Table of Contents

1 Introduction to the Project

1.1 Project Aims and Goals

This project has been undertaken with the intention of fully exploring a "full-stack" implementation of a simple robot capable of moving around its environment and localising itself in said environment. A full-stack implementation means as much of the robot and its accompanying software has been designed and built from the ground up with the exception of any critical software not related to solving the goals of robot movement and localisation. The intention of this is to present the opportunity to experience most of the issues involved with the construction and programming of robotic systems for a greater depth and understanding in the field and fundamental issues involved in it.

An overarching aim for this project is for the robot to be as simple as is necessary to achieve the project aims, in particular, the hardware should be very basic, and ideally leave room for potential incremental additions and advancements to be made in future.

1.2 Formal Project Specification

Some more formal specifications about the robot, and localisation software should be made to keep the implementation of the robot as simple as possible, and layout a feature list to work towards:

Localisation:

The method of localisation that is used should be straightforward to implement and should work with a preferably discrete map representation. The efficiency or complexity of the method will not be of concern but correctness and reliability of the method shall be important. A key aim of the project is to be able to localise the robot in at least one dimension, with a stretch goal of getting the robot to localise in two dimensions or at least produce a simulation of 2D localisation.

Hardware:

The physical hardware on the robot should be sufficient to allow the robot to move around and since the location of objects relative to its position. More specifically the robot should have the following capabilities:

- able to move forwards and backwards
- able to turn left and right
- able to measure the distance of objects in the environment
- able to tell how far it has travelled
- able to tell its orientation

2 Research

In this section, a range of literature related to the projects aims and goals are examined, and information collected is used to help make implementation decisions.

2.1 Hardware Research

A robot is an agent with a physical presence that has the ability to carry out some form of task through interacting with the environment around it. They are equipped with effectors which exert physical forces on the agent's environment and sensors, which allow them to capture information about said environment. (Russell and Norvig 2009, p971)

2.1.1 Design Principles

Before performing more in-depth research about the robot's hardware, and software it is essential to be aware of some design principles that should be taken into account to produce a functional robot.

Jones et al. (1998, p354) put forward the idea that "*complexity kills*", the observation is made that components and systems must interact with one another which brings about the potential for issues to develop due to this interaction. A robot should only have the essential features that are required to perform its task, they label a robot with only essentials as atomic, which can then be extended further once the robot's performance is robust and the interaction between systems is well understood.

Jones et al. (1998, p358) also present "Design Steps", a collection of ordered questions that should be considered when designing a new robot:

1. *"What is the robot supposed to do?"*
2. *"What is the simplest way to accomplish the task?"*
3. *"What mechanical platform is needed?"*
4. *"What information does the robot need?"*
5. *"What sensors can supply this information most effectively?"*
6. *"How can the problem be decomposed into behaviours?"*

These questions present a great set of points to keep in mind throughout this project.
An architecture that could see use in the robot's control software is the three-layer architecture.

The three-layer architecture consists of a reactive layer, an executive layer and a deliberative layer. The reactive layer exposes low-level control of the robot's hardware. The executive later acts as middleware between the deliberative layer and the reactive layer, receiving instructions from the deliberative layer and organising them for execution by the reactive later. (Russell and Norvig 2009, p1004)

The robot operating system (ROS) is a framework that allows the creation of robot software, it provides a structured peer to peer communication method that allows nodes to interact easily. Nodes are each a separate program with its own process performing some computation like low-level hardware control or path planning. Nodes send messages to communicate with one another and perform a task collectively. ROS is open source and aims to make code sharing and reuse easier through modularisation, and prevent roboticists re-inventing the wheel. (Quigley et al. 2009)

ROS could be used to go about implementing the robot's software, the functions of the robot can be broken down into individual nodes and the open source nature means that there is a lot of pre-existing software that can be reused to rapidly build any necessary portions of the software that are outside the aims of the project.

2.1.2 Control Unit

The choice of central processing unit to act as the "brain" of a robot is an important decision, two well supported control units are the Arduino platforms.

The Arduino and Raspberry Pi have all the components necessary for performing computation (CPU, memory, timers). The difference between them exists in their general purpose input output (GPIO) capabilities and CPU specifications. Microcontrollers like the Arduino are designed to have strong GPIO capabilities to drive hardware directly whereas Single Board Computers like the Raspberry Pi, which use microprocessors have weak GPIO capabilities, needing extra circuitry to drive hardware for them, however the microprocessors they use are far more powerful and can perform tasks that require more processing power. (*Ep 7: Comparing Arduino and Rasberry Pi* 2013)

Figure 2.1 A Raspberry Pi (Sparkfun 2014)

Figure 2.2: An Arduino Uno (Sparkfun 2014)

2.1.3 Sensor Technologies

Sensors allow a robot to perceive its environment by taking measurements, which can be processed to create an internal representation of the robot's environment (Nehmzow 2003).

Sensors can be classified by what type of information they provide.

Seigwart, Nourbakhsh and Scaramuzza (2011) provide some definitions:

- *"Proprioceptive sensors measure values internal to the system (robot), for example, motor speed, wheel, robot arm joint angles and battery voltage"*
- *"Exteroceptive sensors acquire information from the robot's environments, for example, distance measurements, light intensity, and sound amplitude."*
- *"Passive sensors measure ambient environment energy entering the sensor."*
- *"Active sensors emit energy into the environment, then measure the environmental reaction."*

To know what sensors are necessary to create the robot one of the questions brought up in the "**Design Principles**" section need to be answered:

"What sensors can supply this information most effectively?"

Information Robot Needs	**Suitable Sensor**
Able to sense the distance of objects in the environment around it	Range finders sensors are sensors that measure the distance of some target using either reflectivity, time-of-flight or geometric triangulation (Seigwart, Nourbakhsh and Scaramuzza 2011, p104).
Able to tell how far it has travelled	Odometry sensors are motions sensors that estimate changes in position over time by measuring wheel/motor speed and position (Seigwart, Nourbakhsh and Scaramuzza 2011, p104).
Able to tell its orientation	Heading and inertial sensors sense changes in orientation and acceleration relative to some initial reference frame (Seigwart, Nourbakhsh and Scaramuzza 2011, p104).

Table 2.1 The information the robot needs and sensor that can supply this information

Range Finders

Ultrasonic sensors: ultrasonic sensors operate in a similar way in which some animals use echolocation, an ultrasonic ping is emitted and the response, which bounces off objects in front of the sensor, is detected, distance can be computed using the known speed of sound and the time it took to receive the response. The sound emitted from the sensor spreads out in a cone shape, known as a beam pattern, as sound naturally spreads out as it travels, the position of a detected object could be anywhere in this cone, this can make it difficult to detect exactly where an object is relative to the sensor. (Nehmzow 2003)

Typical ranges for ultrasonic sensors are between 4m to 15m depending on the beam pattern.

Figure 2.3 An ultrasonic range sensor (MicroControls.org 2015)

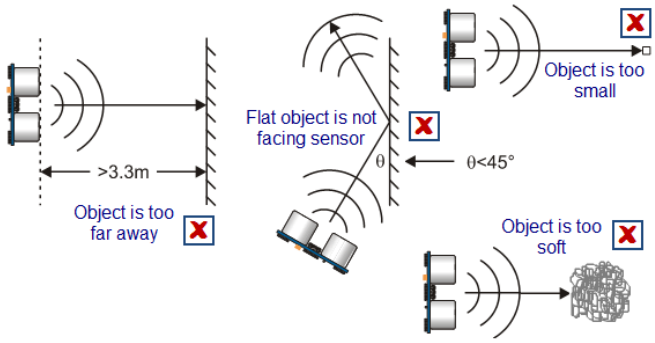


Figure 2.4 Situations that can negatively affect ultrasonic sensor readings (MicroControls.org 2015)

***LIght Detection And Ranging*:** LIDAR sensors operate by emitting a pulse beam of near infrared light and the time taken to detect a reflection as a response is used to calculate the distance of a target. The near infrared light used has a short wave length that reduces the chance of emitted signals being reflected away entirely. This makes the shape of objects less of a problem to LIDAR compared to sound based methods. The maximum range of commercial LIDAR range finders can be in the hundreds of meters and their accuracy is typically on the scale of millimetres. (Nehmzow 2003, p31)

LIDAR sensors are far more expensive than ultrasonic range sensors and are typically larger.

Figure 2.5 A 360° LIDAR scanner (RobotShop 2014)

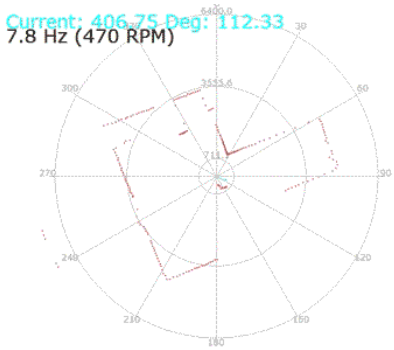


Figure 2.6 Scan Generated By A LIDAR Sensor (RobotShop 2014)

Odometry sensors

Shaft Encoders: encoders are sensors that are mounted on a rotating wheel or shaft on a robot, they are made up of a rotating disc and sensors which read that disc to tell where the wheel is or how much it has rotated. Absolute encoders make use of discs that have unique markings on the disc for each position of the wheel which when read provide its absolute position. Incremental encoders use binary markings that generate pulses that represent how much that disc has moved relative to some starting position. (Nehmzow 2003, p35)

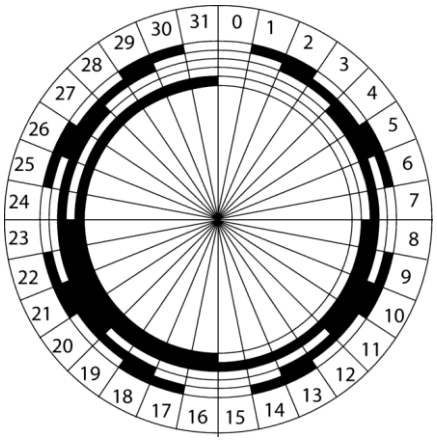


Figure 2.7 An absolute encoder (Rockwell Automation 2016)

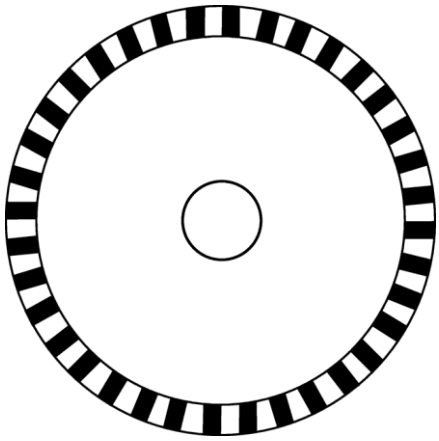


Figure 2.8 A binary encoder (Rockwell Automation 2016)

Heading and Inertial sensors

Inertial Measurement Unit: an IMU is a component that can be used to provide an estimate of the relative position, velocity and acceleration information about the device itself and by extension, some form a robot that it is attached to. It uses a combination of a gyroscope and an accelerometer. Combining these two devices it can provide a *six-degrees-of-freedom pose* of a robot. This pose is an estimate of the robot's position (x, y, z) and its orientation (yaw, pitch, roll) with respect to some initial frame of reference. (Seigwart, Nourbakhsh and Scaramuzza 2011, p122)

2.1.4 Locomotion

Because of the simplicity in using wheels for locomotion, focus is placed on wheeled locomotive systems and their configurations. Wheel configuration refers to the way in which wheels are placed on a robot:

Differential Steering is a popular configuration, this configuration typically makes use of two powered wheels that are on opposite sides of a robot, by controlling the speed and direction of each wheel the robot can be steered in the desired direction. This method of steering provides a high degree of manoeuvrability as it can turn about a pivot point at its centre with a turn radius of zero degrees. (McComb 2011, p213)

Omnidirectional steering or holonomic steering is a configuration that allows movement in any direction without having to change the orientation of the robot. The disadvantage of this type of steering that it is considerably more mechanically complex to build than other configurations. (McComb 2011, p214)

The choice of motor used for the robot's locomotion is also a key point that requires consideration.

