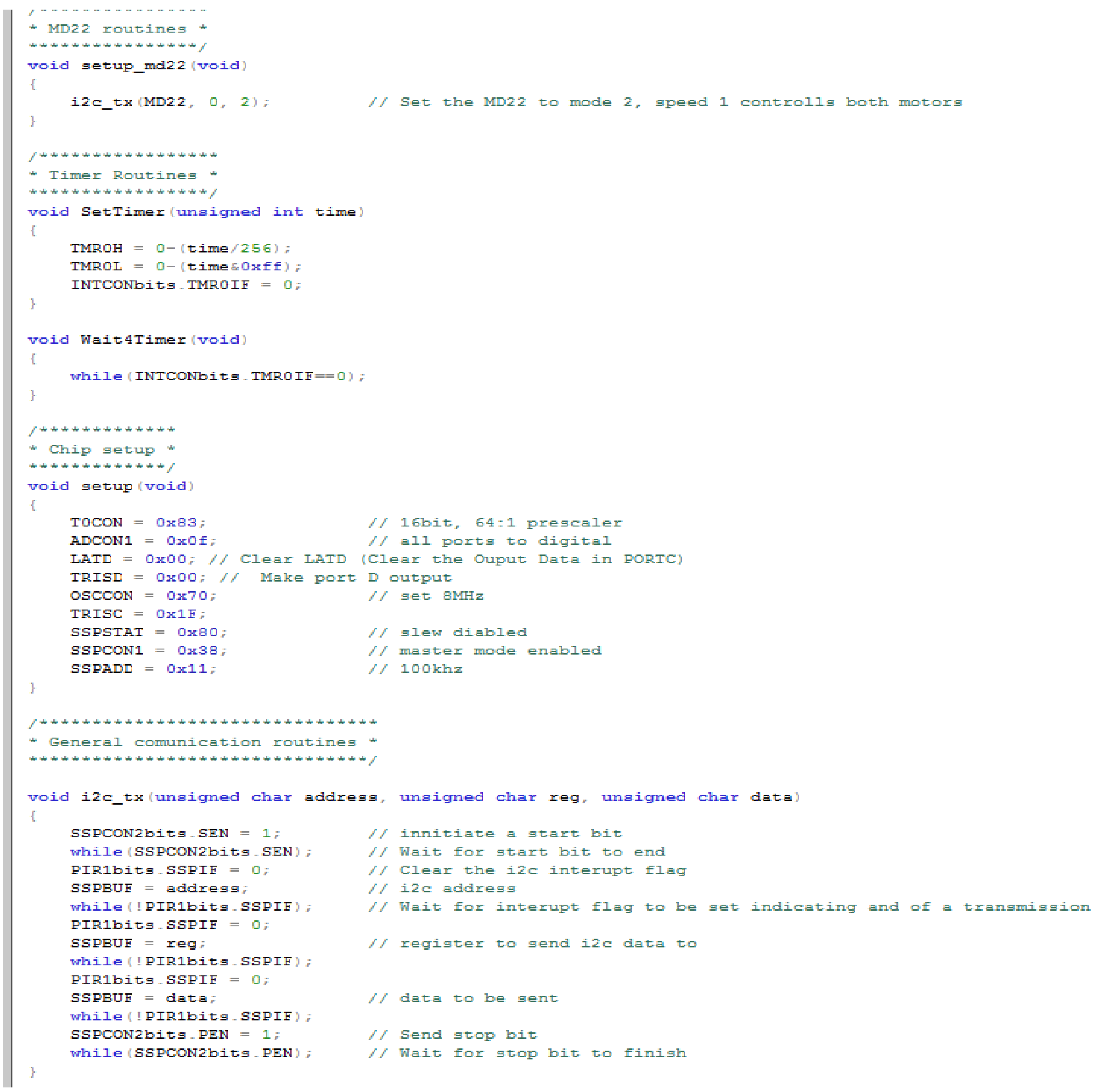
Register Address	Name	Read/Write	Description
0	Mode	R/W	Mode of operation (see below)
1	Speed	R/W	Left motor speed (mode 0,1) or speed (mode 2,3)
2	Speed2/Turn	R/W	Right motor speed (mode 0,1) or turn (mode 2,3)
3	Acceleration	R/W	Acceleration for i2c (mode 0,1)
4	Unused	Read only	Read as zero
5	Unused	Read only	Read as zero
6	Unused	Read only	Read as zero
7	Software Revision	Read only	Software Revision Number

Table 9: Digital operational modes (Robot Electronics 2014)

The operational modes used during software implementation were the speed mode (reg address 1), which controls the speed at which the MD22 drives the differential wheels - the *speed* setting has a range; -128 (Full Reverse) 0 (Stop) 127 (Full Forward). - , and the *speed2/Turn*, this dictates the turn degree – this setting has a range -128 (90° right) 0 (90° left) 127 (stop). Although the turn range is stated differently in the documentation for the MD22 driver (Robot Electronics 2014), component testing revealed otherwise stated. The three speed levels used for implementation were "75", "50" and "25", which were designated slow, normal and fast respectively.

From the hardware implementation sub-chapter, a turn degree of about approximately 30° is needed to facilitate the design for the object detection and avoidance, from the *speed2/turn* range it was calculated that the range data value to be passed to the turn register should be 86 (for left turn) and –86 (for right turn). It is also noteworthy to mention that the amount of steps required to achieve a full cycle of motion is 255 steps, for the implementation two movement cycles (512 steps) for each directional motion was used.

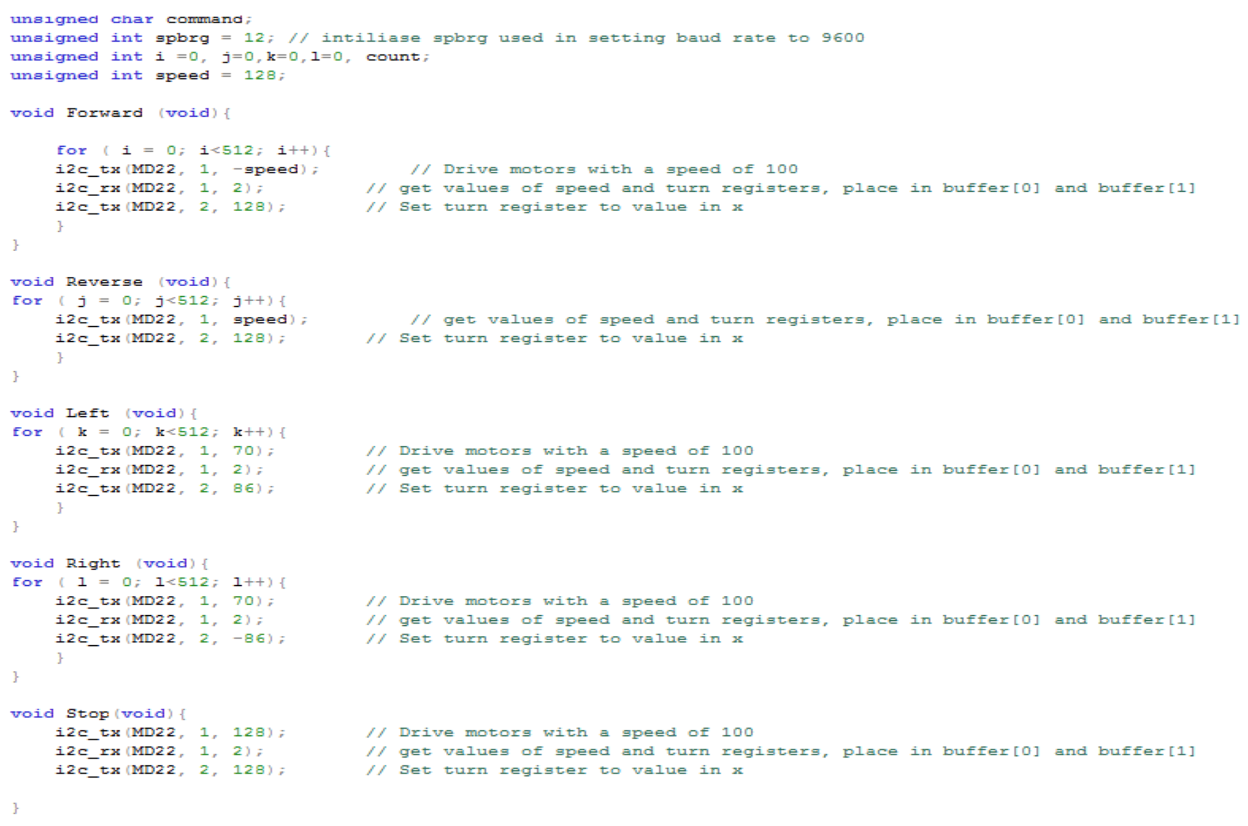
Robot Electronics* (2014), have uploaded in their database a series of I^2C protocol examples for various devices including the MD22, so the program code for controlling the driver was not written from scratch. It includes; chip settings, timer settings as well as the routine for master-slave I^2C bus communication, these codes are all written in C language and designed for microcontrollers.



```
/****************
* MD22 routines *
****************/
void setup_md22(void)
{
    i2c_tx(MD22, 0, 2);          // Set the MD22 to mode 2, speed 1 controlls both motors
}

/*****************
* Timer Routines *
*****************/
void SetTimer(unsigned int time)
{
    TMR0H = 0-(time/256);
    TMR0L = 0-(time&0xff);
    INTCONbits.TMR0IF = 0;
}

void Wait4Timer(void)
{
    while(INTCONbits.TMR0IF==0);
}

/*************
* Chip setup *
*************/
void setup(void)
{
    T0CON = 0x83;                // 16bit, 64:1 prescaler
    ADCON1 = 0x0f;               // all ports to digital
    LATD = 0x00; // Clear LATD (Clear the Ouput Data in PORTC)
    TRISD = 0x00; //  Make port D output
    OSCCON = 0x70;               // set 8MHz
    TRISC = 0x1F;
    SSPSTAT = 0x80;              // slew diabled
    SSPCON1 = 0x38;              // master mode enabled
    SSPADD = 0x11;               // 100khz
}

/*********************************
* General comunication routines *
*********************************/

void i2c_tx(unsigned char address, unsigned char reg, unsigned char data)
{
    SSPCON2bits.SEN = 1;         // inniitiate a start bit
    while(SSPCON2bits.SEN);      // Wait for start bit to end
    PIR1bits.SSPIF = 0;          // Clear the i2c interupt flag
    SSPBUF = address;            // i2c address
    while(!PIR1bits.SSPIF);      // Wait for interupt flag to be set indicating and of a transmission
    PIR1bits.SSPIF = 0;
    SSPBUF = reg;                // register to send i2c data to
    while(!PIR1bits.SSPIF);
    PIR1bits.SSPIF = 0;
    SSPBUF = data;               // data to be sent
    while(!PIR1bits.SSPIF);
    SSPCON2bits.PEN = 1;         // Send stop bit
    while(SSPCON2bits.PEN);      // Wait for stop bit to finish
}
```

Figure 34: Caption of implemented code for I²C communication protocol written by Robot Electronics (2014)

For readability and maintenance purposes the functionalities created by calling for locomotion – Forward movement, Reverse, Left and Right- where factored into appropriately named functions that were called in by the *main()* as needed. These functions are simply executing for loop, such that it facilitates 2 cycles of movement for the differential motors.



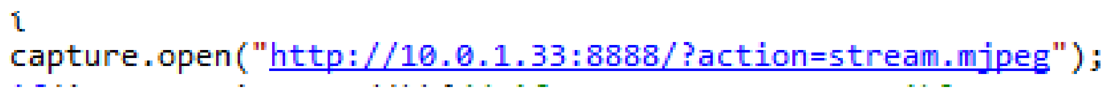
```
unsigned char command;
unsigned int spbrg = 12; // intiliase spbrg used in setting baud rate to 9600
unsigned int i =0, j=0,k=0,l=0, count;
unsigned int speed = 128;

void Forward (void){

    for ( i = 0; i<512; i++){
    i2c_tx(MD22, 1, -speed);         // Drive motors with a speed of 100
    i2c_rx(MD22, 1, 2);         // get values of speed and turn registers, place in buffer[0] and buffer[1]
    i2c_tx(MD22, 2, 128);       // Set turn register to value in x
    }
}

void Reverse (void){
for ( j = 0; j<512; j++){
    i2c_tx(MD22, 1, speed);          // get values of speed and turn registers, place in buffer[0] and buffer[1]
    i2c_tx(MD22, 2, 128);       // Set turn register to value in x
    }
}

void Left (void){
for ( k = 0; k<512; k++){
    i2c_tx(MD22, 1, 70);        // Drive motors with a speed of 100
    i2c_rx(MD22, 1, 2);         // get values of speed and turn registers, place in buffer[0] and buffer[1]
    i2c_tx(MD22, 2, 86);        // Set turn register to value in x
    }
}

void Right (void){
for ( l = 0; l<512; l++){
    i2c_tx(MD22, 1, 70);        // Drive motors with a speed of 100
    i2c_rx(MD22, 1, 2);         // get values of speed and turn registers, place in buffer[0] and buffer[1]
    i2c_tx(MD22, 2, -86);       // Set turn register to value in x
    }
}

void Stop(void){
    i2c_tx(MD22, 1, 128);       // Drive motors with a speed of 100
    i2c_rx(MD22, 1, 2);         // get values of speed and turn registers, place in buffer[0] and buffer[1]
    i2c_tx(MD22, 2, 128);       // Set turn register to value in x

}
```

Figure 35: Caption of implemented code for locomotion in four directions

6.2.2 Target Tracking

Image Processing:

Functions for processing an image of a particular frame were derived from the Opencv C++ library created by IBM (see Chapter 2). These include functions for opening a video capture from a connected device (remote or otherwise), functions to help locate objects with a certain HSV range, converting BGR images to greyscale, HSV or RGB formats, edge detection, dilating and erosion of noise in image threshold, creating image matrices for storing image properties etc. The process by which images are grabbed form the webcam by the system controller is seen in the figure below. An video stream was be captured parsing the stream through an Opencv defined class variable called "*VideoCapture*", to open a video stream, a function "*open()*" of this class is called that takes the address of the video stream to be parsed, in this case the video stream was being parsed from a router port connected to the *cs* network, the address for this stream is *http://10.0.1.33:8888/?action=stream.mjpeg*.



```
{
capture.open("http://10.0.1.33:8888/?action=stream.mjpeg");
```

Figure 36: code lien that capture the streamed images via a wireless router

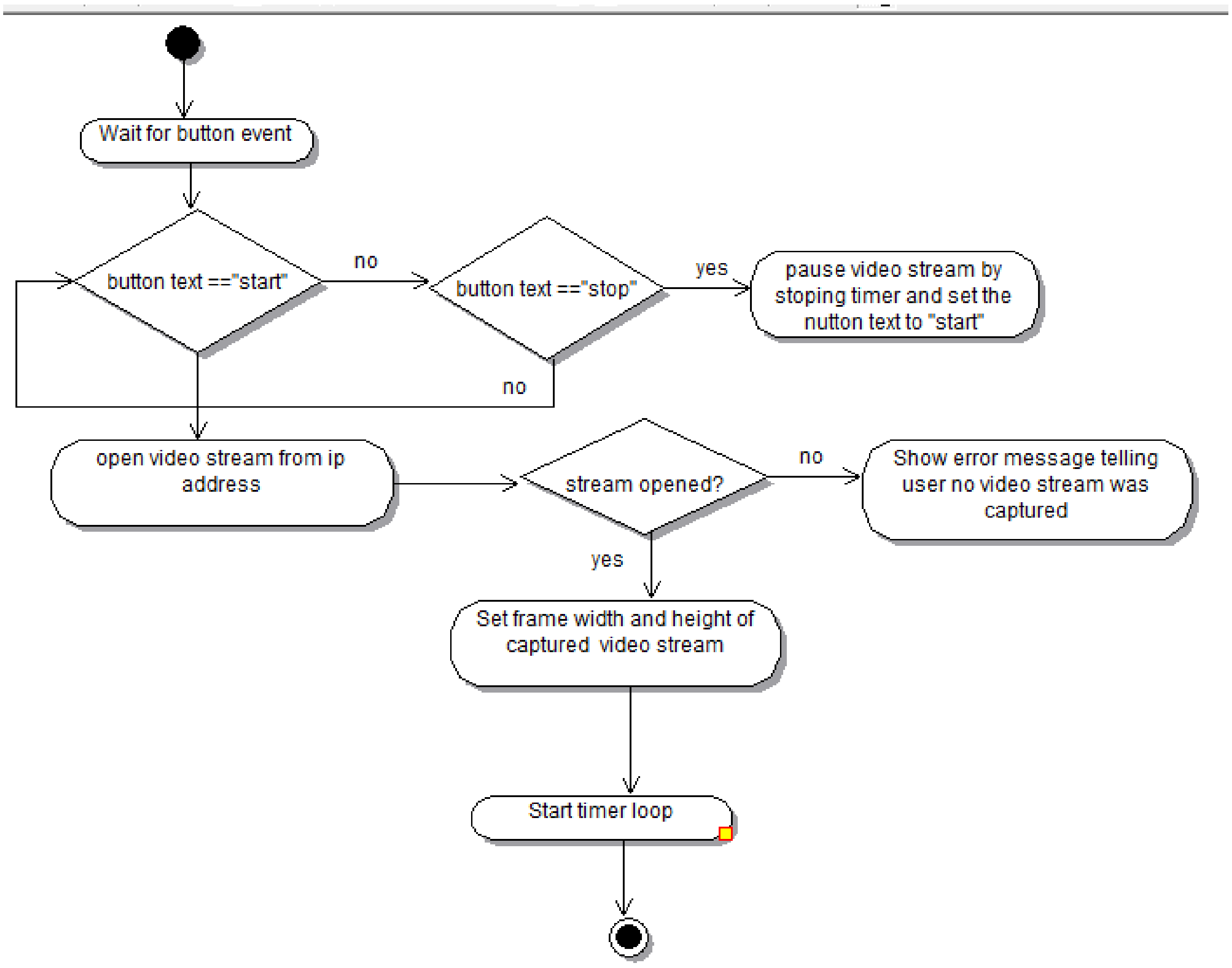

Wait for button event
button text =="start"
no
button text =="stop"
yes
pause video stream by stoping timer and set the nutton text to "start"
no
open video stream from ip address
stream opened?
no
Show error message telling user no video stream was captured
yes
Set frame width and height of captured video stream
Start timer loop

Figure 37: overall video capture process sequence

Once the function above is able to capture a video stream, image processing for each frame can begin, this involves;

- Converting a BGR frame to HSV, this makes each frame easier to process using the **cvtColor()** opencv function, this function takes 3 arguments – two image matrices, one which is the input matrix which is the image to be converted, the other is the output matrix, where the converted images arrays are stored, then what kind of conversion needed, at this point only a colour conversion from BGR to HSV is needed.

```
//convert frame from BGR to HSV colorspace
cv::cvtColor(cameraFeed,HSV,cv::COLOR_BGR2HSV);
```

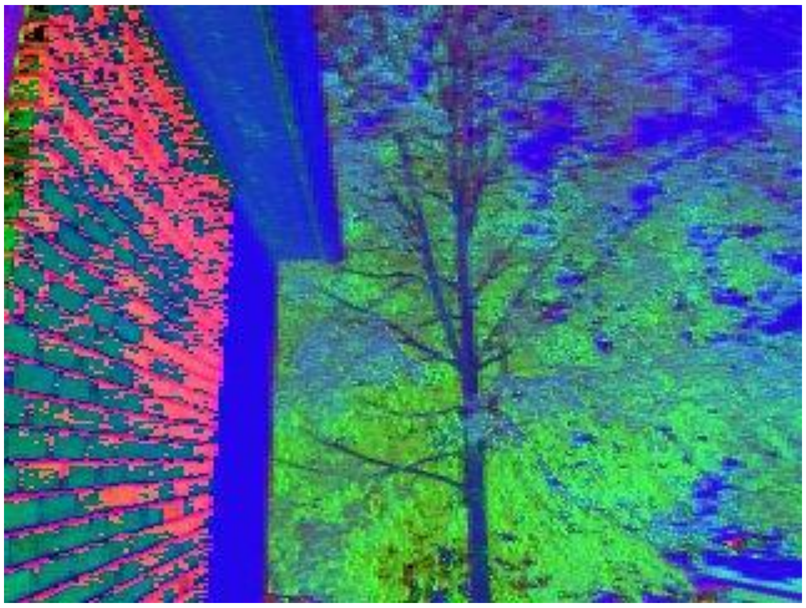
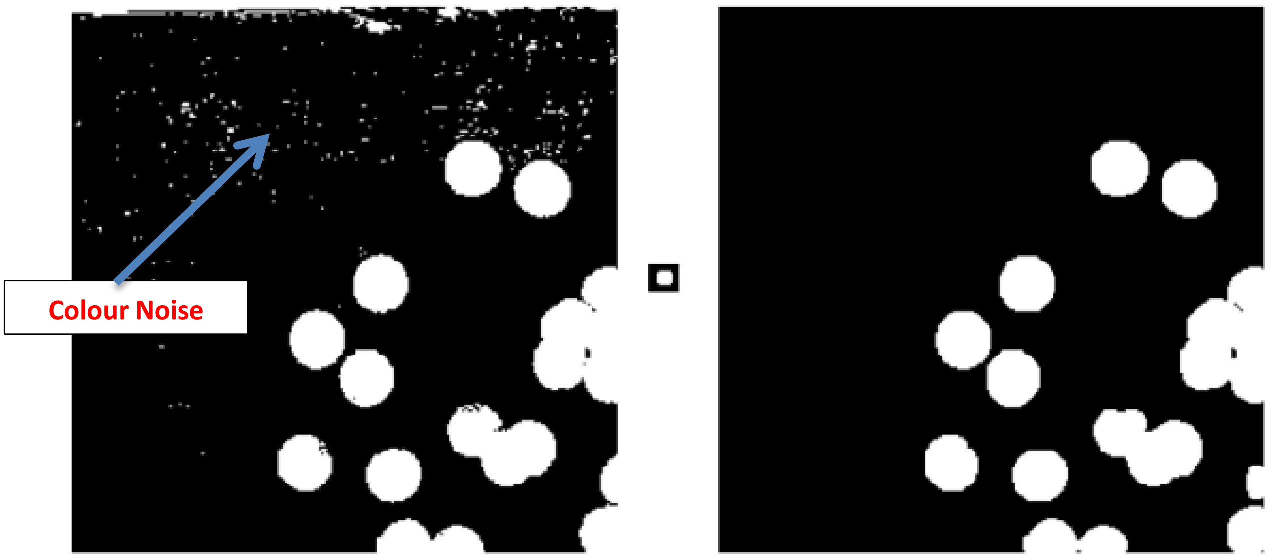


Figure 38: example of an HSV image (Nirav Patel 2009)

- Using the HSV frame matrix, the program can easily filter out unwanted HSV values to acquire a target, this filtered frame is then stored in a threshold image matrix frame. This filtering occurs in real time and does not affect the video stream. The implementation of this is facilitated using the ***inRange()*** open cv function, this takes four arguments, the input image matrix, in this case the HSV image, then the minimum scale for the HSV values (0-255), then the maximum scale for the HSV values (also 0-255), then where to store the output image matrix.
- The thresh hold image can then be manipulated to get a clearer image by filtering out unwanted colour noise and enhancing large colour spaces. The two functions doing so are the opencv functions ***blur()*** and ***threshold()***, the first function takes is used to smoothen an image of a certain size, it takes three arguments; an input matrix array, which is the threshold image matrix if the is unwanted colour noise, and the an output image matrix, this is also the threshold image, next argument is the another Opencv function called ***Sise()*** this takes two integer arguments that compute as the size of the image to smoothen. The second function is a simple segmentation method, that separates pixels of interests from the rest (OpenCV 2014), it takes five arguments, 2 image matrices (input and output), the threshold colour scale (between 0 - 255), the maximum colour scale (255), and in what kind thresholding operation is needed (it is set to binary thresholding which is expressed as

$$destination\,(x,y) = \begin{cases} \max value \;\; if\; source(x,y) > threshold \\ 0 \qquad\qquad\qquad\qquad\quad otherwise \end{cases}$$ (OpenCV 2014)

)



6.3 Figure 39: noise colour filtering (AP186 Basic Video Processing 2012)

- After this this filtering process, a target will be confidently identified accordingly, tracking the targeting real time can be enabled. This is done using the trackFilteredObject function in the Image processing class

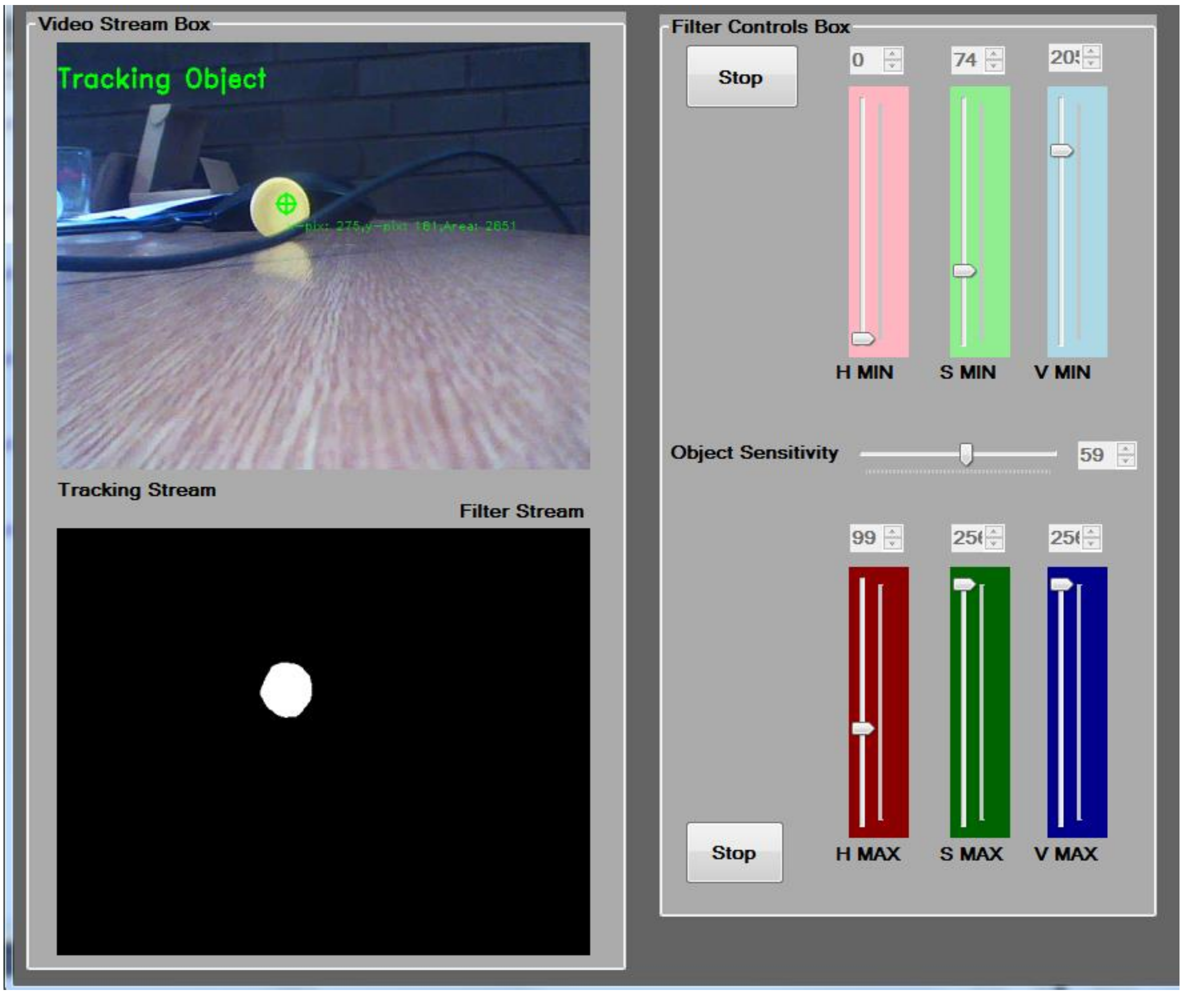


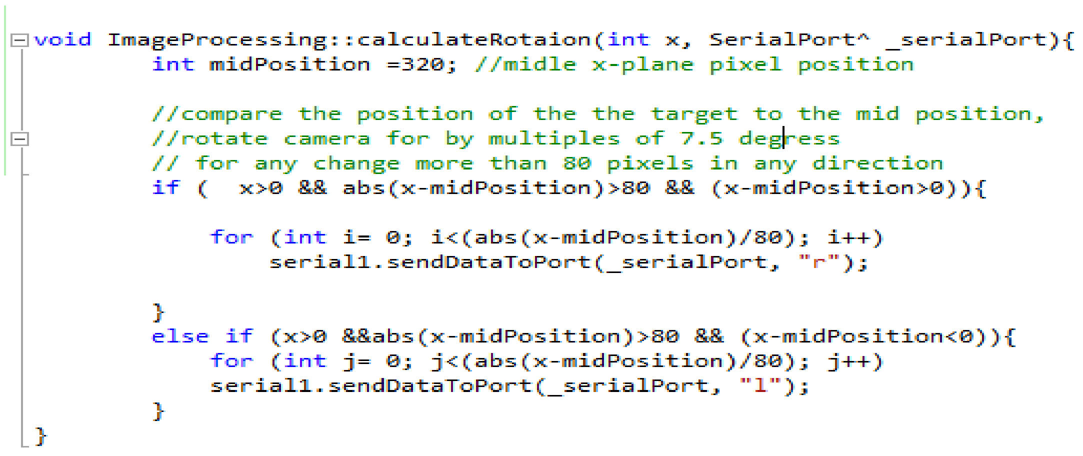
Figure 40: Image processing in the GUI

Tracking:

Tracking is enabled by pressing the "*Track*" button. Tracking is done by edge detection using an opencv function called "*findContours()*", this function takes several arguments including the threshold image matrix, and vector variables etc. In order to identify real object a minimum threshold area of about 400 pixels was set, also to avoid images too large for the frame being tracked a maximum object area of one-third of the frame area was also set, in other words, for an object to be detected;

```
//if the area is less than 20 px by 20px then it is probably just noise
//if the area is the same as the 3/2 of the image size, probably just a bad filter
//we only want the object with the largest area so we safe a reference area each
//iteration and compare it to the area in the next iteration.
if(area>MIN_OBJECT_AREA && area<MAX_OBJECT_AREA && area>refArea){
```

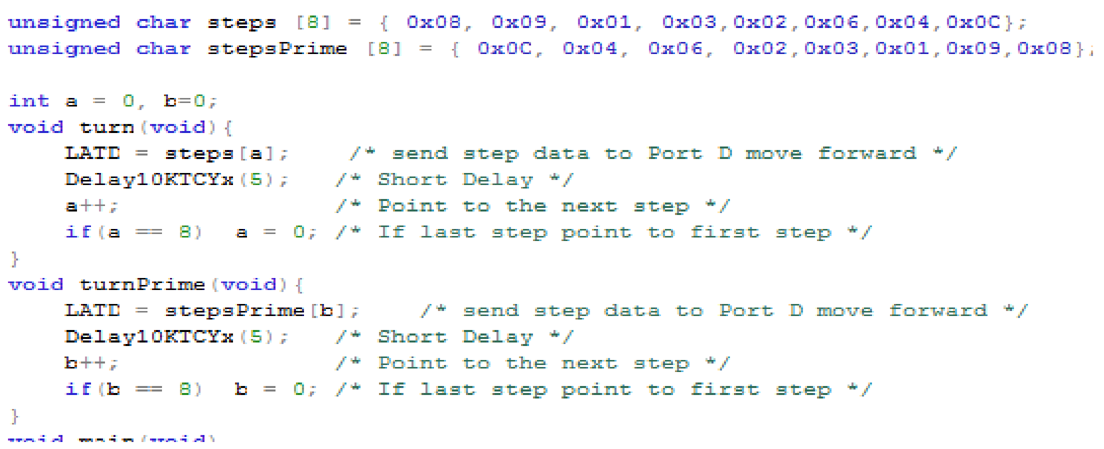
Properties of objects such as their x & ypositions and area in pixelated frame space is calculated using the *opencv* "*moment*" class variable and "*moments*" function, when an object is found a crosshair is draw at the centre position of the object as this is what is being tracked, opencv also has functions that allow one to draw simple shapes and text inside an image frame. Once the pixel position on the x-plane is acquired a function called "*calculateRotation()*"can be called to calculate how many rotation cycles is needed to keep a target within the frame. When implementing this, it was decided that the camera should track any 80 pixel change in position in the x-axis, by calculation based on the view angle of the webcam (see Hardware implementation sub-chapter), a 7.5° degree rotation cycle for the stepper (15 steps) must be implemented to follow such position change (the aim is to keep a target as close to the x-plane mid-point as possible). This gives the target tracking subsystem of the robot some autonomy, while the user remotely controls the main robot system. Directional signals are sent to the robot via the computer for stepper motor control, using the sendDataToPort() function of the SerialComm class.