Continuous DC Motors are motors that simply rotate continuously as long as power is supplied to them, the speed and direction of the motor can be controlled (ModMyPi 2013). The position of these motors cannot be measured but their speed and revolutions can be tracked with the use of incremental encoders mounted on the shaft.

Servo Motors are essentially DC motors with other components to control the positions of the motor, they have a power and ground wire as well as a wire for controlling the position of the servo motor by sending an output signal to it. These motors do not rotate continuously and are limited to a rotational range of 180 degrees. (ModMyPi 2013)

Stepper Motors are similar to servos in that they can be instructed to move to a certain position through a series of steps. Control circuitry is required to instruct a stepper to move to a certain position. The speed and relative position of the shaft can be easily controlled but there is no feedback mechanism to determine how the stepper has moved. (ModMyPi 2013)

2.2 Localisation Research

"*Mobile Robot localisation is the problem of determining the pose of a robot relative to a given map of the environment. It is often called position estimation*" (Thrun, Fox and Wolfram 2005, p191)

Probabilistic Robotics is a term used to refer to the approach of using probability to model and represent the state of a robotic system. The probabilistic approach takes into account the inherent uncertainty and error in robot perception, and represents the state of a robotic system as a distribution of estimates across a state space rather than a single estimate. (Thrun, Fox and Wolfram 2005, p5)

2.2.1 Current State of the Art

There are a number of state of the art approaches to mobile robot localisation that are not applicable to the robot built in this project due to their complexity and scope but are mentioned to display the range of approaches available to this problem. One such state of the art method is Visual Simultaneous Localisation and Mapping (Visual SLAM).

Visual SLAM

Visual SLAM refers to the problem of using only visual data, in the form of images, as the only source of information to determine the position of a robot while simultaneously creating an internal representation of the environment as a map. A range of camera technology is used in this approach such as wide-angle cameras, multi-camera setups or cameras that can simultaneously produce depth maps and colour images. (Fuentes-Pacheco et al. 2012)

Fuentes-Pacheco et al. (2012) present coverage of the three main categories of approach that offer solutions to the problem of Visual SLAM, the traditional probabilistic filter approach, the structure from motion approach and approaches that have roots in biology.

Probabilistic filter based techniques are the more common and traditional approach that represent the state of a robot's position and the robot's environment as a probability distribution. The structure from motion approach is rooted in computer vision and is able to create 3D structure from a series of images captured from the environment, features are extracted from images using a range of techniques examples being visual odometry and salient feature extraction. Biology inspired approaches attempt to study, model and apply the way in which certain animals such as rats or ants use visual landmarks to guide themselves. (Fuentes-Pacheco et al. 2012)

These vision based approaches are an important path in which the larger problem of simultaneous localisation and mapping is heading but these techniques are beyond the scope of this project which attempts to use a simple distance measuring sensor to solve the problem of localisation. The following techniques are more applicable to this project.

It is important to make note of a crucial general algorithm that is used in probabilistic robotics called the Bayes filter, many other probabilistic belief estimation algorithms are based on this filter.

The Bayes filter is used to recursively calculate a probability density function that represents a belief about the position of the robot in its environment. This is done in two steps, a control update uses control information about movement to calculate the new belief based on said motion and the prior belief. A measurement update is then performed using sensory information to update the belief, given the observed measurement. (Thrun, Fox and Wolfram 2005, p27)

2.2.2 Grid Localisation

Grid localisation is a technique that uses a histogram filter (a discrete variant of the Bayes filter) to represent a belief about the position of a robot in its environment. Grid localisation works by separating the space of all possible poses for a robot in a given map into a finite amount across a grid of cells, each cell having a probability that represents the likely hood of the robot being in that cell. (Thrun, Fox and Wolfram 2005)

The Grid localisation algorithm takes in as input the previous belief, the most recent sensor measurement, control information and a map of the environment. The probabilities of the grid cells are updated by matching the sensor measurements with the map and are moved in the grid according to the control information producing a new belief. The resolution of the grid in this algorithm is key to what kind of sensor can be used, the time the algorithm takes to produce a belief estimate, and the kind of output that can be expected of the algorithm. (Thrun, Fox and Wolfram 2005)

2.2.3 Monte Carlo Localisation

Monte Carlo Localisation (MCL) refers to a localisation technique that uses a particle filter (another variant of the Bayes filter) that makes use of rapid sampling and resampling to represent the belief about the robots state, it efficiently uses an adaptive sampling method that dynamically determines the amount of samples necessary. (Fox et al. 1999)

MCL represents the belief of the robot's position as a collection of weighted particles, with the weight of a given particle representing the probability of the robot being at said particle's position, on initialisation particles are randomly distributed throughout a provided map to represent global uncertainty of the robot's position. Readings from sensors are used to assign the particles a weight, and robot motion is used to re-sample the particles and move them according to the robot's motion. Re-sampling randomly picks member particles for the new sample according to their weight. (Fox et al. 1999)

This method essentially makes the particles which most likely represent the robots true state survive longer into successive cycles until a fairly good estimate of the robot's position is achieved. This method of localisation can be use in discreet or continuous spaces.

2.3 Decisions Made Based On Research

This section presents the decisions made about what hardware will be used to go about constructing the robot as well as the localisation method that will be implemented, these decisions are based on the research collected with it used to justify the decisions made.

2.3.1 Hardware

Design Principles

The architecture for the robot's software will actually be made up of two pieces of software, local on-board software running on the robot and off-board software running on a computer connected remotely via Bluetooth. This architecture is an implementation of the three-layer architecture. The reactive and executive layers are the on-board software and the deliberative layer is the off-board software.

The deliberative is the layer that will perform the task of localisation, it will send instructions to the executive layer telling the robot to move or gather data from its sensors. This decision was made so that the deliberative layer is run on hardware with more computational power, since it is the layer that will perform the higher-level task of localisation.

Control Unit

The Arduino platform will be used as the control unit for the robot, this decision was made due to it being the simpler of the choice between itself and the Raspberry Pi, it does not require an operating system and possesses many GPIO pins that do not need to be connected to any extra circuitry to control electronic devices.

The low computational power of the Arduino CPU is counteracted by the decision to connect the robot remotely to a computer.

Sensor Technologies

Range Finder: the robot will make use of a LIDAR range finder this is due to it being easy to determine the position of a distance measurement compared to the beam pattern produced by Ultrasonic sensors and the reduced chance of an object's shape affecting distance readings. LIDAR also offers a greater maximum measurement distance compared to ultrasonic sensors.

Odometry: the robot will not make use of any odometry sensors due to the type of drive motor selected, this will keep the mechanical complexity of the robot as low as possible.

Heading Sensor: the robot will make use of a microelectromechanical systems (MEMS) inertial measurement unit (IMU), due to it simply being the most accurate way of determining the heading of the robot compared to alternatives.

Locomotion

The robot will make use of the differential steering configuration and stepper motors. The differential steering configuration was chosen as it is the simplest mechanical configuration available and gives the robot a high degree of manoeuvrability.

Stepper motors were selected as the type of drive motor as the exact amount of steps made by each motor can be controlled by software, if the circumference of the drive wheels is known the distance travelled can be calculated using these two pieces of information. This means the movement of the robot can be estimated without the need for encoders or the software necessary to operate them.

Using stepper motors as the drive motor from the robot is somewhat an original locomotive approach, most robots tend to use continuous DC motors with encoders for odometry tracking, it will be interesting to see the outcome of only using steppers without explicit odometry and if it is a feasible alternative to the standard choice in drive motor.

2.3.2 Localisation

For the robot's localisation, both the Grid Localisation and Monte Carlo Localisation methods will be explored and implemented. The decision two use both is simply to develop a deeper understanding of both rather than a single method, also since grid localisation only operates on discrete spaces and the Monte Carlo method can work in discrete or continuous space, it will provide a good insight on localisation in continuous space.

The grid localisation method will be used to implement the one-dimensional localisation software; this implementation is meant to be simple and grid localisation is the simpler method. The Monte Carlo method will be used to address the more challenging problem of 2D localisation.

3 Planning and Design Overview

3.1 Mobile Platform Design

This section presents a basic design of the physical robot. Listed is all the hardware that is used and the role of each component in the system, Appendix 1 provides component specifications and why these were chosen specifically. Also presented are some 3D sketches to give an idea what the robot will look like, in particular two conceptual ways of making the LIDAR spin have been modelled to assist with making this later.

The scope and budget of the project has limited the type of hardware used in the project to small low cost items.

Key Points

- The LIDAR will be on a rotating platform to measure 360° around the robot
- The LIDAR will be rotated using a stepper motor
- The wheels will have a differential steering configuration
- The robot will make use of a simple power system documented in Appendix 2

3.1.1 Component List

Hardware Component	Component Role
 *Figure 3.1 A Lidar Lite module* (Sparkfun 2014)	Used as the range sensor.
 *Figure 3.2 A Bipolar Stepper Motor* (Sparkfun 2014)	Used as drive motors and to rotate the LIDAR

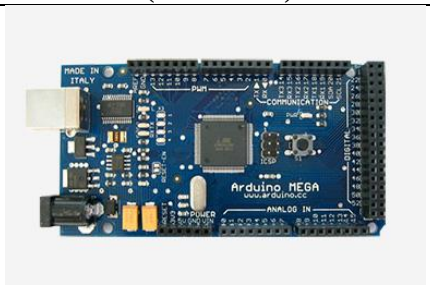
 *Figure 3.3 HC-05 Module* (HobbyComponents 2014)	Used for Bluetooth communication
 *Figure 3.4 The MPU-6050* (Arduino.cc 2016)	Used to provide orientation information
 *Figure 3.5 The A4988 Stepper Driver* (Pololu 2012)	Used to operate the stepper motors
 *Figure 3.6 An Arduino Mega* (Arduino.cc 2016)	Used as the control unit that runs the on-board software

Table 3.1 The component list with image and the role the components play in the robot

3.1.2 Sketches

These simple 3D sketches were created to plan out what the physical robot will look like and how a key component like the LIDAR could be put together, construction was based on these simple sketches.

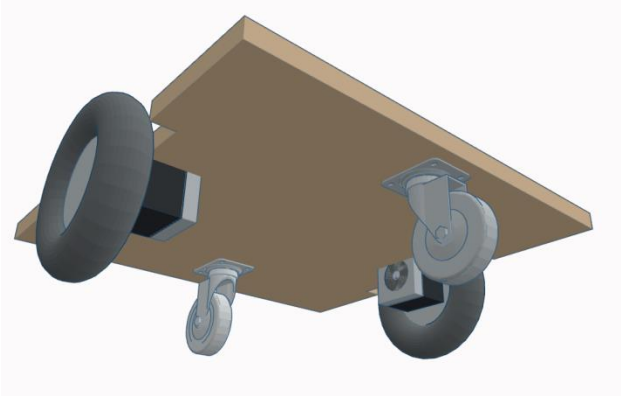


Figure 3.7 Sketch of the platform from below

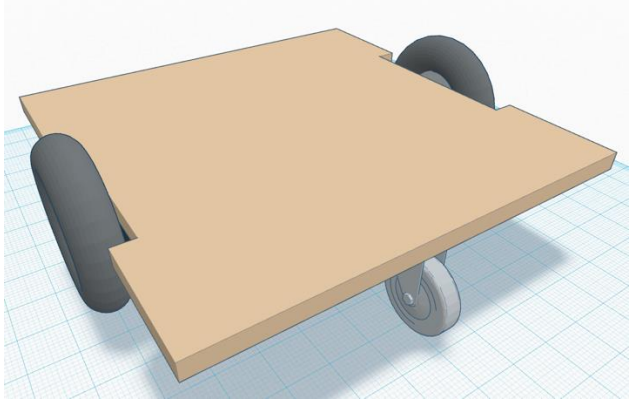


Figure 3.8 Sketch of the platform from above

The sketches in figures 3.7 and 3.8 show what the robot platform will look like, a differential steering configuration with larger drive wheels on opposite sides and castor wheels providing support at the front and back. The drive motors are fastened on the bottom to allow more space on the top for components.

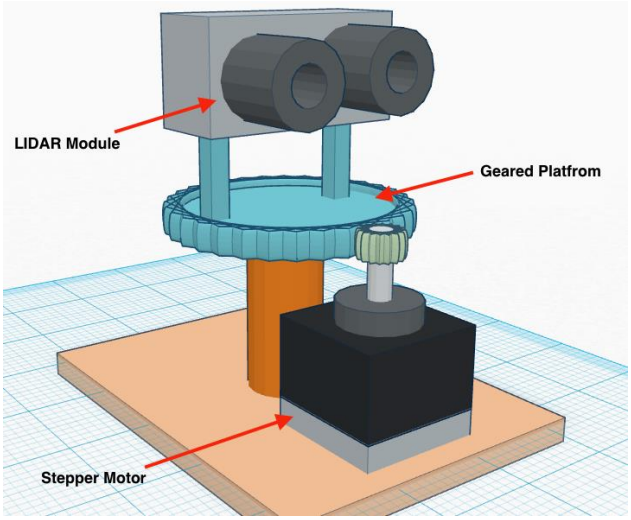


Figure 3.9 Rotating the LIDAR directly on a geared platform

Figure 3.9 shows a concept for the mechanism that rotates the LIDAR on the robot. In this configuration the module is attached to a platform that is rotated using gearing on the platform and a stepper, this achieves 360° scanning. This would require a slip ring to connect the LIDAR's wires to the control unit so the module can spin continuously.

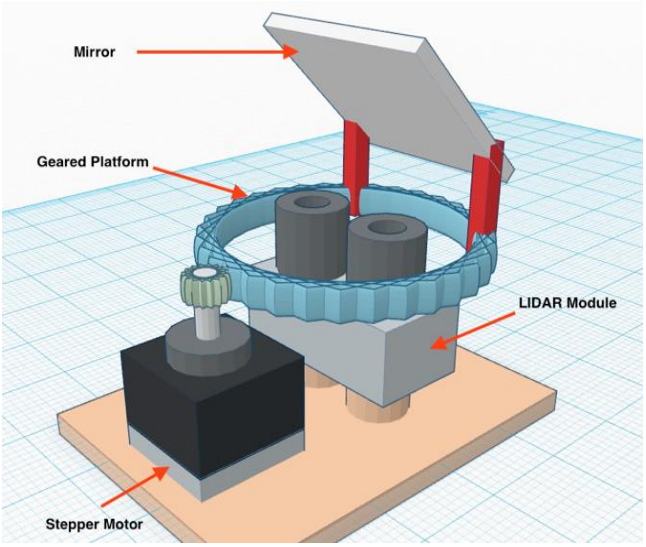


Figure 3.10 Rotating a mirror above the LIDAR

Figure 3.10 shows an alternative concept for the mechanism that achieves 360°scanning. In this configuration, the module is kept still and a mirror attached to a platform which rotates above the LIDAR, the module can still measure distances through the reflection on the mirror. This configuration requires no slip ring.

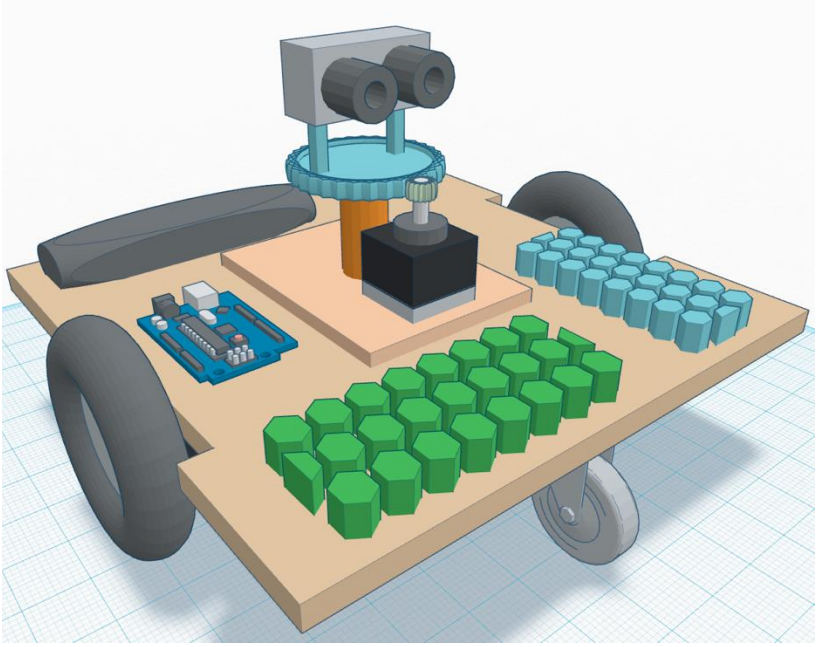


Figure 3.11 Mock up of the robot with components on the platform

Figure 3.11 shows a concept of what the robot will look like when completed. The electronic components are all distributed across the top of the platform connected to the control unit, with the LIDAR in the very centre of the platform.

3.2 On-board Software Design

Following the design philosophy of simplicity, the robot's on-board software is designed to be as straight forward as the hardware, to do this it has been broken down into modules, with each module representing a separate "system" in the hardware.

The term "system" is used to describe a collection of components or a single component that have a particular purpose in the robot, the modules that will be created are called **Core**, **DiffDrive**, **Lidar360**, **Common**, **RemoteComm** and **TurnSensor**.

Module	Role
Core	Acts as the "glue" module, it is responsible for the overall functioning of the robot. It runs the main loop that receives, processes and executes commands using functionality provided by the other modules
DiffDrive	Provides access the robots locomotive capabilities, it has methods that take motion commands and synchronises the robot's drive motors to move the robot to a desired position.
Lidar360	Provides access to the distance measurement capabilities of the robot. The module can be instructed to spin the LIDAR to a certain heading to take a measurement or take an entire 360° sweep of measurements.
Common	Provides common functionality used throughout multiple modules in the software. It defines the instruction set the robot can perform, defines data types to be used by other modules and provides a single place to give named definitions to individual pins on the control board according to what their role is.
RemoteComm	Responsible for establishing a connection to a remote computer using Bluetooth, and sending or receiving data on the connection once it is established.
TurnSensor	Provides access to the robot's IMU. It receives, processes and abstracts the data read from the IMU into a format that is easier to use in other modules that need inertial information such as DiffDrive.

Table 3.2 The modules in the on-board software and their respective roles

Figure 3.12 illustrates the on-board software architecture and how the modules interact.

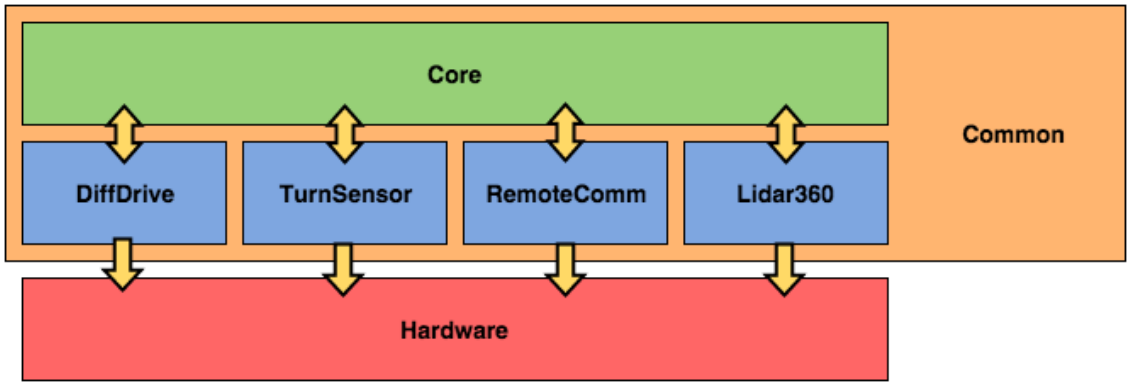


Figure 3.12 On-board software block diagram

3.3 Off-board Software Design

It should be noted that the off-board software design was created once the construction of the robot and its on-board software were completed. The decision to implement the project in this order was to make the design for the localisation software as flexible as possible meaning difficulties encountered during the robot's construction could be accounted for.

A major difficulty during construction was getting the IMU to work reliably, details of this are covered in the implementation chapter, essentially the IMU was excluded from the robot. This meant that the robot cannot turn accurately so the decision was made to have the physical robot perform one dimensional localisation and simulate an implementation of two dimensional localisation (as this requires the robot being able to turn accurately).

3.3.1 One Dimensional Localisation (ODL)

The ODL software implements the arguably easier Grid Localisation algorithm but since it interacts with the robot in the real world (rather than simulating one) there are some factors regarding communication with the robot to consider.

The ODL software again follows the design philosophy of simplicity through modularity, the software is broken down using an object oriented approach with the main objects being **CoreUI**, **Map** and **Robot**.

Class	Role
CoreUI	Provides "glue" that connects the other two classes together as well as providing a user interface. It provides a simple user interface that can be used to send the instructions to the physical robot, via a Robot object, as well as set information about the map and its layout. It also feeds data captured from a Robot object into a Map object which allows it to transform the probability distribution, it then takes the probability distribution in a Map object and creates a visualisation.
Robot	Represents the robot in the software, all commands that are intended to go to the physical robot can be sent using the class methods, which send corresponding instructions via Bluetooth. Data created by the physical robot's sensors can also be received by the program through this object.
Map	Responsible for holding information about map size and layout as well as containing the probability distribution that represents the chance of the robot being in a given gird cell. This class implements the actual Grid Localisation algorithm that allows it to transform its probability distribution according to the Robots motion and sensor data.