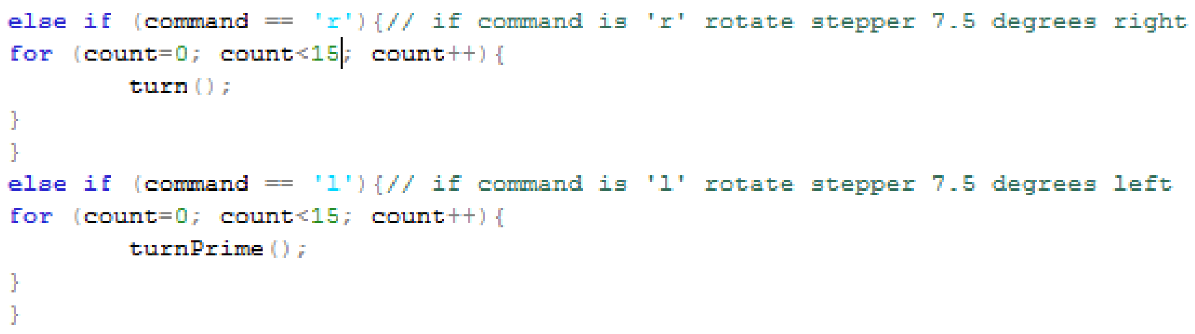
```
void ImageProcessing::calculateRotaion(int x, SerialPort^ _serialPort){
        int midPosition =320; //midle x-plane pixel position

        //compare the position of the the target to the mid position,
        //rotate camera for by multiples of 7.5 degress
        // for any change more than 80 pixels in any direction
        if (  x>0 && abs(x-midPosition)>80 && (x-midPosition>0)){

            for (int i= 0; i<(abs(x-midPosition)/80); i++)
                serial1.sendDataToPort(_serialPort, "r");

        }
        else if (x>0 &&abs(x-midPosition)>80 && (x-midPosition<0)){
            for (int j= 0; j<(abs(x-midPosition)/80); j++)
            serial1.sendDataToPort(_serialPort, "l");
        }
}
```

Figure 41: PC controller code extract for determining rotation cycle for web cam subsystem

On the robot side, the controller receives the instructions for rotation sequentially every 700ms if more than one turn cycle is needed, a single step sequence for the stepper is an iteration of the hexadecimal half step values in Table 8 above, to iterate through these steps, they were defined in an array and then using a *For* loop each step sequence is passed into the port D pins 0-3 of the micro controller which is configured as an output port, they are then sent to the stepper motors.



```
unsigned char steps [8] = { 0x08, 0x09, 0x01, 0x03,0x02,0x06,0x04,0x0C};
unsigned char stepsPrime [8] = { 0x0C, 0x04, 0x06, 0x02,0x03,0x01,0x09,0x08};

int a = 0, b=0;
void turn(void){
    LATD = steps[a];    /* send step data to Port D move forward */
    Delay10KTCYx(5);   /* Short Delay */
    a++;               /* Point to the next step */
    if(a == 8)  a = 0; /* If last step point to first step */
}
void turnPrime(void){
    LATD = stepsPrime[b];    /* send step data to Port D move forward */
    Delay10KTCYx(5);   /* Short Delay */
    b++;               /* Point to the next step */
    if(b == 8)  b = 0; /* If last step point to first step */
}
```



```
else if (command == 'r'){// if command is 'r' rotate stepper 7.5 degrees right
for (count=0; count<15; count++){
        turn();
}
}
else if (command == 'l'){// if command is 'l' rotate stepper 7.5 degrees left
for (count=0; count<15; count++){
        turnPrime();
}
}
```

Figure 42: Robot Controller webcam system stepper control

6.3.1 Communication

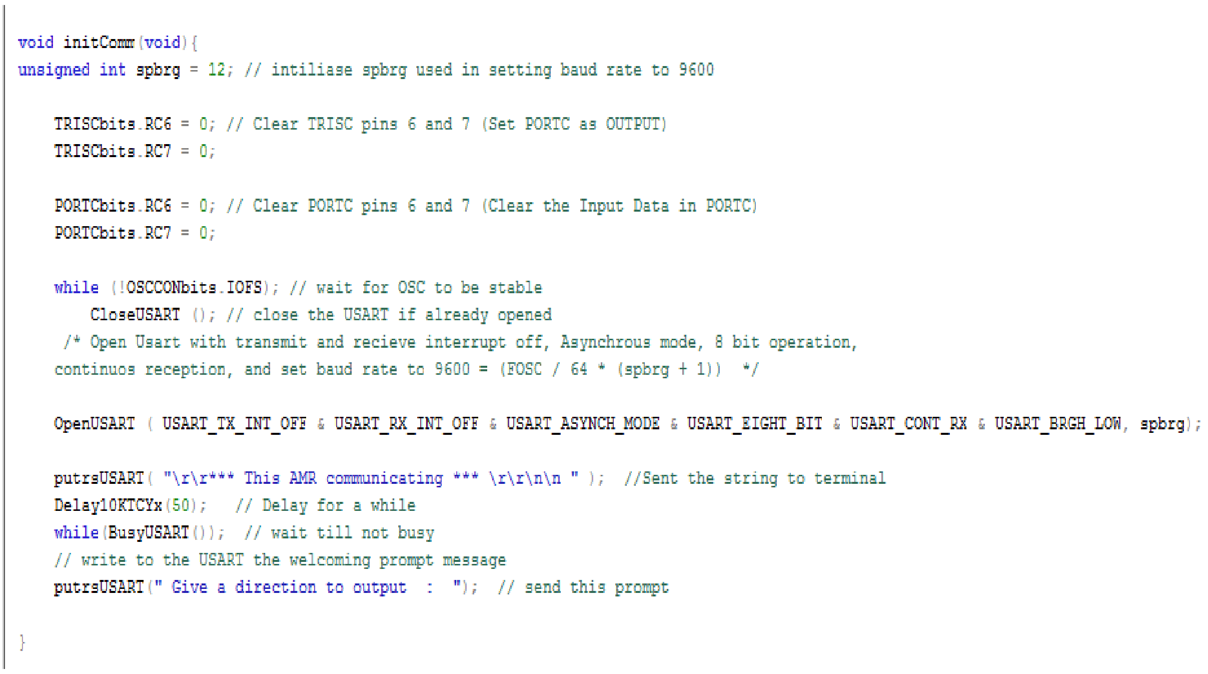
Robot Communication:

In the hardware implementation sub-chapter, it was mentioned that a router module is connected to the robot controller for receiving instructions from the PC controller interface. These is instructions are received via inbuilt USART instruction set for the MCC18 compiler of the microcontroller. It is worth mentioning that the micro controller has to be set to receive using the appropriate baud rates (bits/per second), the function "i*nitComm()*" was created for such a purpose as setting up the micro for USART communication (this setting only needs to be done once per program load).

This function:

- Initializes the port C pins 6 and 7 of the micro controller which are used for USART communication
- Reset these pins to 0 in case they set high (1) before,
- Wait for clock oscillation of micro to be stable
- Close the USART bus if previously opened

- Open the USART bus again and set to send and receive data at 9600 baud rate
- Send a message to PC controller saying the robot is read to receive communication



```
void initComm(void){
unsigned int spbrg = 12; // intiliase spbrg used in setting baud rate to 9600

    TRISCbits.RC6 = 0; // Clear TRISC pins 6 and 7 (Set PORTC as OUTPUT)
    TRISCbits.RC7 = 0;

    PORTCbits.RC6 = 0; // Clear PORTC pins 6 and 7 (Clear the Input Data in PORTC)
    PORTCbits.RC7 = 0;

    while (!OSCCONbits.IOFS); // wait for OSC to be stable
        CloseUSART (); // close the USART if already opened
     /* Open Usart with transmit and recieve interrupt off, Asynchrous mode, 8 bit operation,
    continuos reception, and set baud rate to 9600 = (FOSC / 64 * (spbrg + 1))  */

    OpenUSART ( USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE & USART_EIGHT_BIT & USART_CONT_RX & USART_BRGH_LOW, spbrg);

    putrsUSART( "\r\r*** This AMR communicating *** \r\r\n\n " );  //Sent the string to terminal
    Delay10KTCYx(50);   // Delay for a while
    while(BusyUSART());  // wait till not busy
    // write to the USART the welcoming prompt message
    putrsUSART(" Give a direction to output  :  ");  // send this prompt

}
```

Figure 43: USART communication initialisation function for the robot controller

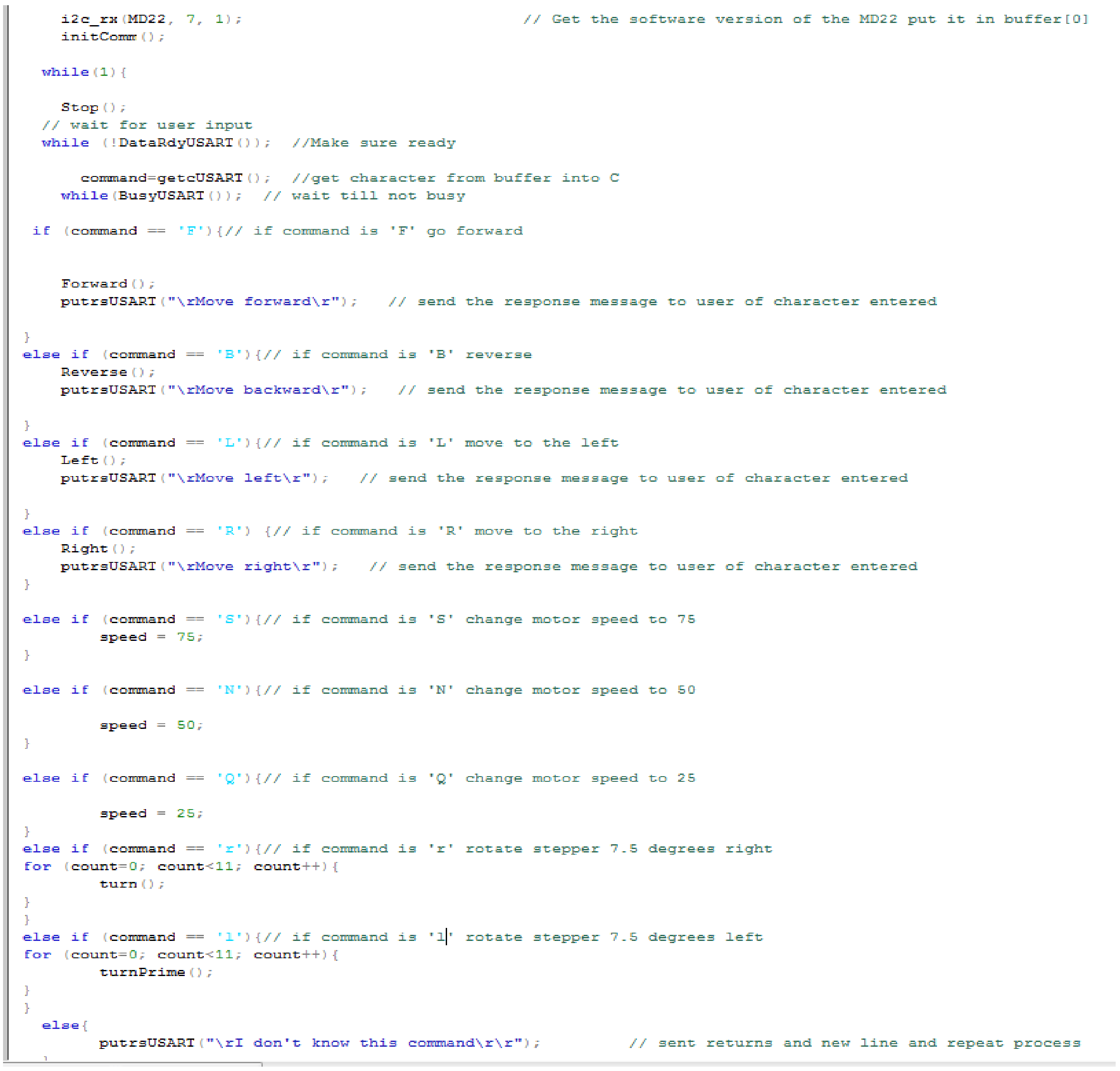
The main MCC18 USART functions used are:

- **OpenUSART()**
- **BusyUSART()**
- **DataReadyUSART()**
- **getcUSART()**
- **closeUSART()**

See Appendix D for full list of MCC18 USART functions and their meaning

Motor control and camera control steps are both acquired from the robot controller via USART "***getcUSART()***", when this function is called it gets a *char* value from the USART bus, this is then checked against predefined control variables in the robot controller using "*IF-statements*", see Chapter 5 for software locomotion process design.