Table 3.3 The classes in the ODL software design and their respective roles

Figure 3.13 below illustrates this design:

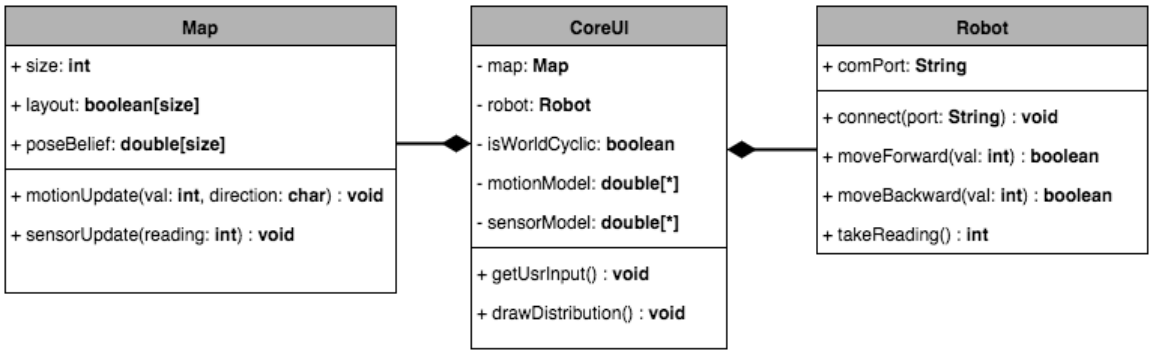


Figure 3.13 Basic class diagram for the one dimensional localisation software

3.3.2 Two Dimensional Localisation (TDL)

The TDL will be implemented in a simulated world using explicit landmarks that the robot can detect the distance to, even though this is not exactly like the configuration of the physical robot which uses distance sensing based on a beam it is useful for exploring a simple implementation of 2D localisation using the Monte Carlo Method.

Like the ODL software this design uses and object oriented approach with main classes: **CoreUI**, **Landmark**, **FilterAlgorithm**, **Robot** and **Particle**.

Class	Role
CoreUI	Used as the "glue" between all the other classes, provides an interface where the user can change settings regarding the robot's attributes, the position of landmarks in the map or the amount of particles present. It provides a visualisation of the localisation happening as an animated sequence, showing the particle estimates as well as the position of the robot.
Landmark	Used to specify positions on the map where "landmarks" are located, the robot can sense the distance to landmarks that are in range to localise itself.
FilterAlgorithm	Used to contain all of the functionality that actually implements the Monte Carlo localisation method, such as probability calculation or re-sampling
Robot	Used to represent the simulated robot, its position in the map and it's heading, it has functionality that can simulate the robot moving and taking measurement readings from it sensors
Particle	A subclass of Robot, has all the same functionality with extra features exclusive to particles such as a weight assigned to it according to its feasibility as a robot pose.

Table 3.4 The classes in the TDL software design and their respective roles

Figure 3.14 below illustrates this design:

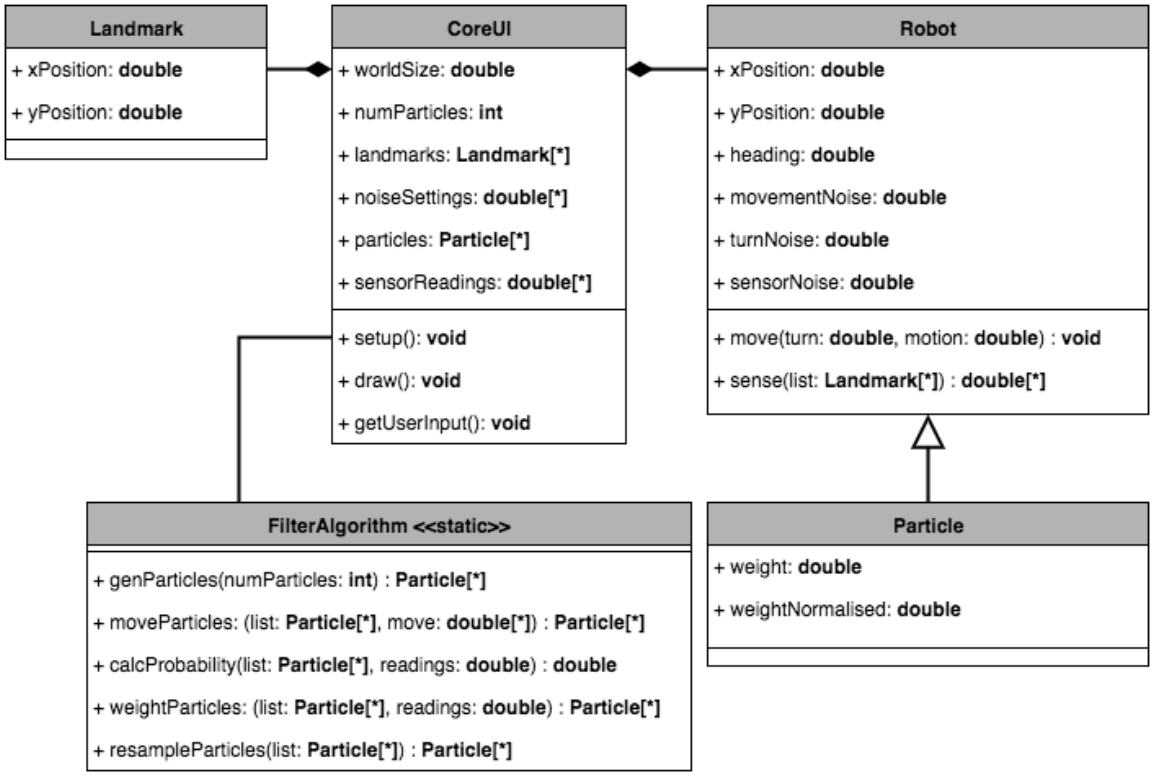


Figure 3.14 Basic class diagram for the two dimensional localisation software

4 Implementation of Deliverables

This section covers the implementation of each of the deliverables this project produced. The approach, methods and techniques used to implement each deliverable are discussed as well as a detailed operational overview and key issues faced during implementation. Attributions are also given for libraries, examples and ideas from others that were used in the implementation of the deliverables.

4.1 Robot Hardware and On-board Software

4.1.1 Acknowledgement & Sources

Library / Idea / Example	Usage
Arduino Standard Lib (Arduino LLC 2015)	Used in the software to control the Arduino board itself and by extension all the connected electronics.
Pulsed Light Examples (Pulsed Light 2015)	Examples that show how to interact with the LIDAR through I2C to initialise it and take measurements were reused in the Lidar360 class.
I2C Master Lib (DSS Circuits 2014)	Used to provide I2C communication with the LIDAR
AccelStepper Lib (McCauley 2010)	Used to provide accurate control over the drive stepper motors and the stepper motor used in the LIDAR system

Table 4.1 Libraries & examples used in implementation and what they were used for

4.1.2 Aims Achieved

Table 4.2 lists project aims the hardware deliverable has achieved and how well it was done.

Project Aim (for Robot)	Notes on Achievement
Forwards and backwards movement	Fully implemented can be commanded to move forwards or backwards, in grid spaces or millimetres
Ability to turn left and right	Implemented with limited success, uses dead reckoning to calculate turns but is prone to errors
Able to sense distance to objects	Fully implemented can be commanded to take a measurement at a specific heading or perform a 360° sweep.
Able to tell how far it has travelled	This can be done if the control application keeps track of motion commands sent to the robot.
Able to tell orientation	Not implemented, IMU component was left out, explained in issues later on.

Table 4.2 Relevant hardware project aims and if the aim was achieved with supporting notes

4.1.3 Development Approach & Methods

The approach to development of the hardware and its software mirrored the design presented earlier, the robot was broken down into modules that represent systems in the robot which each have a purpose, with each module providing an interface for interaction. This design of modular components with interfaces was implemented in the hardware, in the form of proto-boards with connection pins, and in the software through classes with public methods exposed through a header file. The proto-boards meant that a hardware system was easy to interface with, just by connecting the Arduino pins to the correct proto-board pins.

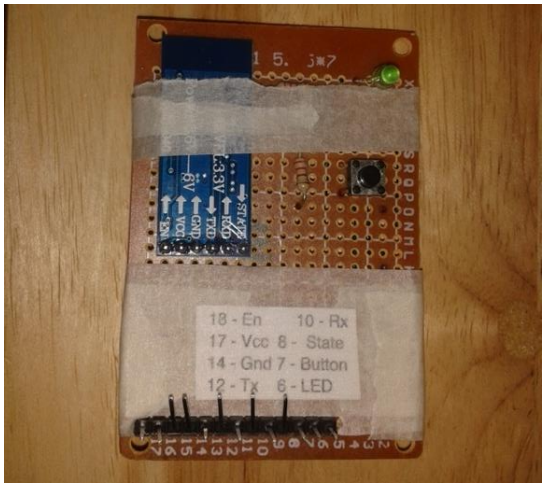


Figure 4.1 The Bluetooth proto-board interface

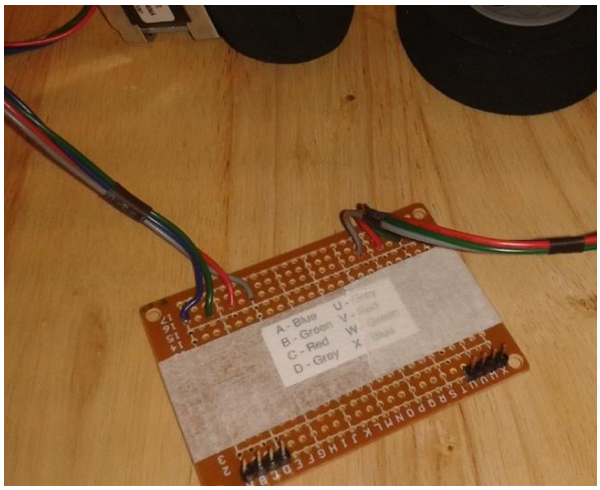


Figure 4.2 The drive system proto-board interface

This design choice worked well and meant that individual systems in the Robot could be worked on in isolation without concern for the other systems other than the interface that they provided. With the conceptual separation of the robot into systems, a fairly consistent method for developing the functionality of each system emerged shown in figure 4.3.

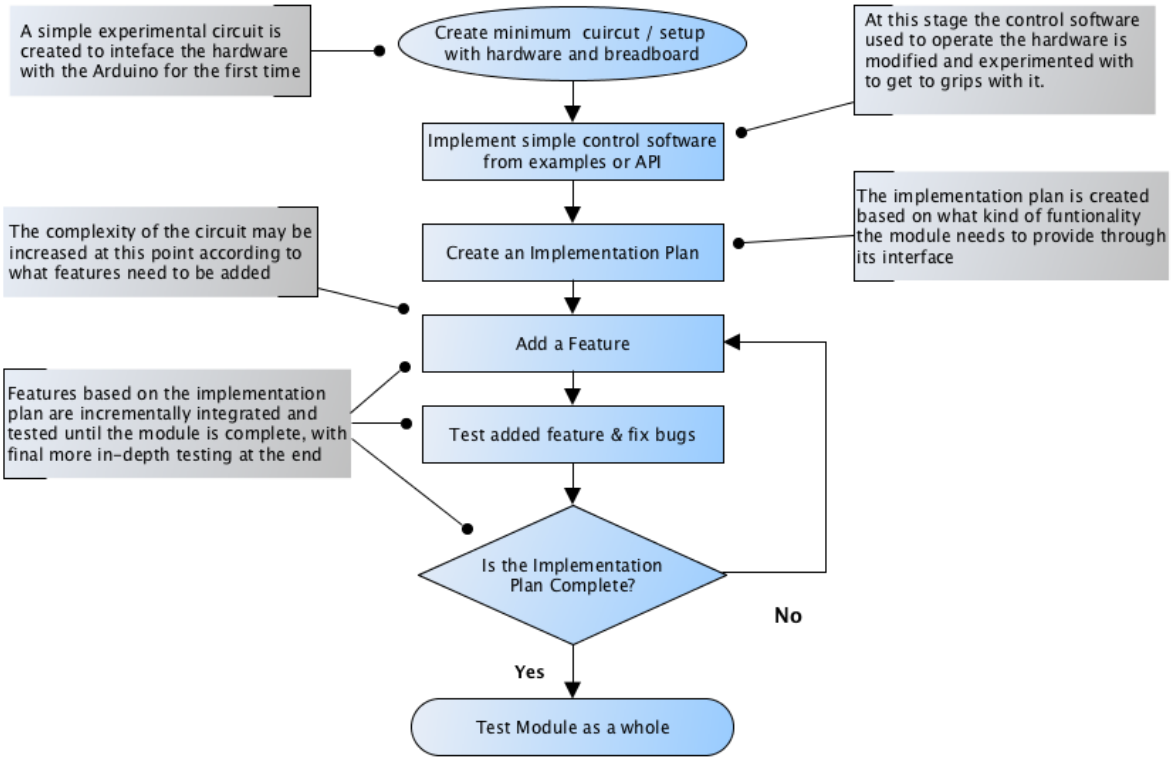


Figure 4.3 The development process for each system in the robot

This method of hardware development seemed the most natural way to go about developing each hardware system and worked well as the time was taken to understand the hardware well before attempting to implement feature critical to the robot's operation, bugs were found and dealt with typically before a new feature was added to a given system.

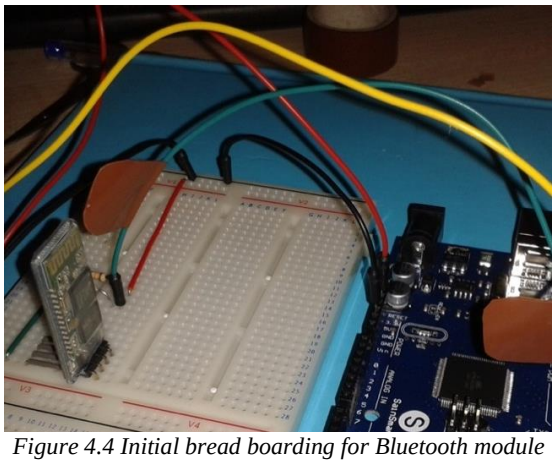


Figure 4.4 Initial bread boarding for Bluetooth module

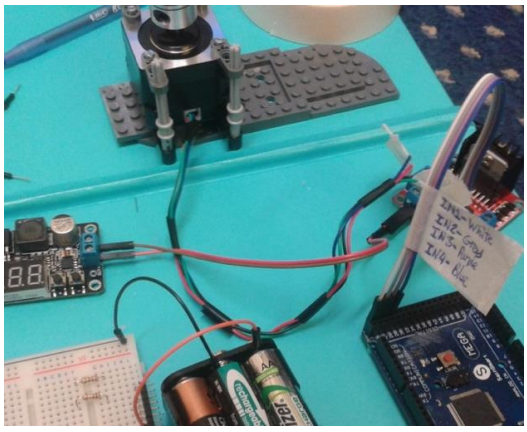


Figure 4.5 Initial bread boarding for stepper motors

Through development emphasis was put on creating on-board control software that was as portable as possible, meaning that the software could be uploaded to another Arduino model or Arduino compatible board with minimal changes. This resulted in the **Common** module becoming home to files that could be changed to configure the software. In the **Common** module, the function of pins can be changed, new instructions added to the instruction set, and defined values like delays or buffer sizes changed to different settings.

Debugging mechanisms were also a key point throughout development, when new features were actively being developed for a module – simple serial logging mechanisms were implemented to be used for debugging purposes, this worked extremely well. An LCD screen was also added to the robot to allow short debug messages to be printed to the LCD once all the systems were put together, this compensated for not being able to attach a serial cable when the robot is mobile.

```
send handshake signal
read vaue sent:S
values match
send hand shake stage B signal
Connected
--------------------------------------------------
Instruction available read it into buffer
instruction bytes: 10, message: 2,35,F,424, bytes re
Checking for mismatch between bytes read & bytes rec
No error occured, parse instruction & attempt execut
check to see if an error occured during verification
No error occured, send okay message
No error occured, send complete message
--------------------------------------------------
```

Figure 4.6 Serial debug message printed out on a terminal

Figure 4.7 Debug Messages displayed on the robot's LCD

4.1.4 Operational Overview

The robot's software is based on the three-layer architecture with an implementation of a remote fetch-execute cycle, the deliberative and reactive layers reside in the on-board software and it uses a fetch-execute cycle to get and execute remote instructions sent to it, the **Core** module facilitates all of this.

The Core module begins by calling the "init" methods for all modules, these methods set the pins and settings each module will use. Core then waits for the user to verify that the LIDAR output on the LCD looks okay (the module may need resetting rarely) and press the continue button (A), the zero position of the LIDAR can then be set using the A (confirm) and B (spin right) buttons. Once the LIDAR is zeroed, the robot waits for a connection from the control software, when connected the instruction cycle begins. Figure 4.8 illustrates the cycle.

This fetch-execute cycle works extremely well. It allows the robot software to remain simple and focused on its specialized task of parsing and executing instructions, leaving higher-level tasks like localisation to the remote software. Having the robot only execute commands set to it from an instruction set also means that the control software can simply be swapped out to change what the robot does – the robot's on-board software is not specific to any task.

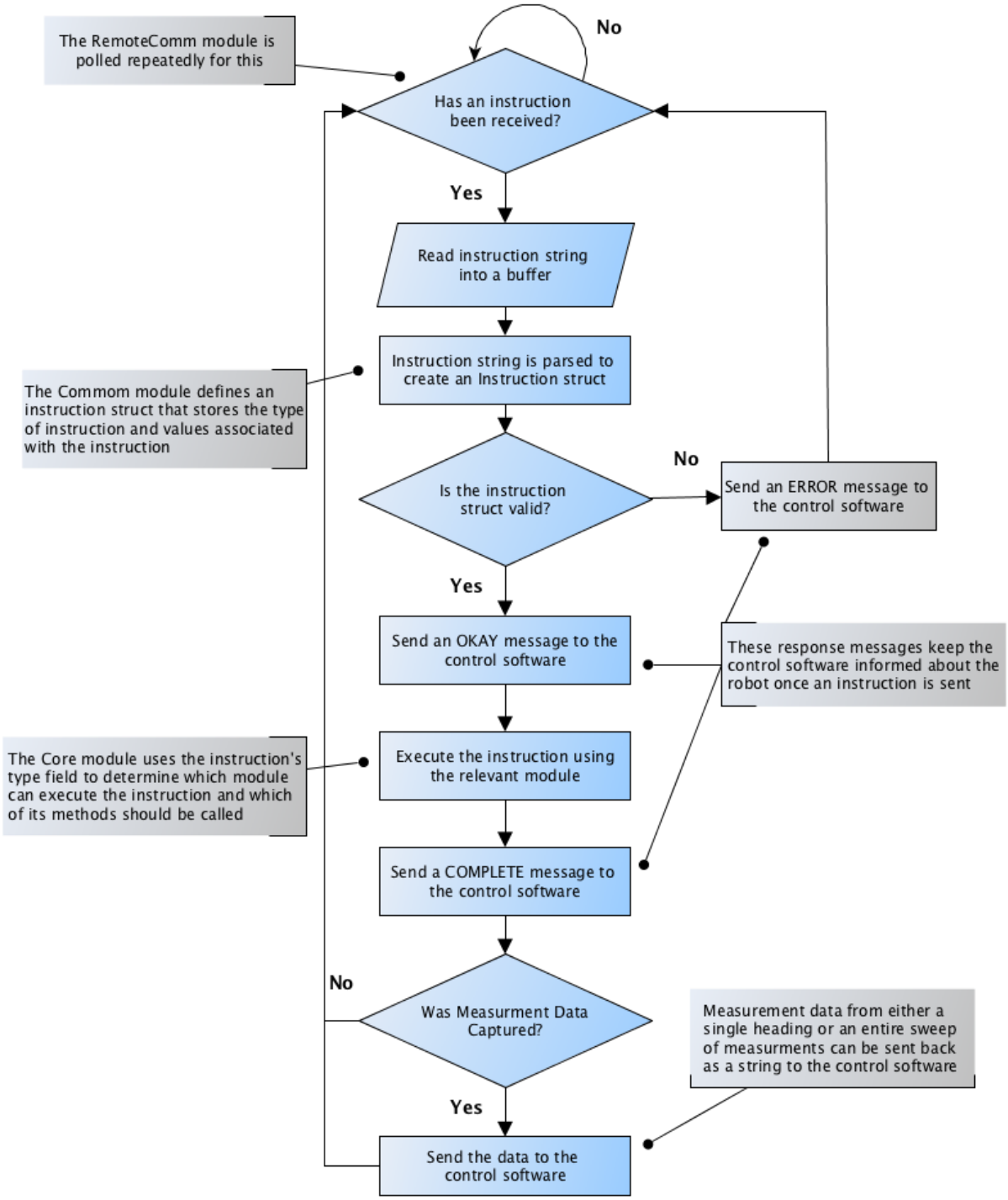


Figure 4.8 How the on-board software running on the Arduino operates

4.1.5 Key Issues

Fitting Components on Platform:

This issue came about due to the robot possessing a large amount of components and a relatively small platform for them to be placed on, as well as the systems specified in the robot's design there are other components such as breadboards, power regulators, wires and batteries that all take up additional space on the platform. All of the components did fit on a single platform but it was crowded, some components needed to be strapped to one another, and the weight was not evenly distributed.

The solution to this was to add a second platform to the robot. The breadboard, Arduino and power system are on the lower platform and the LIDAR and Bluetooth systems on the upper platform. The robot weighs more but the components are laid out spaciously, distributing the weight well and leaving room for future components to be added.