Using these if statements which check for the *char* value grabbed from the USART bus, against certain predefined commands that call either the robot directional functions, set the robot speed or send cycle instructions for rotating the stepper motors.



```
    i2c_rx(MD22, 7, 1);                          // Get the software version of the MD22 put it in buffer[0]
    initComm();

  while(1){

    Stop();
  // wait for user input
  while (!DataRdyUSART());  //Make sure ready

      command=getcUSART();  //get character from buffer into C
    while(BusyUSART());  // wait till not busy

 if (command == 'F'){// if command is 'F' go forward


    Forward();
    putrsUSART("\rMove forward\r");   // send the response message to user of character entered

}
else if (command == 'B'){// if command is 'B' reverse
    Reverse();
    putrsUSART("\rMove backward\r");   // send the response message to user of character entered

}
else if (command == 'L'){// if command is 'L' move to the left
    Left();
    putrsUSART("\rMove left\r");   // send the response message to user of character entered

}
else if (command == 'R') {// if command is 'R' move to the right
    Right();
    putrsUSART("\rMove right\r");   // send the response message to user of character entered
}

else if (command == 'S'){// if command is 'S' change motor speed to 75
        speed = 75;
}

else if (command == 'N'){// if command is 'N' change motor speed to 50

        speed = 50;
}

else if (command == 'Q'){// if command is 'Q' change motor speed to 25

        speed = 25;
}
else if (command == 'r'){// if command is 'r' rotate stepper 7.5 degrees right
for (count=0; count<11; count++){
        turn();
}
}
else if (command == 'l'){// if command is 'l' rotate stepper 7.5 degrees left
for (count=0; count<11; count++){
        turnPrime();
}
}
  else{
        putrsUSART("\rI don't know this command\r\r");         // sent returns and new line and repeat process
```

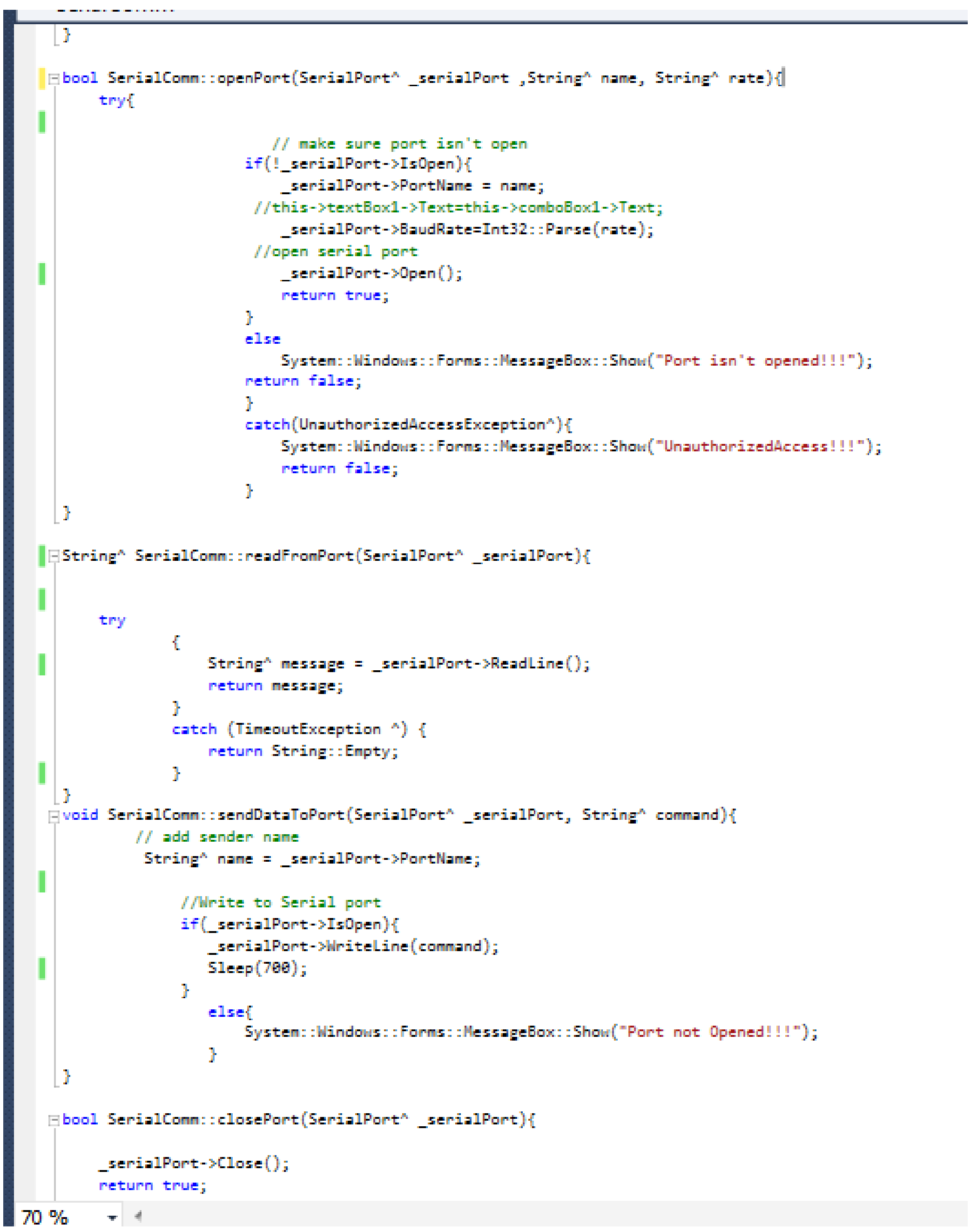
Figure 44: IF- statement loop code extract for motor and stepper control

PC controller communication:

A zigbee module is also connected to the controlling PC via a USB port, the PC uses serial communication to convey messages to the module to broadcast to the receiver router module, as seen in the Chapter 5, serial communication is a model class that was written from scratch, it comprises of several functions essential for serial communication such as;

- **openPort()**: this function takes 3 arguments; the serial Port to be opened, the COM port name and the baud rate for transmission (the latter two are chosen from a drop down menu). This function configures the settings for opening a serial COM port and the opens the port if previously unopened.

- **readFromPort()** : This function only takes a single argument which is the serial port being used, it reads from the serial COM buffer and inputs the contents of the buffer into the a string and returns said string
- **sendDataToPort():** This function takes two arguments; the serial port to communicate with and the command to be sent to that port. If the port is opened then data can be sent, else a windows form message box appears to warn the user that no port has been opened.
- **closePort():** This function also takes a single argument, this is the serial port involved, and this function is simply to close the serial port and returns a Boolean value.

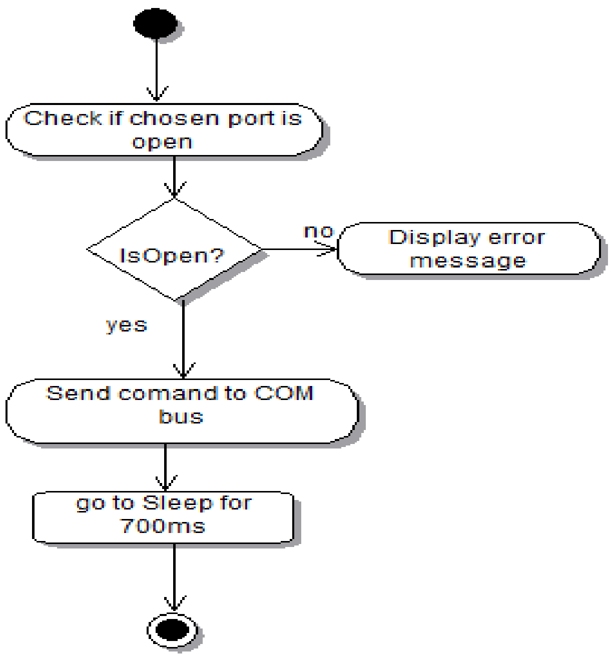


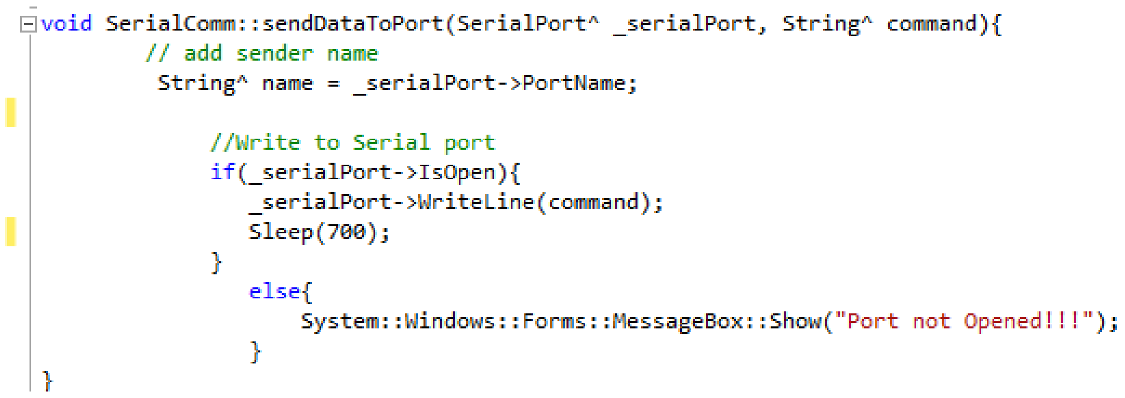
```
}

bool SerialComm::openPort(SerialPort^ _serialPort ,String^ name, String^ rate){
    try{

                        // make sure port isn't open
                    if(!_serialPort->IsOpen){
                        _serialPort->PortName = name;
                     //this->textBox1->Text=this->comboBox1->Text;
                        _serialPort->BaudRate=Int32::Parse(rate);
                     //open serial port
                        _serialPort->Open();
                        return true;
                    }
                    else
                        System::Windows::Forms::MessageBox::Show("Port isn't opened!!!");
                    return false;
                    }
                    catch(UnauthorizedAccessException^){
                        System::Windows::Forms::MessageBox::Show("UnauthorizedAccess!!!");
                        return false;
                    }
}

String^ SerialComm::readFromPort(SerialPort^ _serialPort){

    try
            {
                String^ message = _serialPort->ReadLine();
                return message;
            }
            catch (TimeoutException ^) {
                return String::Empty;
            }
}
void SerialComm::sendDataToPort(SerialPort^ _serialPort, String^ command){
        // add sender name
         String^ name = _serialPort->PortName;

             //Write to Serial port
             if(_serialPort->IsOpen){
                _serialPort->WriteLine(command);
                Sleep(700);
             }
                else{
                    System::Windows::Forms::MessageBox::Show("Port not Opened!!!");
                }
}

bool SerialComm::closePort(SerialPort^ _serialPort){

    _serialPort->Close();
    return true;
```

70 %

Figure 45: SerialComm class function set as described above.

During implementation testing, it was discovered that sending continuous stream of data to the router USART bus can cause the bus to overflow and lock up, to prevent this, for each command sent to the serial port a 700ms second delay will flow, this time frame was found most appropriate threshold for preventing the serial communication line from seizing up as well as allowing the system to be more than fairly responsive to user command. Visual studio allows for easy interfacing of the COM port of a PC, as it has a GUI component which inherits from the System I/O port. The general implemented routine for communicating commands to the serial port for wireless transfer to the robot is shown in the figure below.





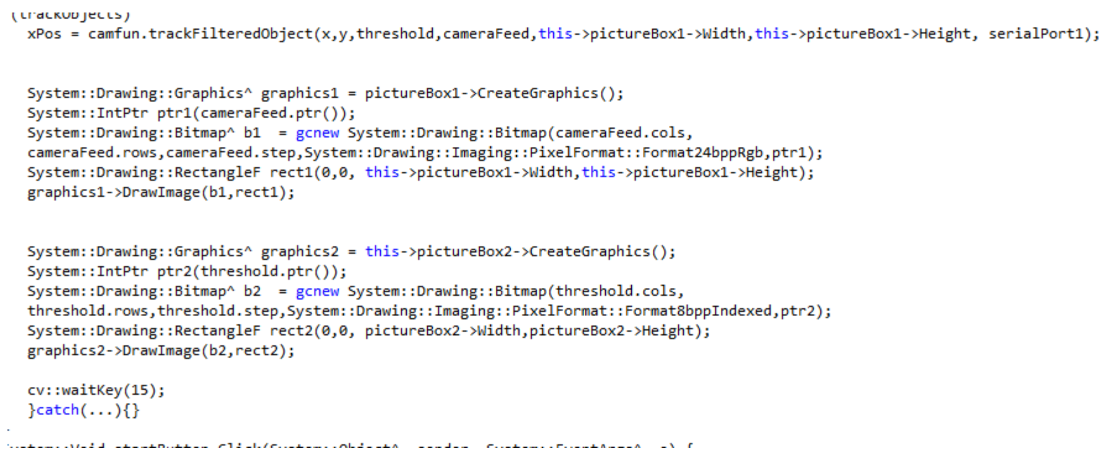
```
void SerialComm::sendDataToPort(SerialPort^ _serialPort, String^ command){
        // add sender name
         String^ name = _serialPort->PortName;

             //Write to Serial port
             if(_serialPort->IsOpen){
                _serialPort->WriteLine(command);
                Sleep(700);
             }
                else{
                    System::Windows::Forms::MessageBox::Show("Port not Opened!!!");
                }
}
```

Figure 46: Sending command routine process and code extract

6.4 Interface

The interface was developed using MS visual studio 2010 *windows form application* package (which is a drag and drop interface development package, where GUI icons can easily be placed on a pre-generated but resizable Windows frame and Visual studio automatically generates codes for the GUI icons), and serves as the *view* for the system controller, there were several iterations of the interface developed (see Appendix C below), each iteration tries to capture some the requirements of the system and evolves as requirements become more apparent. The interface allows for the following;

- **Port search and selection** – through a button press events and drop down menu
- **Baud rate selection (2 of the most popular choices for serial communication)** - using a drop down menu, the user can select a baud rate of their choosing.
- **Opening and closing of selected port** – these are also so button press events, that call on the openPort() and closePort() function of the SerialComm class.
- **Real-time speed change (3 fixed speed levels – slow, normal and fast)** - These are radio buttons by selecting on the speed choices, the speed of the motor is set.
- **Starting of camera feed and pausing**. – this is a button press event that starts a timer, within this timer function, functions for grabbing the video stream from the web cam as well as processing of these image streams (morph) are called there.
- **User defined in real time video processing from the camera feed -** using windows form track bars, the user can set the HSV values of a frame at any given time, which sets the minimum and maximum values of the HSV scale in the InRange() open cv function, also the sensitivity of the
- **Reading from serial port** (not perfected yet)
- **Sending motor direction to the robot remotely** – this is a button click event that simply calls on the sendDataToPort() function of the SerialComm class.
- **Show images (original and threshold), identifying object currently being tracked** – the images are shown on a windows form picture box, picture boxes in Windows form only show bitmap type with an RGB colour scale this means that there is need to convert the images from the webcam which uses BGR colour scale (normal for all webcams), from an image matrix to bitmap and then switching the colour scale. The code extract to do so can be view below



```
(trackObjects)
  xPos = camfun.trackFilteredObject(x,y,threshold,cameraFeed,this->pictureBox1->Width,this->pictureBox1->Height, serialPort1);


  System::Drawing::Graphics^ graphics1 = pictureBox1->CreateGraphics();
  System::IntPtr ptr1(cameraFeed.ptr());
  System::Drawing::Bitmap^ b1  = gcnew System::Drawing::Bitmap(cameraFeed.cols,
  cameraFeed.rows,cameraFeed.step,System::Drawing::Imaging::PixelFormat::Format24bppRgb,ptr1);
  System::Drawing::RectangleF rect1(0,0, this->pictureBox1->Width,this->pictureBox1->Height);
  graphics1->DrawImage(b1,rect1);


  System::Drawing::Graphics^ graphics2 = this->pictureBox2->CreateGraphics();
  System::IntPtr ptr2(threshold.ptr());
  System::Drawing::Bitmap^ b2  = gcnew System::Drawing::Bitmap(threshold.cols,
  threshold.rows,threshold.step,System::Drawing::Imaging::PixelFormat::Format8bppIndexed,ptr2);
  System::Drawing::RectangleF rect2(0,0, pictureBox2->Width,pictureBox2->Height);
  graphics2->DrawImage(b2,rect2);

  cv::waitKey(15);
  }catch(...){}
```

Figure 47: Code extract for image matrix to bitmap image to be displayed on picture box

- **Cancel the GUI** - This is a simple button click even that closes the GUI.

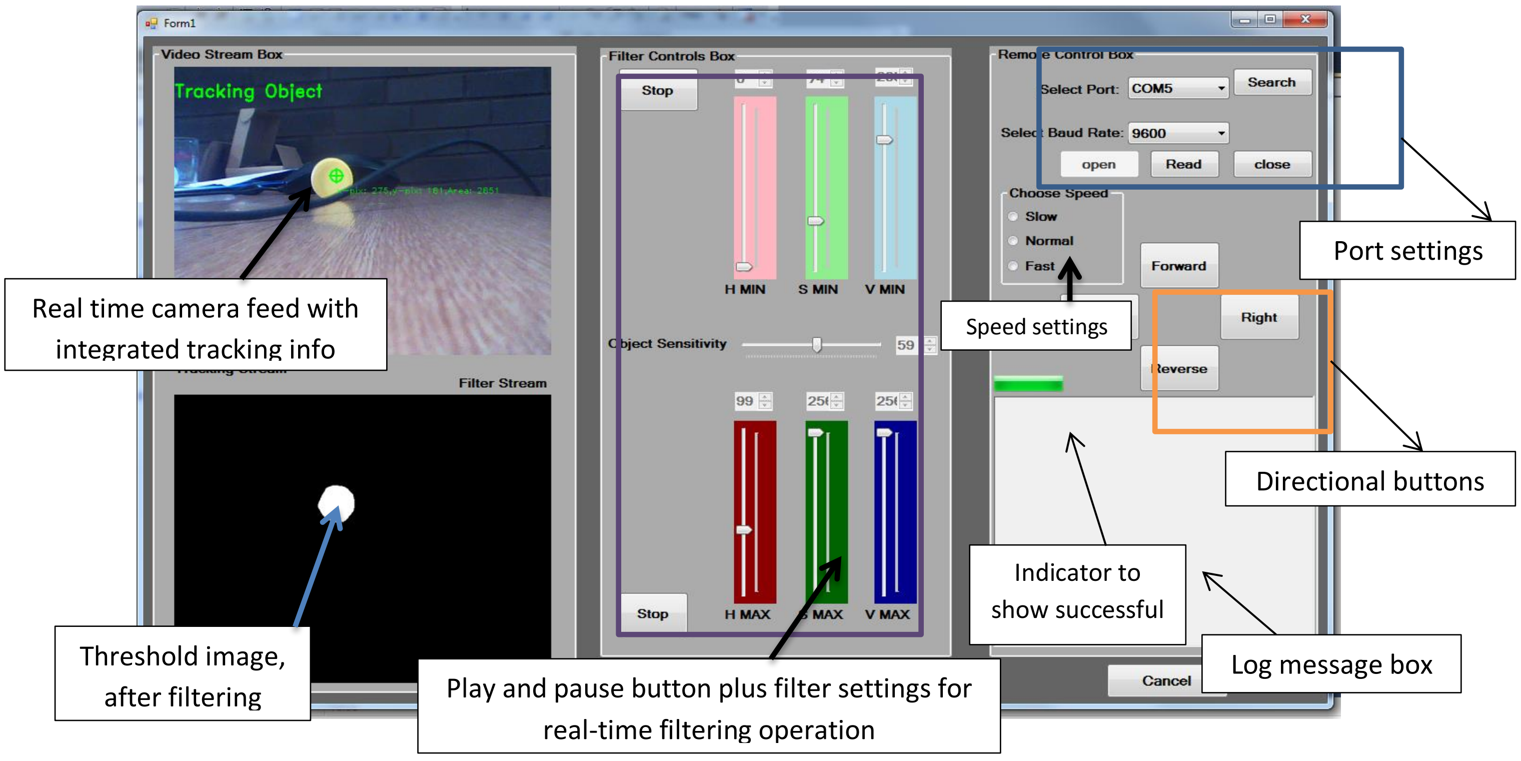


Figure 48: Final Graphic user interface implemented

6.5 Overall Implementation Architecture

To complete the implementation, both the software aspects and the hardware aspects must be interfaced together, the architecture developed from this integration resembles the figure seen below.

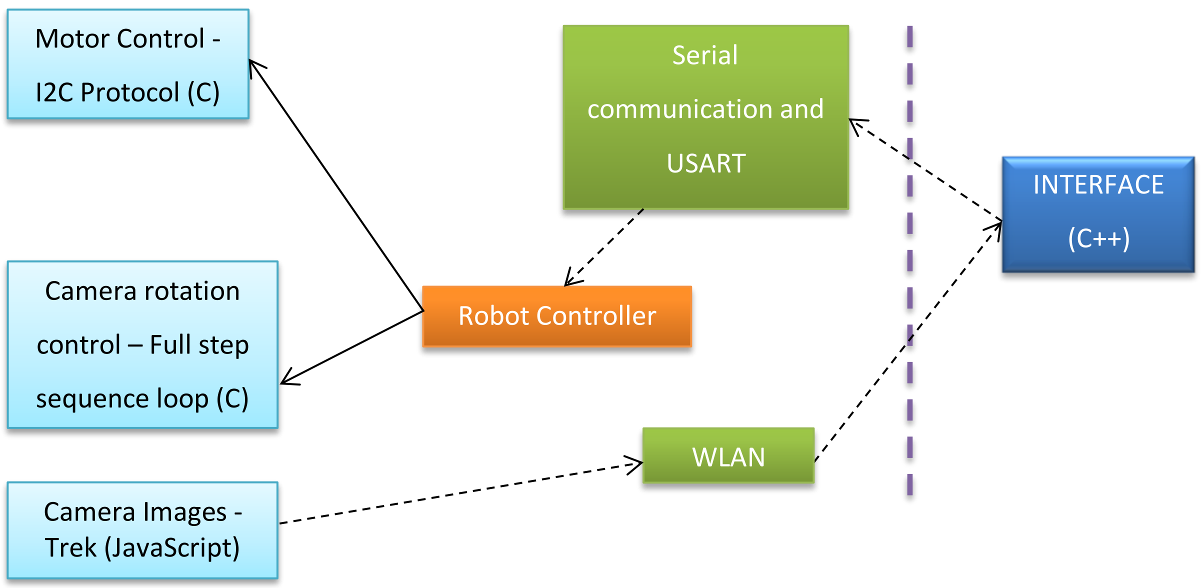

Motor Control - I2C Protocol (C)
Serial communication and USART
INTERFACE (C++)
Camera rotation control – Full step sequence loop (C)
Robot Controller
WLAN
Camera Images - Trek (JavaScript)

Figure 49: Implementation architecture of the entire system

7 Summary and Conclusion

7.1 Critical Appraisal

Evaluation of the system implementation cannot be explicitly explained, without divulging too much into Electrical and electronics engineering. Some software testing was done to evaluate the performance of the GUI when running full functionalities. The key achievements evaluated from these tests as well as other testing can be seen below.

At the time of this report conclusion, by evaluating the progress on design and implementation for a mobile robot system, based on the objects (see Chapter 1), despite certain draw backs, there have been some key achievements, as well acquired knowledge on robotics as a computing paradigm.

7.1.1 Achievements

- **Extensive Research on Topic Areas:**

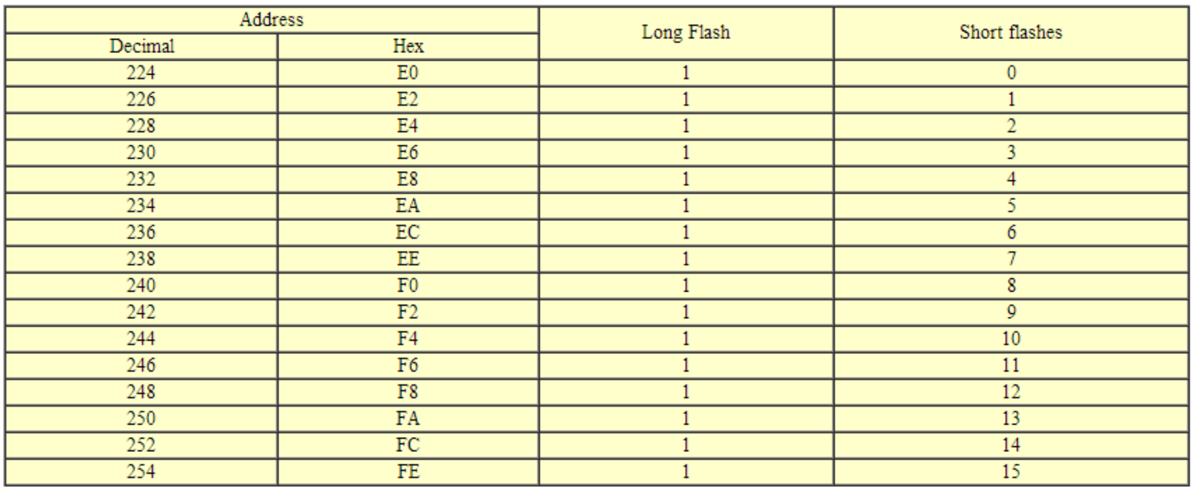
  Research conducted on the various topic areas including; Computing in mobile robotics, Motion planning and target tracking were adequate enough to serve as guides for developing and implementing the project system, highlighting the constraints and variables to consider when developing mobile robotic systems and designing autonomy for such systems.

- **Component Identification:**
  Components, both hardware and software systems, for the current system, were adequately identified and were able to perform most of the functionalities – including; Opencv C++ for computer vision, the PIC micro controller as a low powered robot control with extensibility, the wireless mini web cam, the RF module etc. - of the requirements save some that require addition components like, stereo vision camera set, an accelerometer and/or gyroscope for odometry purposes to better enable motion planning,.
- **Motion Planning/ Navigation design:**
  An obstacle navigation design was drawn up by the end of the project schedule that will enable the use of the ultrasonic sensors ranging prowess to avoid obstacles in a path and also accounts for the beam spread weakness from these sensor when detecting objects in near proximity. Unfortunately due to some draw backs and a tight schedule, this design could not be implemented.

- **Automation:**
  The developed system by the end was able to track a target autonomously, but also had user-enabled full range motion control of the robotic system as well as some level of speed control as well.
- **Tracking System:**
  A tracking subsystem that autonomously tracks a target's translational displacement was successfully implemented using the coordinates from grabbed video streams from the webcam attached to a stepper motor for rotation.
- **Communication subsystem:**
  A subsystem that allows for private communication between the robot and the controlling PC was also successfully implemented using RF modules.
- **Graphical User Interface:**
  A user-friendly interface was created that allows for portability and user freedom in accomplishing most of the tasks listed in the requirements.
- **Interfacing implemented functionality:**
  By the end of this projected all functionalities implemented, were successfully interfaced together to create a mobile robotic system, that can be remotely controlled by the user, while its tracking subsystem autonomously tracks a user defined target.

7.1.2 Drawbacks

- A major drawback to this project was that autonomous motion planning algorithms were not implemented, this was majorly due to lack of appropriate odometry devices that could provide localisation of the goal, obstacles and even the robot its relative to the robotic system in a space constraint.
- Also adding to the drawbacks for this project was the previously mismanaged motor and sensor devices, as well as no prior information on some of the components previously used. First the circuitry connections to the previous devices for the older version of this system was wrong, despite identifying this earlier on, it was said otherwise by the previous project student and the technicians, this greatly reduced the time frame and slowed down development processes, where majority of the time spent was on relentless and unnecessary component testing! It was recently discovered that the ultrasonic ranging devices had their addresses wrongly assigned to them, and there was no time to re-address and test these components. The correct possible addresses and how to identify them for these sensors can be seen below, simply by powering each sensor and observing the flash sequence the right addresses for these sensors can be known, but they were labelled wrongly by previous students:



Address		Long Flash	Short flashes
Decimal	Hex		
224	E0	1	0
226	E2	1	1
228	E4	1	2
230	E6	1	3
232	E8	1	4
234	EA	1	5
236	EC	1	6
238	EE	1	7
240	F0	1	8
242	F2	1	9
244	F4	1	10
246	F6	1	11
248	F8	1	12
250	FA	1	13
252	FC	1	14
254	FE	1	15

Table 10: Correct addresses and how to identify them (Robot Electronics 2014)

- Another of the drawbacks was that some of the components purchased were faulty and new ones had to be re-ordered at the developer's cost, these include:
  - The Xbee programming board
  - The battery charger adaptor
  - Two new Zigbee modules

  - The wireless webcam
  - An additional ultrasonic detector (That was how the addressing errors, as well as the circuitry errors were discovered).

7.1.3 Future Works

- Future works include using triangulation techniques for depth perception to acquire real world space position of targets. Depth perception can help create an attraction potential field algorithm for better navigation techniques.
- The use of an accelerometer and other odometric devices, can be interfaced with the robot for localisation capabilities, that also help with motion planning.

7.2 Summary

This report discusses the development process – both in system and software engineering -, and designs implemented to create a mobile robotic system. With the research done on computation facilitation in robotics as well as on robot vision and motion planning techniques - where constraints apply when path planning for the two types of holonomy in robotics (holonomous and non-holonomous), also on how to identify and distinguish them. The technologies and architectures that have been used to facilitate the functionalities similar to those for this project in related systems. It discusses the mechanical and electric constraints considered into other to facilitate autonomy in the overall system and its subsystem, its states the draw backs during developments and also give future recommendations for further improving the system.

8 References

- NASA. (2014). *Mars Science Laboratory - Curiosity | NASA*. [ONLINE] Available at:http://www.nasa.gov/mission_pages/msl/#.U_fxefldV8E.
- RobotShop. (2008). History of Robotics: Timeline. *ROBOTSHOP*, 1, 5. Available at:http://www.robotshop.com/media/files/PDF/timeline.pdf
- Thomasnet. (2014). *History of Robotics*. [ONLINE] Available at: http://www.thomasnet.com/articles/engineering-consulting/robotics-history.
- G. Dudek, M. Jenkin. (2010). *Computational Principles of Mobile Robotics*. 2 Edition. Cambridge University Press.
- J.H Reif, H. Wang. (1999). Social potential fields: A distributed behavioral control for autonomous robots. *Robotic and Autonomous Systems*, 4, p171-194.
- R.G, Simmons. (1994). Structured Control for Autonomous Robot. *IEEE JOURNAL OF ROBOTICS AND AUTOMATION*, 10, p34-36.
- T. Livneh, (2006). The "Piano Movers Problem". *Vision and Robotics Workshop*, 1, pg4.
- R, Murray. (1993). Nonholonomic Motion Planning: Steering Using Sinusoids. *IEEE TRANSACTIONS ON AUTOMATIC CONTRO*, 38, p1
- J, Lengyel et al. (2006). Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware. pg1, 1,2,3,8.
- J.A, Batlle, and A, Barjau, (2009). Holonomy in mobile robots. *Robotics and Autonomous Systems*, 57, p433-435.
- R.A, Brooks. (1986). A Robust Layered Control Syste For A Mobile Robot. *IEEE JOURNAL OF ROBOTICS AND AUTOMATION*, 2, p14 -15.
- H.G, Tanner, (2003). Nonholonomic Navigation and Control of Cooperating Mobile Manipulators. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, 19, p53.
- E.I, AKANGSON. (2013). AUTOMATIC BATCH IMAGE PROCESSING. *COVENTRY UNIVERSITY Faculty of Engineering and Computing Department of Aerospace, Electrical and Electronic Engineering*, 1, 8.
- H. Choset. (2014). Robotic Motion Planning: Potential Functions. 1, p8.
- W. Burger, J. M. Burge. (2009). Principles of Digital Image Processing: Fundamental Techniques. London: Springer-Verlag London Limited. 26, p51.