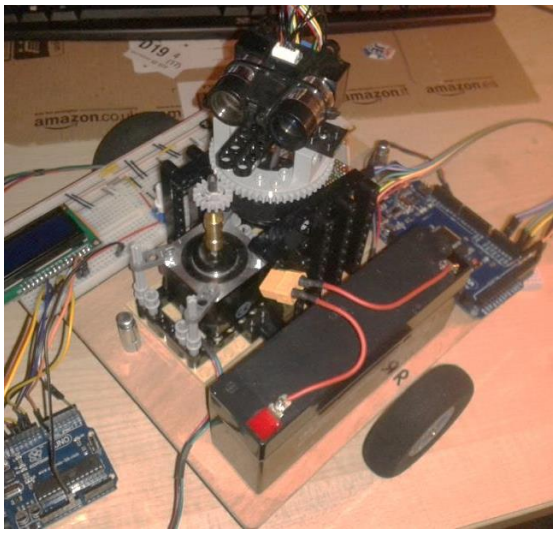


Figure 4.9 The robot with a single platform setup

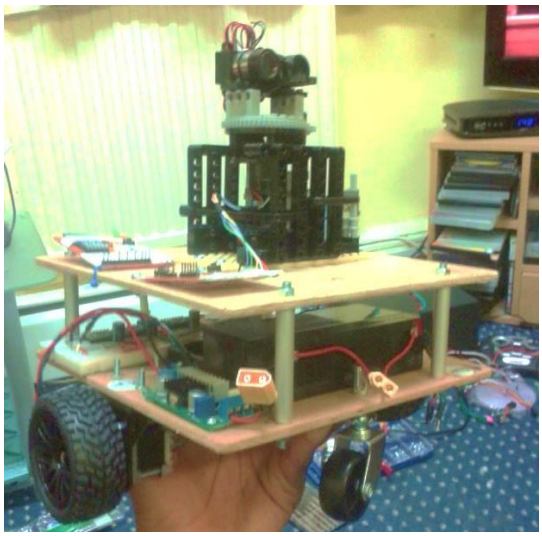


Figure 4.10 The robot with the new two platform setup

Stepper Motor Operation:

Initially development of the systems that used steppers was done using an H-Bridge circuit to try to drive the stepper, this did not work. More research determined that stepper motors are commonly driven above their rated voltage and the only safe way of doing this is to use a current limiting stepper driver like the A4988. Using this driver simplified operation of the robot's steppers as it only needed to be sent a step and direction signal, it generates the phases needed to move the motor, with the H-Bridge the Arduino needed to generate phases itself.

When the second platform was added to the robot this increased the weight which stopped the drive steppers being able to move, to overcome this the voltage supplied to the stepper motors was simply increased, this meant they could produce more torque and move the extra weight.

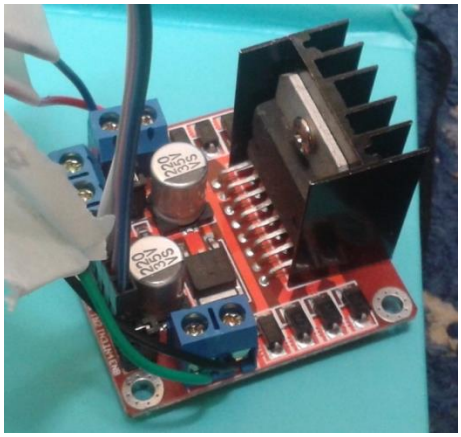


Figure 4.11 H-Bridge originally used for stepper control

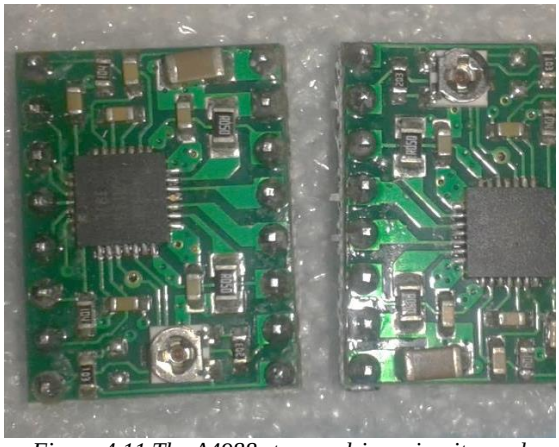


Figure 4.11 The A4988 stepper driver circuits used

CNC Shield Integration:

Initially to interface the stepper motor drivers with the Arduino a CNC shield was used, it could interface four A4988 drivers with the Arduino at once in a small compact size. This worked well until the LIDAR was connected to the Arduino's I2C lines, when in operation via the CNC shield the drivers would cause communication with the LIDAR to constantly crash.

This was most likely caused by an electromagnetic field being created when current flows through the drivers which being so close to the Arduino, interferes with its I2C communication. The solution to this issue was to stop using the shield and opt for a larger breadboard circuit, the increased distance from the Arduino solved the issue.

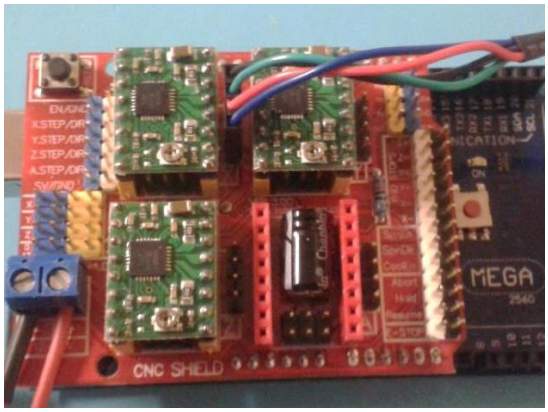


Figure 4.12 The CNC shield the stepper drivers were initially used with

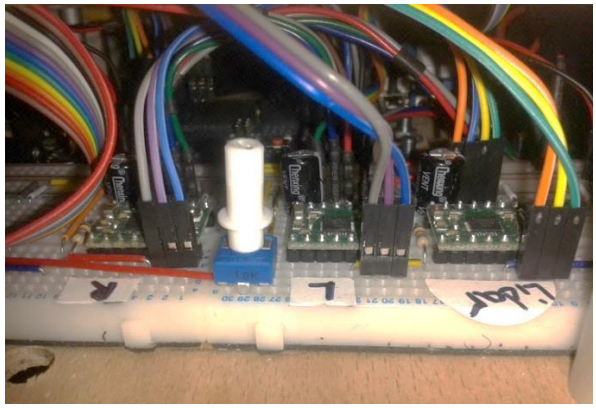


Figure 4.13 The drivers wired up using a breadboard

Integrating IMU System:

The IMU system would have been used to determine the robot's orientation and this information used to make turns. The problem is that integration of the change in angular velocity of the IMU is used to determine orientation, due to its angular velocity never being equal to exactly zero (even when still) these small measurements are integrated overtime causing rapid drifting of the orientation value. This can be filtered with a reference to correct against, like a compass but the MPU-6050 only possesses a gyroscope and accelerometer.

Pre calculating the movements for turns was the other option – this had limited success due to the deformation of the robots tires, which changes the effective circumference of the tire. The tires deform slightly under the weight of the robot and the level of deformation depends on the surface the robot is on. The deformation can be modelled and accounted for on one surface type but on a different surface type, the pre-calculated turns become inaccurate.

Not being able to get the robot to turn accurately on a range of surfaces is the reason behind the decision to simulate the 2D localisation rather than perform it with the physical robot, which would need to be able to turn to move in a 2D map.

Figure 4.14 The robot's tire with no weight applied

Figure 4.15 The robots tire under the weight of the robot

LIDAR System:

The key issue in the development of the LIDAR system was how the module would be rotated so that it could take measurements in 360°, a number of physical setups were considered and experimented with but two main setups were constructed:

- Rotate the LIDAR module directly
- Rotate a mirror around the LIDAR module

The first method required the inclusion of a slip ring so that wires could connect the control unit to the LIDAR module while it spins, making it electronically more complex. The second method required a large mirror and way of holding the mirror above the LIDAR at the correct angle making it more mechanically complex. The option to rotate the LIDAR instead of the mirror was made due to the mirror setup being much larger and fragile compared to simply rotating the module. In addition, the LIDAR uses pulse and receive optics for measuring distance, the positions of these in the mirrors' reflection changes as it rotates – changing the measured distance slightly at every angle.

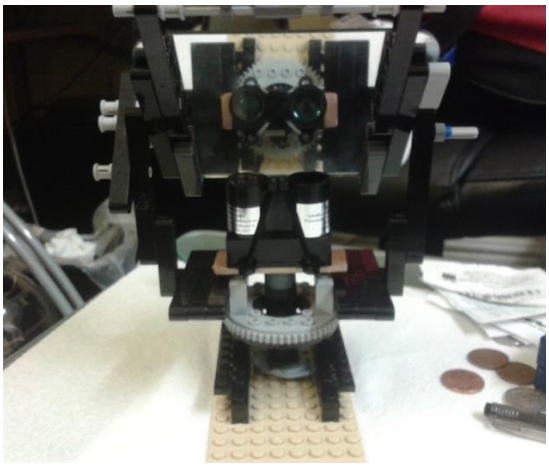


Figure 4.16 Rotating a mirror above the LIDAR to measuring using light reflected by the mirror

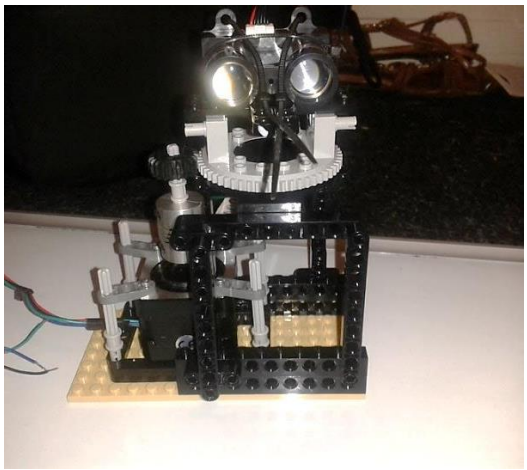


Figure 4.17 Rotating the LIDAR module directly and using a slip ring to connect its wires to the control unit

See Appendix 4 for a collection of images showing the fully built robot.

4.2 One Dimensional Localisation Software

4.2.1 Acknowledgement & Sources

Library / Idea / Work	Usage
Processing Environment (Reas and Fry 2016)	Used as the framework on which the user interface and visualisation is built upon.
Udacity CS 373 : AI For Robotics (Georgia Institute of Technology 2013)	This implementation's algorithms used for the Motion and Sensor update steps are based on examples presented in this course.
giCentre Utils (gi Centre 2015)	This third party processing library was used to create the histogram visualisation and draw it on screen

Table 4.3 Libraries & examples used in implementation and what they were used for

4.2.2 Aims Achieved

The project specification did not have explicit aims for what the one-dimensional localisation software would operate like, but through its implementation, the project aim of localising the robot in one dimension has been successfully achieved. The implementation works very well and Appendix 3 shows an example of its operation.

4.2.3 Development Approach & Methods

A key development approach that was taken was using a simple software interface to control the physical robot, this was done through the creation of a ***Robot*** class. The class possesses methods related to what the physical robot can do ***connect***, ***move***, ***checkForDoor*** etc. This makes control of the robot very simple and it can be extended easily to give access to more robot functionality.

In creating this interface to control the robot with, the rest of the development took a bottom-up approach, initially the **Robot** class was implemented and tested to ensure proper control of the robot. The **Map** class could then be built on top of this, updating itself according to the movement and sensor readings from the robot. Finally, the **CoreUI** class could then be built on top of this, visualising the probability distribution contained in the map and by taking user control commands and sending them to the Robot class by calling the corresponding method. The same sequence can be taken to add new features to the software.

Through development, emphasis was put on making the software flexible, in this case flexibility refers to the parameters that are put into the grid localisation algorithm. Functionality was added to the CoreUI class to allow parameters that affect the algorithm to be changed easily while the software is running. This is much more flexible and intuitive compared to using command line arguments or default "hard-coded" values.

4.2.4 Operational Overview

Figure 4.18 presents the basic grid localisation algorithm, information sourced from Thrun, Fox and Wolfram (2005, p239) , the ODL software uses a more specific version of this that modifies how the motion and measurement models are applied.

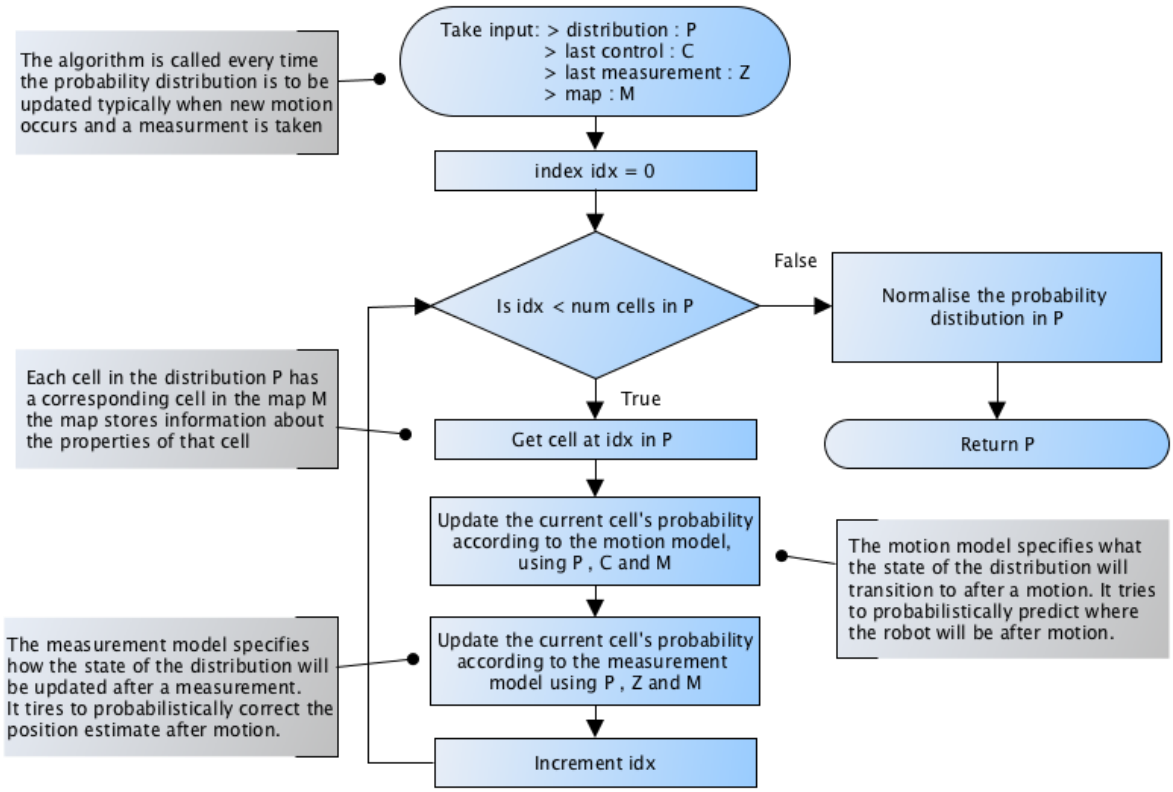


Figure 4.18 The basic Grid Localisation algorithm

Figure 4.19 presents the implementation of the grid localisation used in the ODL software, the pseudo code in figure 4.20 and 4.21 covers in depth the operation of the sensor update and motion update steps respectively. It is assumed the robot can only move to the right or left in the world and that it can sense the doors, walls and ends of the world.

The implementation makes use of a simple sensor and motion model presented in Udacity Course CS 373 (Georgia Institute of Technology 2013), in the motion model the robot has a probability of undershooting, overshooting or arriving exactly at its intended target this is used to probabilistically predict what grid cell the robot will be in after a motion. The sensor model specifies multiples that are used to raise or lower the probability that the robot is in a given cell given measurements from the sensor. These simple motion and sensor models were used as they are suitable at this level and developing models that are more specific are outside the scope of the project.

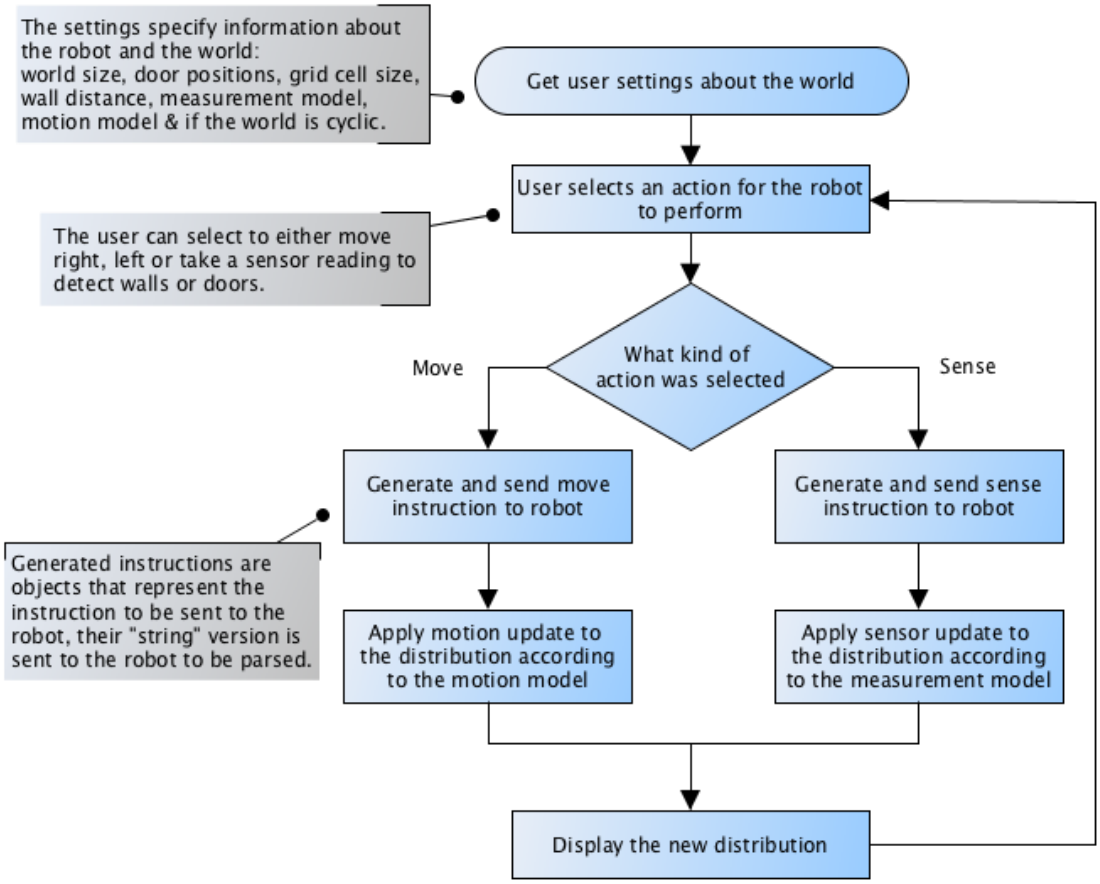


Figure 4.19 The version of the Grid Localisation implemented in the one-dimensional localisation software

SensorUpdate: - increase the belief of cells in a one dimensional grid who's properties match the given measurement, decrease the belief of cells which do not match the measurement

Parameters: *grid*, *measurment*, *hitMultiple*, *missMultiple*
Output: a new grid with an updated probability distribution

```
   /* a GridCell posses a belief and a doorState                                    */
 1 newGrid ← empty list of type GridCell;
 2 for i ← 0 to len(grid) do
       /* newCell's are properties copied from original corresponding cell in grid,
          only it's belief will be modified                                         */
 3     newCell ← grid.getCell(i);
       /* use hit & miss multiples to scale belief according to measurement       */
 4     if measurement = currentCell.doorState then
 5         newCell.belief ← newCell.belief × hitMultiple;
 6     else
 7         newCell.belief ← newCell.belief × missMultiple ;
 8     end
 9     newGrid.append(newCell);
10 end
   /* distribution in newGrid is normalised to sum to 1                             */
11 normalise(newGrid);
12 return newGrid
```

Figure 4.20 Pseudo code for the algorithm used during the "Apply Sensor Update" step

MotionUpdate: - move probability distribution in a one dimensional grid according to motion of the robot

Parameters: *grid*, *numCells*, *direction*, *cyclic*, *pExact*,*pOver*, *pUnder*
Output: a new grid with an updated probability distribution

```
   /* a GridCell posses a belief and a doorState                                   */
 1 newGrid ← empty list of type GridCell;
 2 gridLen ← len(grid);
 3 if direction = LEFT then
 4 |   numCells ← −numCells;
 5 end
 6 for i ← 0 to gridLen do
   |   /* calculate indices of all places robot may have come from               */
 7 |   if cyclic = TRUE then
   |   |   /* wrap indices to stay within length of grid                         */
 8 |   |   indexExact ← (i − numCells) mod gridLen;
 9 |   |   indexOver ← (i − numCells + 1) mod gridLen;
10 |   |   indexUnder ← (i − numCells − 1) mod gridLen;
11 |   else
12 |   |   indexExact ← i − numCells;
13 |   |   indexOver ← i − numCells + 1;
14 |   |   indexUnder ← i − numCells − 1;
15 |   end
   |   /* get the cells from the grid using the indices we calculated, getCell;
   |   gets an empty GridCell with belief 0 for out of range indices             */
16 |   exactCell ← grid.getCell(indexExact);
17 |   overCell ← grid.getCell(indexOver);
18 |   underCell ← grid.getCell(indexUnder);
   |   /* calculate the probability of arriving from each cell                   */
19 |   exactMotionProb ← exactCell.belief × pExact;
20 |   overMotionProb ← overCell.belief × pOver;
21 |   underMotionProb ← underCell.belief × pUnder;
   |   /* newCell's are properties copied from original corresponding cell in grid,
   |      only it's belief is updated                                            */
22 |   newCell ← grid.getCell(i);
23 |   newCell.belief ← exactMotionProb + overMotionProb + underMotionProb;
24 |   newGrid.append(newCell);
25 end
26 if cyclic = TRUE then
   |   /* distribution in newGrid is normalised to sum to 1                      */
27 |   normalise(newGrid);
28 end
29 return newGrid
```

Figure 4.21 Pseudo code for the algorithm used during the "Apply Motion Update" step

4.2.5 Key Issues

Serial Communication:

Communication with the Robot was done using a virtual serial port over a Bluetooth connection, each end can wait for and read data from the port or write to the port. Processing uses a draw method that is called repeatedly to update the display, all of the methods called in the ODL software are called at some point in the draw method, the draw method runs on a single thread.