- C. Solomon, T.Breckon . (2011). *Fundamentals of Digital Image Processing*, A Practical Approach with Examples in MATLAB. West Sussex, UK: Wiley Blackwell . p6-7, p11-14.
- McGill University. (2013). *IT Project Management.* Available: http://www.mcgill.ca/it/it-projects/project-management.
- Blanchard B. S. (2003). *System Engineering Management*. 3 Edition. Wiley
- Robot Electronics. (2014). *I2C Tutorial.* Available at: http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html.
- 2014. *How to Interface I2C-EEPROM with 8051*. [ONLINE] Available at: https://www.pantechsolutions.net/microcontroller-boards/i2c-eeprom-interfacing-with-8051-slicker].
- MicroChip. (2014). *PIC18F4520 - 8-bit PIC® Microcontrollers*. Available at: http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010297.
- Trek (2014). *Ai-Ball Specs* - Trek . Available at: http://www.thumbdrive.com/aiball/specs.html.
- 2014. *MD22 Technical Documentation.* Available at: http://www.robot-electronics.co.uk/htm/md22tech.htm.
- DIGI. (2012). XBee®/XBee-PRO® ZB RF Modules. *ZigBee RF Modules by Digi International*,. 90000976_K, 9. Available at: http://www.adafruit.com/datasheets/XBee%20ZB%20User%20Manual.pdf
- MicroChip. (2014). *PICkit 3 In-Circuit Debugger* - PG164130 | Microchip Technology Inc. Available at: http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=PG164130
- Robot Electronic. (2012). *SRF08 Ultra sonic range finder*. Available at: http://www.robot-electronics.co.uk/htm/srf08tech.shtml.
- National Instruments. (2014). *3D Imaging with NI LabVIEW* - National Instruments. Available at:http://www.ni.com/white-paper/14103/en/.
- D. DadaChanji. (2006). *File:Sonar Principle EN.svg* - New World Encyclopedia. Available at:http://www.newworldencyclopedia.org/entry/File:Sonar_Principle_EN.svg.
- AdaFruit (2014). *Small Reduction Stepper Motor - 12VDC 32-Step 1/16 Gearing ID: 918 - $4.95:* Adafruit Industries, Unique & fun DIY electronics and kits. Available at: http://www.adafruit.com/products/918.

- OpenCV 2.4.9.0 documentation. (2014).*Basic Thresholding Operations* — OpenCV 2.4.9.0 documentation. Available at: http://docs.opencv.org/doc/tutorials/imgproc/threshold/threshold.html.
- BWeeter (2012). *AP186 Basic Video Processing* - bweeter | The greatest WordPress.com site in all the land!. Available at:http://bweeter.wordpress.com/.
- Ober (2012). *XBee Series 2 Point to Point Communication* - Tutorial by Cytron. Available at: http://tutorial.cytron.com.my/2012/03/08/xbee-series-2-point-to-point-communication/.
- N. Patel (2009). *Pygame Tutorials* - Camera Module Introduction. Available at:http://www.pygame.org/docs/tut/camera/CameraIntro.html.

9 Appendices:

9.1 Appendix A - User Manual

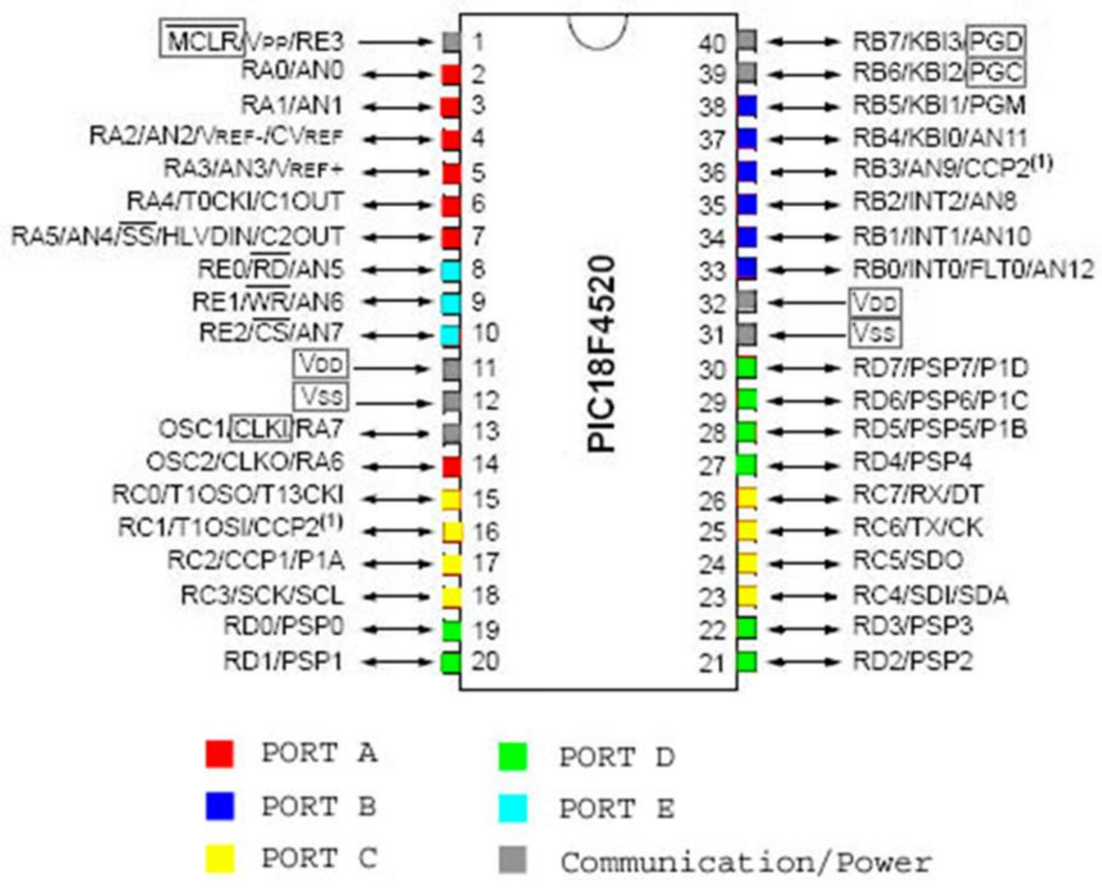
This appendix gives instructions on how to run the robot system.

- Turn on the switch on the robot.
- Turn on the attached webcam
- Run the AMR_interface C++ Program
- Click on the "Search Port button", this fills the drop down menu with available COM ports on the system
- Select the COM in which the Zigbee module is connected to
- Click on the open button to open port – if port is opened successfully indicator bar will go green, else a message box will pop up giving message
- Select a motor speed from three choices – The motor is naturally set to off, until speed is chosen.
- Click on the start button to start acquiring video feed from camera – this will also enable the sliders for adjusting the HSV and noise levels of the feed. If no camera images (e.g. camera no turned on) are grabbed a Message box with an error message will pop up.
- Adjust the HSV values accordingly to filter video stream images, also adjust the sensitivity setting to filter out colour noise
- When image filtering is done to user satisfaction, click on the track button to begin tracking the filtered object.
- The user can also use the directional button meanwhile to send directional commands to the robot system.
- When done with the interface, click on the "Stop" button, then the close button to close the serial port communication, then click on the cancel button to close the frame.

9.2 Appendix B – Software Packages used

- MPLSB IDE v8.90
- Microsoft Visual Studio 2010
- X-CTU
- Opencv 2.4.8

9.3 Appendix C - Pin Diagram of PIC and other features



Features	PIC18F4520
Operating Frequency	DC – 40 MHz
Program Memory (Bytes)	32768
Program Memory (Instructions)	16384
Data Memory (Bytes)	1536
Data EEPROM Memory (Bytes)	256
Interrupt Sources	20
I/O Ports	Ports A, B, C, D, E
Timers	4
Capture/Compare/PWM Modules	1
Enhanced	1

Capture/Compare/PWM Modules	
Serial Communications MSSP	Enhanced USART MSSP
Parallel Communications (PSP)	Yes
10-Bit Analog-to-Digital Module	13 Input Channels
Resets (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST), MCLR (optional), WDT Programmable
High/Low-Voltage Detect	Yes
Programmable Brown-out Reset	Yes
Instruction Set	75 Instructions; 83 with Extended Instruction Set Enabled

9.4 Appendix D - User Interface Iterations

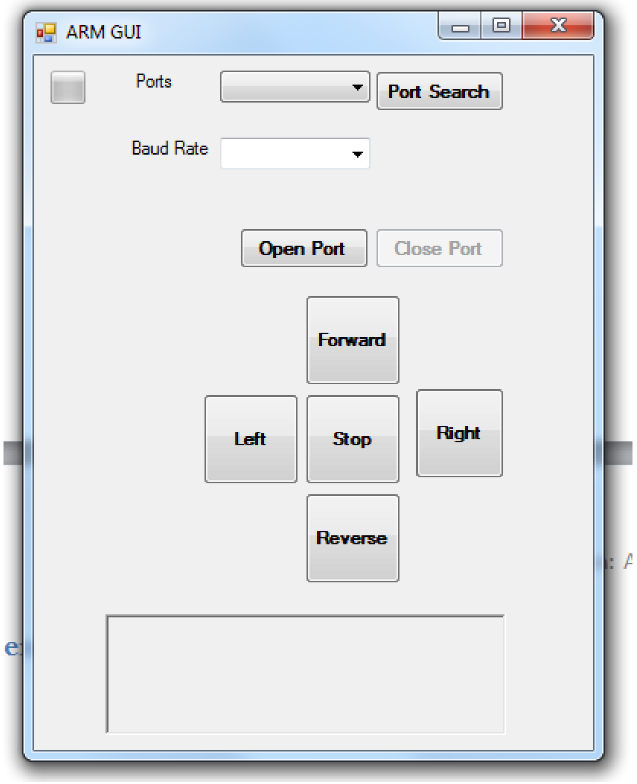


Figure 50: UI version one

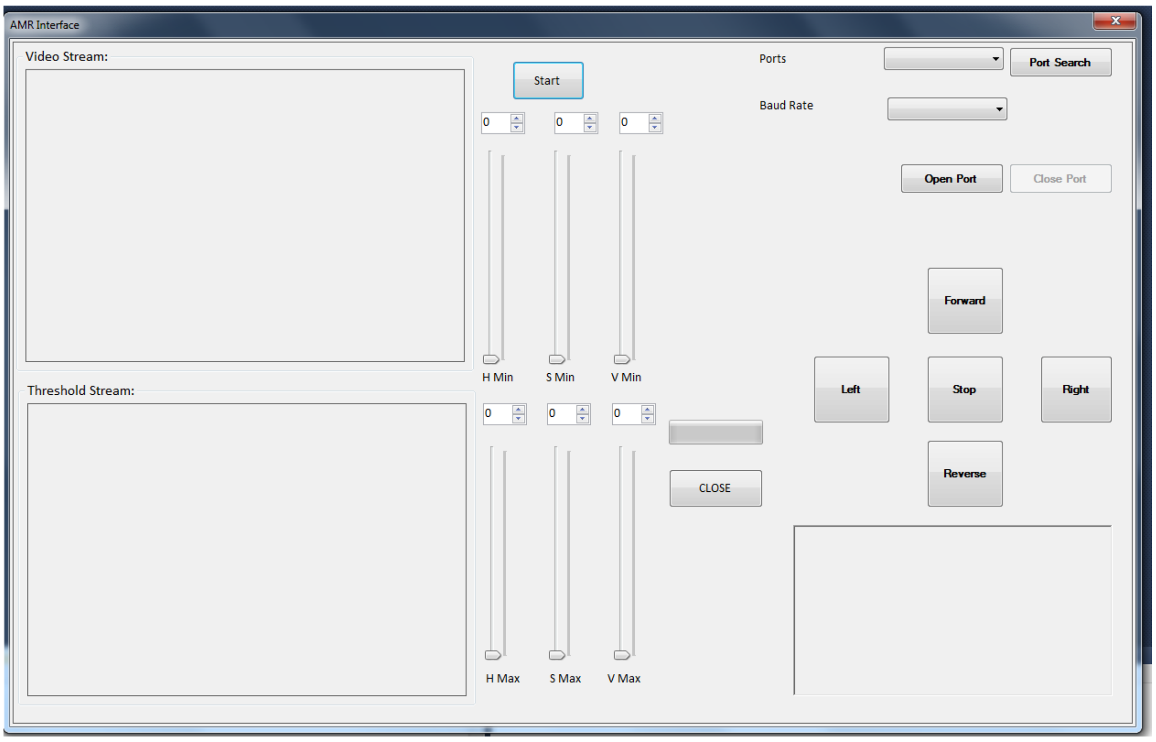


Figure 51: UI version 2

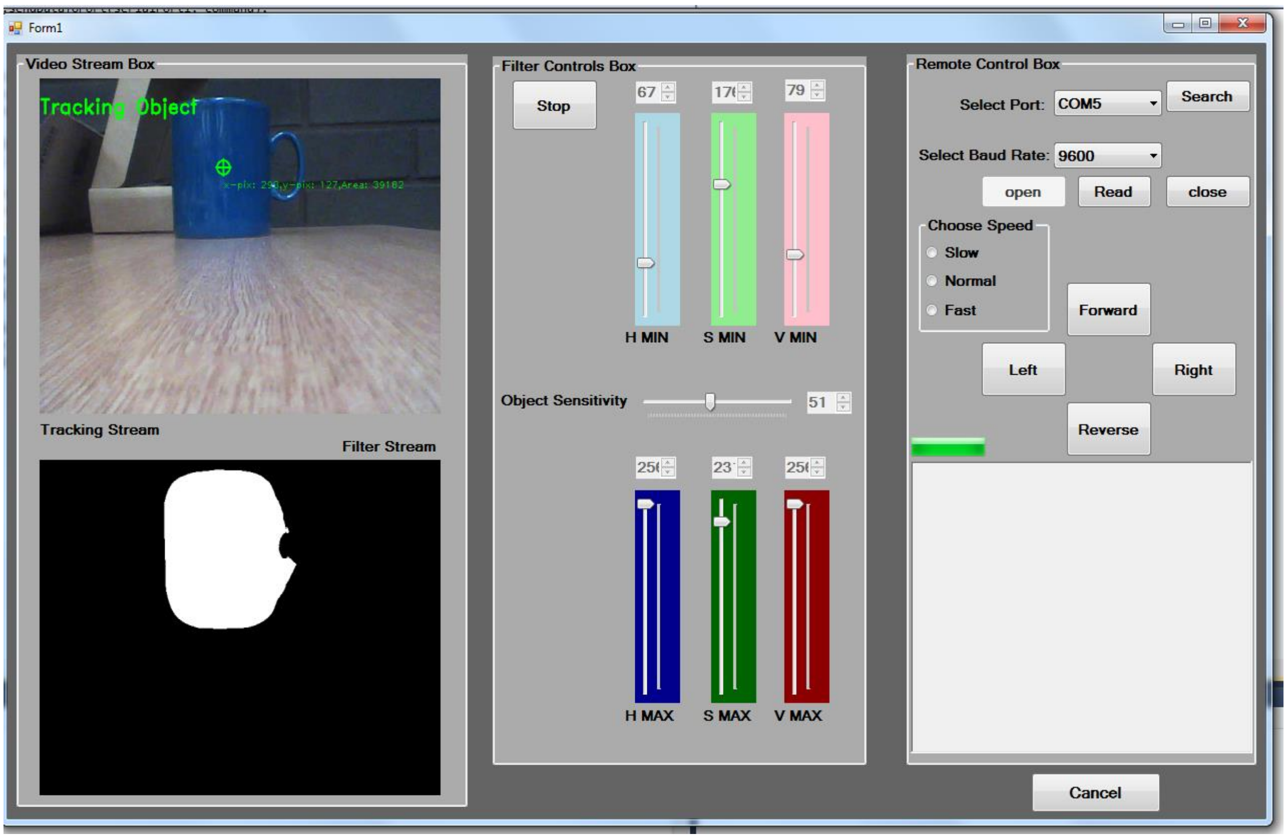


Figure 52: UI version 3

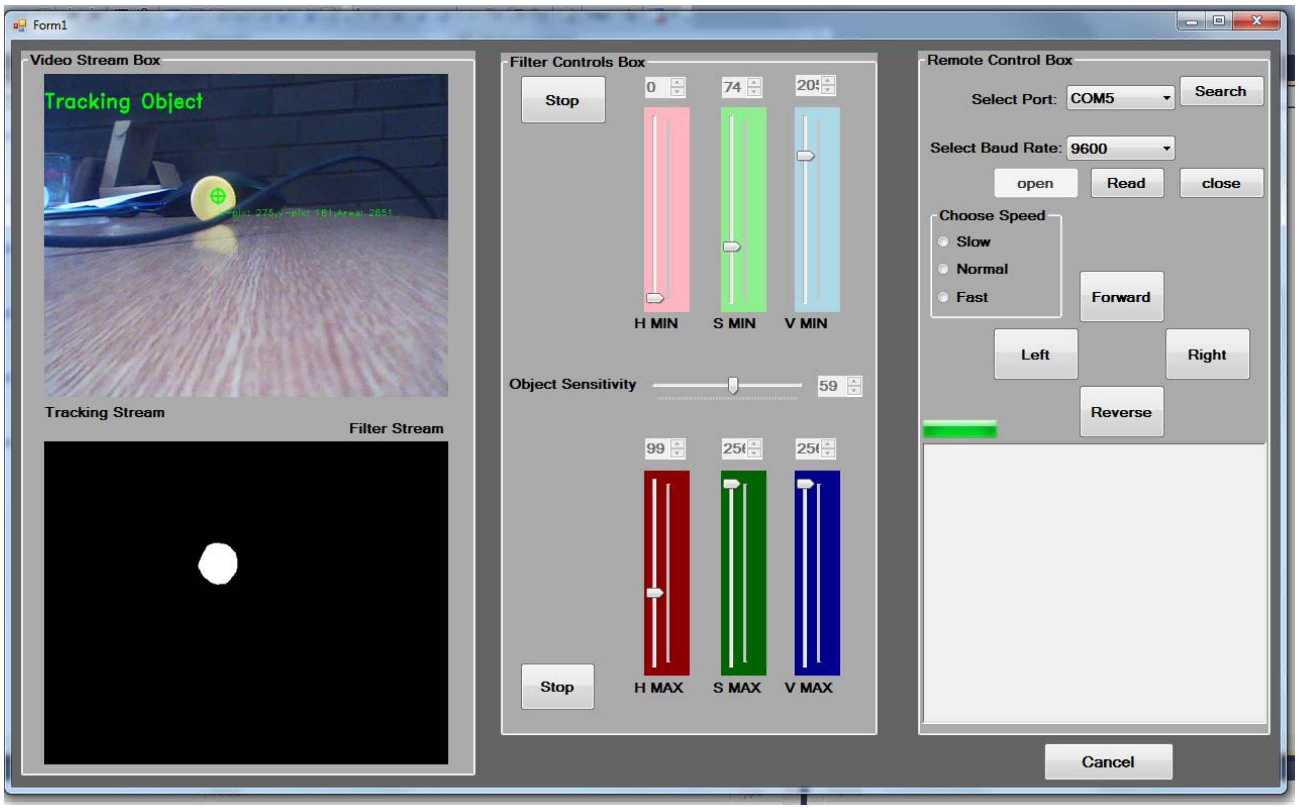


Figure 53: UI final version

9.5 Appendix E - USART Functions

USART FUNCTIONS

The information in this document is obtained from the following Microchip manuals:

- PIC18Fxx2 Datasheet
- PICmicro® 18C MCU Family Reference Manual
- MPLAB C18 C Compiler Libraries

Function Prototypes:
For a detailed description of these functions, please see:
Section 2.10 USART Functions, in MPLAB C18 C Compiler Libraries manual.

```
#include <usart.h>

void OpenUSART    ( unsigned char config, unsigned int SPBRG);         // Configure the USART.
----------
char BusyUSART    ( void );                                             // Is the USART transmitting?

void WriteUSART   ( char data );            // Write a byte (one character) to the USART transmit buffer.
void putcUSART    ( char data );                                  // putcUSART is defined as WriteUSART.

void putsUSART    ( char *data );
                    // Write a string from data memory to the USART, including the null character.
void putrsUSART   ( const rom char *data );
                  // Write a string from program memory to the USART, including the null character.
----------
char DataRdyUSART( void );                                     // Is data available in the read buffer?

char ReadUSART    ( void );                // Read a byte (one character) out of the USART receive buffer.
char getcUSART    ( void );                                      // getcUSART is defined as ReadUSART.

void getsUSART    ( char * buffer, unsigned char len );
                                        // Read a fixed-length string of characters from the USART.
----------
void CloseUSART   ( void );                                                       // Disable the USART.
```

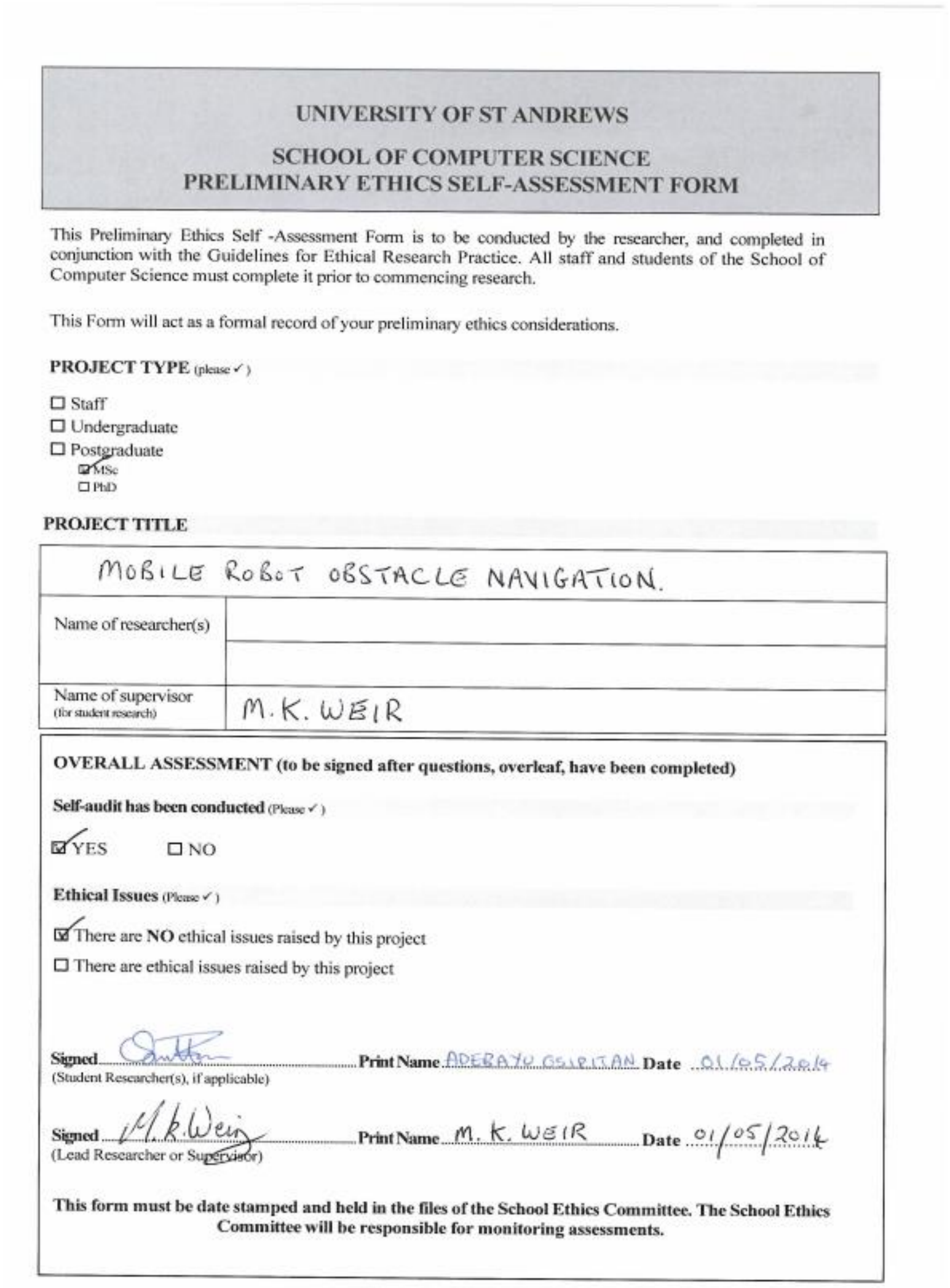
Notes:

1. The three most useful functions in the above list are `OpenUSART`, `getsUSART` and `CloseUSART`.

2. The `getsUSART` function waits for and reads `len` number of characters out of the USART. There is no time out when waiting for characters to arrive.

3. For serial output, the preferred functions are `putc(…)`, `puts(…)`, `printf(…)` etc., which are listed in the **Character Output Functions** document.

9.6 Appendix F - Ethics



UNIVERSITY OF ST ANDREWS

SCHOOL OF COMPUTER SCIENCE
PRELIMINARY ETHICS SELF-ASSESSMENT FORM

This Preliminary Ethics Self -Assessment Form is to be conducted by the researcher, and completed in conjunction with the Guidelines for Ethical Research Practice. All staff and students of the School of Computer Science must complete it prior to commencing research.

This Form will act as a formal record of your preliminary ethics considerations.

PROJECT TYPE (please ✓)

☐ Staff
☐ Undergraduate
☐ Postgraduate
  [✓ MSc]
  ☐ PhD

PROJECT TITLE

MOBILE ROBOT OBSTACLE NAVIGATION.	
Name of researcher(s)	
Name of supervisor (for student research)	M.K. WEIR

OVERALL ASSESSMENT (to be signed after questions, overleaf, have been completed)

Self-audit has been conducted (Please ✓)

[✓ YES] ☐ NO

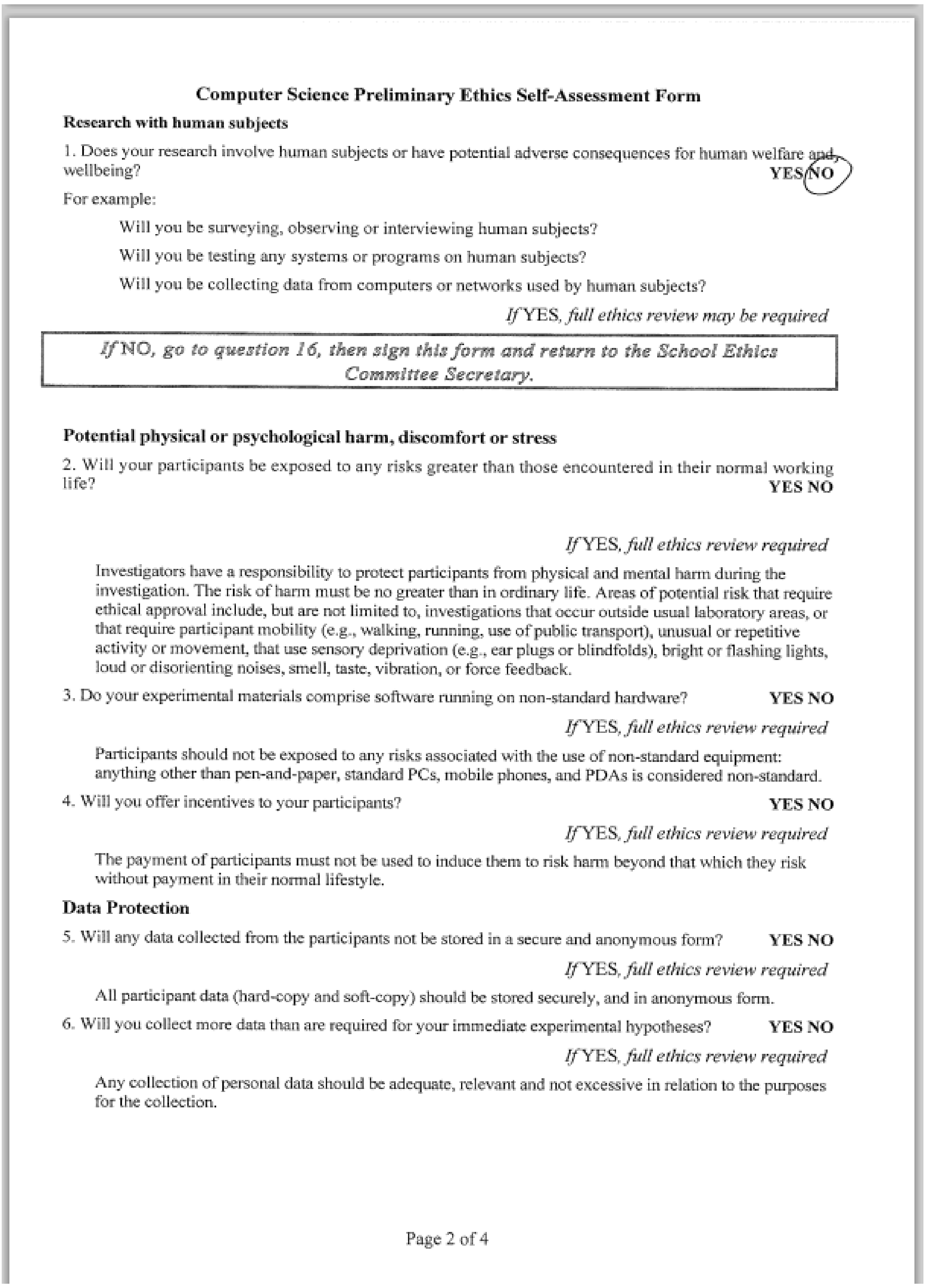
Ethical Issues (Please ✓)

[✓ There are **NO** ethical issues raised by this project]
☐ There are ethical issues raised by this project

Signed [illegible] Print Name ADEBAYO OSIPITAN Date 01/05/2014
(Student Researcher(s), if applicable)

Signed M.K.Weir Print Name M. K. WEIR Date 01/05/2014
(Lead Researcher or Supervisor)

This form must be date stamped and held in the files of the School Ethics Committee. The School Ethics Committee will be responsible for monitoring assessments.



Computer Science Preliminary Ethics Self-Assessment Form

Research with human subjects

1. Does your research involve human subjects or have potential adverse consequences for human welfare and wellbeing? **YES [✓ NO]**

For example:

Will you be surveying, observing or interviewing human subjects?

Will you be testing any systems or programs on human subjects?

Will you be collecting data from computers or networks used by human subjects?

If YES, *full ethics review may be required*

If NO, *go to question 16, then sign this form and return to the School Ethics Committee Secretary.*

Potential physical or psychological harm, discomfort or stress

2. Will your participants be exposed to any risks greater than those encountered in their normal working life? **YES NO**

If YES, *full ethics review required*

Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life. Areas of potential risk that require ethical approval include, but are not limited to, investigations that occur outside usual laboratory areas, or that require participant mobility (e.g., walking, running, use of public transport), unusual or repetitive activity or movement, that use sensory deprivation (e.g., ear plugs or blindfolds), bright or flashing lights, loud or disorienting noises, smell, taste, vibration, or force feedback.

3. Do your experimental materials comprise software running on non-standard hardware? **YES NO**

If YES, *full ethics review required*

Participants should not be exposed to any risks associated with the use of non-standard equipment: anything other than pen-and-paper, standard PCs, mobile phones, and PDAs is considered non-standard.

4. Will you offer incentives to your participants? **YES NO**

If YES, *full ethics review required*

The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

Data Protection

5. Will any data collected from the participants not be stored in a secure and anonymous form? **YES NO**

If YES, *full ethics review required*

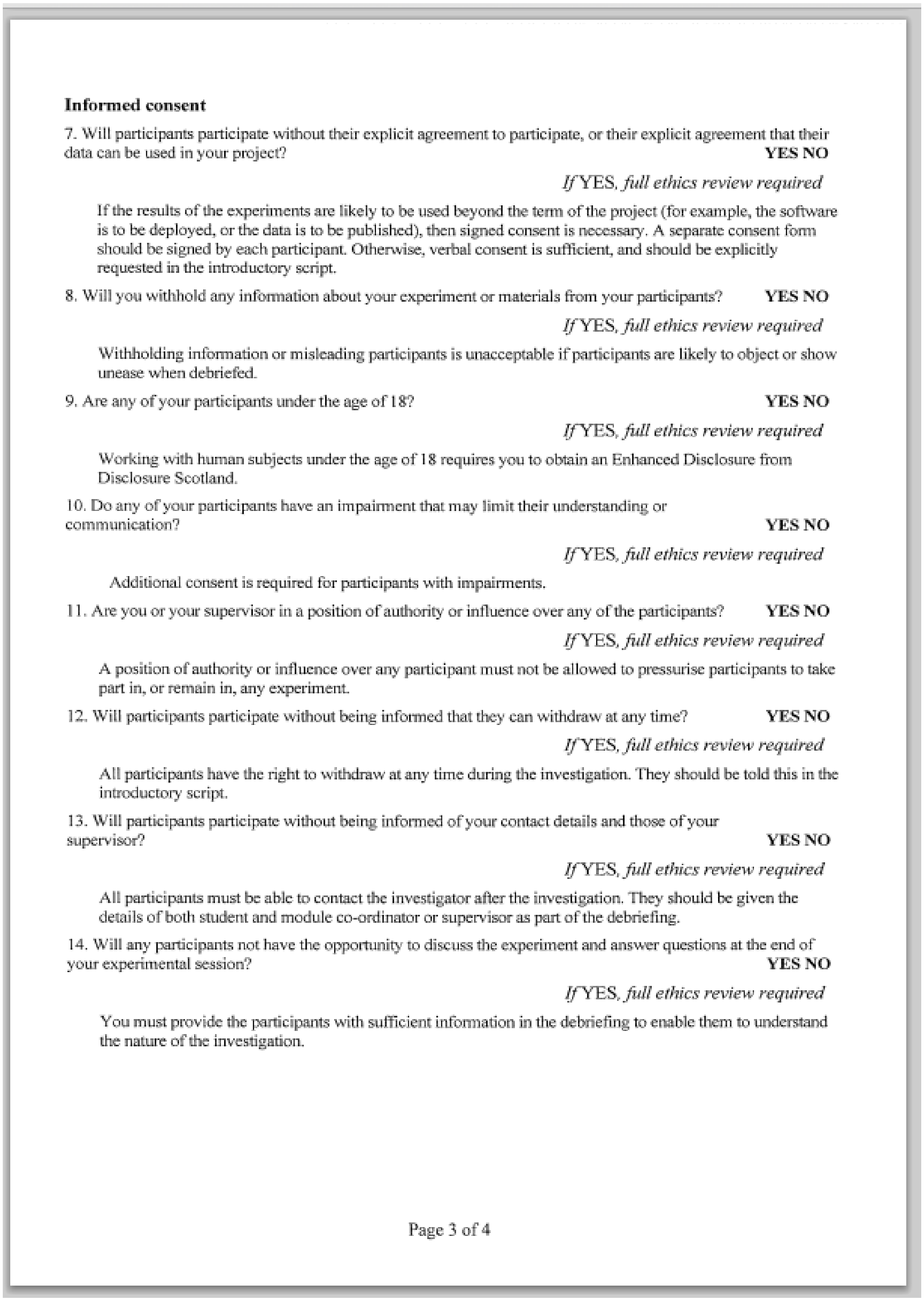
All participant data (hard-copy and soft-copy) should be stored securely, and in anonymous form.

6. Will you collect more data than are required for your immediate experimental hypotheses? **YES NO**

If YES, *full ethics review required*

Any collection of personal data should be adequate, relevant and not excessive in relation to the purposes for the collection.

Page 2 of 4



Informed consent

7. Will participants participate without their explicit agreement to participate, or their explicit agreement that their data can be used in your project? **YES NO**

If YES, *full ethics review required*

If the results of the experiments are likely to be used beyond the term of the project (for example, the software is to be deployed, or the data is to be published), then signed consent is necessary. A separate consent form should be signed by each participant. Otherwise, verbal consent is sufficient, and should be explicitly requested in the introductory script.

8. Will you withhold any information about your experiment or materials from your participants? **YES NO**

If YES, *full ethics review required*

Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

9. Are any of your participants under the age of 18? **YES NO**

If YES, *full ethics review required*

Working with human subjects under the age of 18 requires you to obtain an Enhanced Disclosure from Disclosure Scotland.

10. Do any of your participants have an impairment that may limit their understanding or communication? **YES NO**

If YES, *full ethics review required*

Additional consent is required for participants with impairments.

11. Are you or your supervisor in a position of authority or influence over any of the participants? **YES NO**

If YES, *full ethics review required*

A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

12. Will participants participate without being informed that they can withdraw at any time? **YES NO**

If YES, *full ethics review required*

All participants have the right to withdraw at any time during the investigation. They should be told this in the introductory script.

13. Will participants participate without being informed of your contact details and those of your supervisor? **YES NO**

If YES, *full ethics review required*

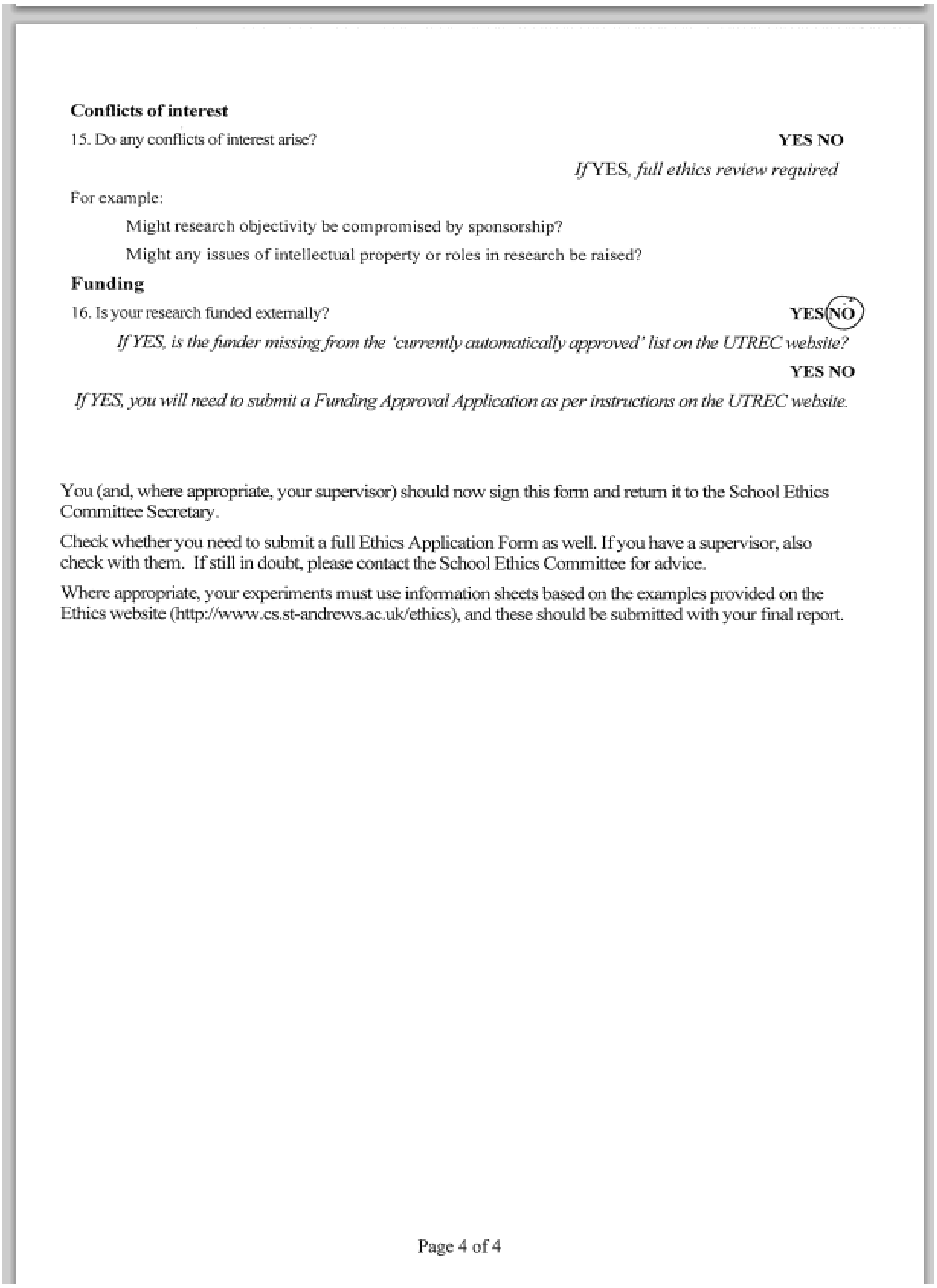
All participants must be able to contact the investigator after the investigation. They should be given the details of both student and module co-ordinator or supervisor as part of the debriefing.

14. Will any participants not have the opportunity to discuss the experiment and answer questions at the end of your experimental session? **YES NO**

If YES, *full ethics review required*

You must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation.

Page 3 of 4



Conflicts of interest

15. Do any conflicts of interest arise? **YES NO**

If YES, full ethics review required

For example:

Might research objectivity be compromised by sponsorship?

Might any issues of intellectual property or roles in research be raised?

Funding

16. Is your research funded externally? **YES [✓ NO]**

If YES, is the funder missing from the 'currently automatically approved' list on the UTREC website? **YES NO**

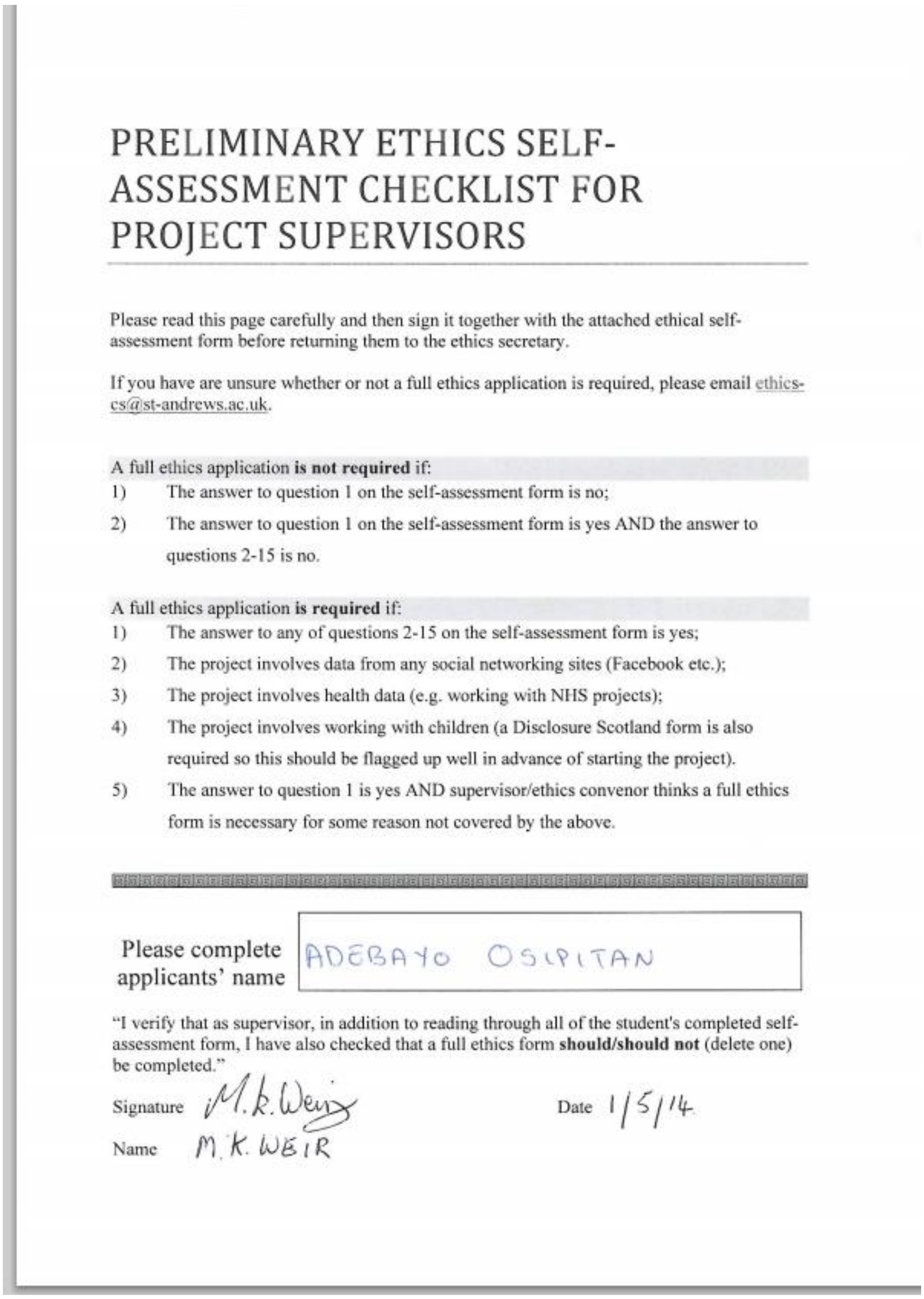
If YES, you will need to submit a Funding Approval Application as per instructions on the UTREC website.

You (and, where appropriate, your supervisor) should now sign this form and return it to the School Ethics Committee Secretary.

Check whether you need to submit a full Ethics Application Form as well. If you have a supervisor, also check with them. If still in doubt, please contact the School Ethics Committee for advice.

Where appropriate, your experiments must use information sheets based on the examples provided on the Ethics website (http://www.cs.st-andrews.ac.uk/ethics), and these should be submitted with your final report.

Page 4 of 4



PRELIMINARY ETHICS SELF-ASSESSMENT CHECKLIST FOR PROJECT SUPERVISORS

Please read this page carefully and then sign it together with the attached ethical self-assessment form before returning them to the ethics secretary.

If you have are unsure whether or not a full ethics application is required, please email ethics-cs@st-andrews.ac.uk.

A full ethics application **is not required** if:

1) The answer to question 1 on the self-assessment form is no;
2) The answer to question 1 on the self-assessment form is yes AND the answer to questions 2-15 is no.

A full ethics application **is required** if:

1) The answer to any of questions 2-15 on the self-assessment form is yes;
2) The project involves data from any social networking sites (Facebook etc.);
3) The project involves health data (e.g. working with NHS projects);
4) The project involves working with children (a Disclosure Scotland form is also required so this should be flagged up well in advance of starting the project).
5) The answer to question 1 is yes AND supervisor/ethics convenor thinks a full ethics form is necessary for some reason not covered by the above.

Please complete applicants' name	ADEBAYO OSIPITAN

"I verify that as supervisor, in addition to reading through all of the student's completed self-assessment form, I have also checked that a full ethics form **should/should not** (delete one) be completed."

Signature [illegible] Date 1/5/14

Name M.K. WEIR