A problem arose when the Robot class' communication functioned by waiting using a while loop until data was available (**port.available()**) to read from the serial port. Due to this method being called in the draw method, it is not aware of new data coming until the draw loop runs again blocking and creating an infinite loop where the robot waits for serial data forever.

To solve this the method **serialEvent(port p)** was implemented which Processing calls whenever a serial port receives data, this is done on a separate thread from the draw method. **serialEvent** is used to build up semi-colon terminated strings received from the serial port and place them into a thread-safe blocking-queue, the robot's communication methods then wait for strings to become available from the blocking queue instead. This solves the infinite loop issue as a timeout period can be set for how long to wait.

Robot's Motion:

The robot is moving in one dimension, in this implementation it is horizontal motion across the floor with cardboard squares being used to represent the walls in a corridor and gaps representing doors in the corridor. In a perfect world the robot can only move forward and backward in the corridor in a straight line as this is the only dimension that the robot is localising in. In reality when moving, the robot makes very slight uneven wheel slippages that over time build up and make the robot veer slightly to the right or left. This means that it will eventually go off course, outside of the map.

There is no solution to this issue when localizing in one dimension, as to the robot it is only moving forward and backward and can only detect doors or walls, it has no concept of a second dimension that allows movement to the left or right. The solution to this problem would be to perform localisation in two dimensions, using the distance to objects to localise rather than detecting walls or doors.

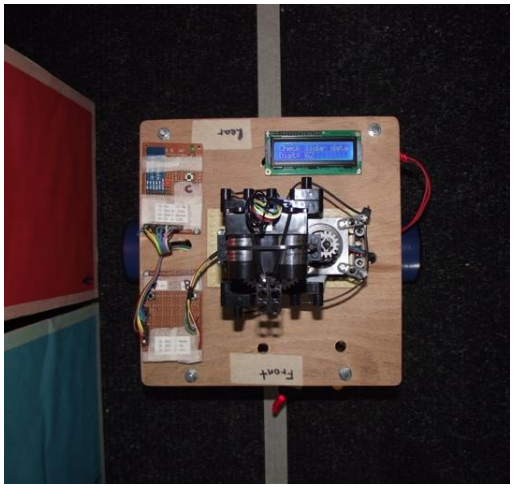


Figure 4.22 The robot prior to any movement, the robot is straight on the line

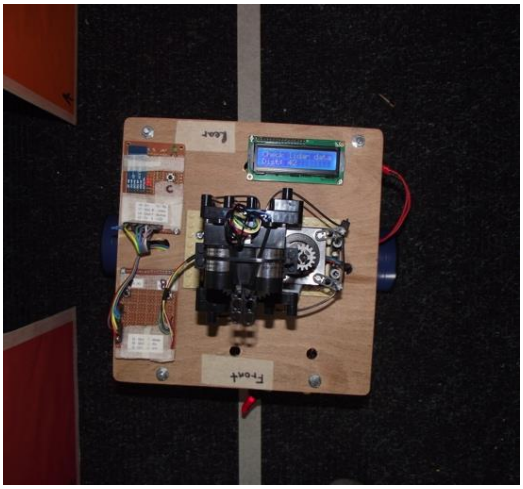


Figure 4.23 After a number of motions, the robot is no longer straight on the line

4.3 Two Dimensional Localisation Simulator

4.3.1 Acknowledgement & Sources

Library / Idea / Work	Usage
Processing Environment (Reas and Fry 2016)	Used as the framework on which the user interface and visualisation is built upon.
Udacity CS 373 : AI For Robotics (Georgia Institute of Technology 2013)	This implementation of a particle filter is based on a simple particle filter example presented in this course.

Table 4.4 Libraries and examples used in implementation and what they were used for

4.3.2 Aims Achieved

An advanced goal of the project was to perform 2D localisation with the physical robot this was not achieved, in its place a simulation of 2D localisation was implemented performed in a simplified virtual world. This implementation was very successful in the way in which it visualises localisation, Appendix 5 shows an example of its operation.

4.3.3 Development Approach & Methods

The two dimensional localisation software makes use of the Monte Carlo localisation method to localize a robot in a simulated environment, as mentioned earlier a simulator was used due to issues with the robot's turning, but firstly building a simulator would be the natural route to take before implementing the method with a physical robot.

With the software being a simulator realism was a key requirement. Originally the software was going to work by simulating map geometry and the robot would cast rays to detect the distance to said geometry and localise, however this initial implementation of the software uses a simpler model with explicit landmark points in the world that the robot can detect the distance to. The more realistic ray casting model can be implemented in future. Other features that are present to add realism to the simulation are a limit to the range of the landmark sensor as well as Gaussian noise that is added to the movement and measurements that the robot makes just like there would be in the real world.

Just like the ODL software, development took a bottom-up iterative approach, focus was firstly put on getting the core algorithms to work. Testing was done using only console output then adding very simple visualization once the core algorithms were stable. The addition of visualization made it possible to “visually test” the software to detect and fix some issues with the resampling method used in the software. Improved quality of visualization was then added on top of this and more advanced features like the sensor range limit and adding landmarks at runtime.

4.3.4 Operational Overview

In the simulation the robot is limited to a world size defined in pixels and can only use landmark points to localise, the distance to these point is calculated using geometry and noise in the robot's movement and sensing is simulated using Gaussians, the mean being the measurement and the variance being the amount of noise. The simulation is based on a simpler example of Monte Carlo Localisation presented in Udacity Course CS 373 (Georgia Institute of Technology 2013)

Figure 4.24 presents how the two dimensional localisation software implements Monte Carlo Localisation. The supporting pseudo code in figure 4.25 shows how the critical resampling step is performed.

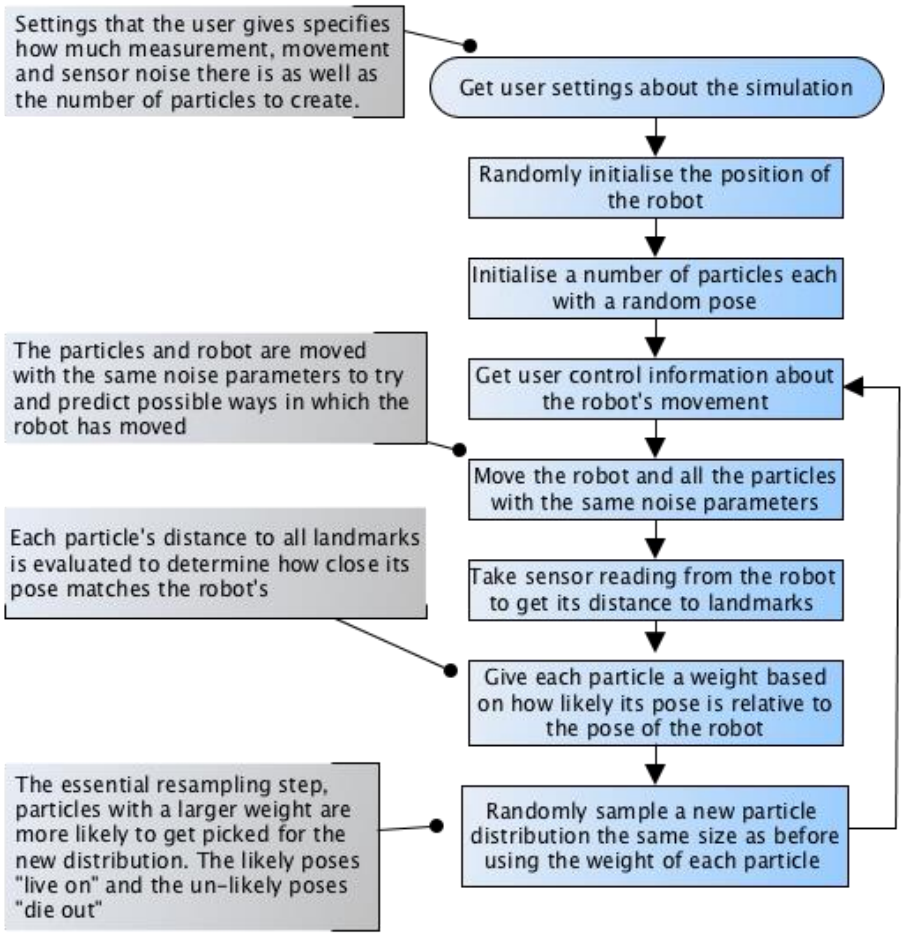


Figure 4.24 How the two dimensional localisation simulator operates

ResampleParticles: - Create a new sample of particles according to the weight of the particles in the given list, more probable particles are more likely to survive

Parameters: *originalSample*
Output: a new sample of particles

```
1  sampleMap ← empty sorted map, keys of type Floating-Point, values of type Particle
   newParticles ←empty list of type Particle;
2  probabilitySum ← 0
   /* map each particle to a range < 1, the range is the probability particle will
      be picked for the new sample                                              */
3  foreach element currParticle of the list originalSample do
4  |  probabilitySum ← probabilitySum + currParticle.weight;
   |  /* put makes a map entry with key probabilitySum and value currParticle, the
   |     map is sorted according to the keys                                    */
5  |  sampleMap.put(probabilitySum, currParticle);
6  end
   /* randomly pick new particles for next sample using sampleMap               */
7  for i ← 0 to len(originalSample) do
8  |  randVal ← randomFloat();
   |  /* ceilingKey gets the key with the smallest value that is greater than or
   |     equal to randVal (basically the range randVal falls into)             */
9  |  key ← sampleMap.ceilingKey(randVal);
   |  /* get retrieves the particle associated with key                        */
10 |  selectedParticle ← sampleMap.get(key);
11 |  newParticles.append(selectedParticle);
12 end
13 return newParticles
```

Figure 4 25 Pseudo code for the algorithm used during the Resampling step

4.3.5 Key Issues

Resampling with double

As demonstrated in the pseudo code the resampling method works by mapping the particles to a range less than one based on their likelihood, then randomly generating a number between 0 – 1 and picking the particle who's range the random number falls in. The range of each particle is created by iterating over all the particles and summing their weight into one variable. The beginning of the range is the previous weight sum before adding the weight of the current particle and the end of the range is the weight sum after adding the weight of the current particle.

The initial resampling implementation stored the weight sum as a "**double"**, in Java this is a 64-bit floating point number and is subject to rounding error. The issue was caused by the weight of some particles being so small that when added to the weight sum, the addition is rounded down to adding 0, effectively this overwrote the range of the previous particle. This meant that some particles did not get resampled into the new distribution and possibly meant that some particles with very unlikely poses get to stay in the distribution much longer than they should.

The solution was to use the Java class called **BigDecimal** instead of **double,** which can store and perform operations on floating point numbers with arbitrary precision. The software localises much faster with this method.

Kidnapped Robot problem

The kidnapped robot problem refers to a robot that has been localised and is taken from its location to another without being informed of this, the robot then has to re-localise itself to find its position again. (Fox et al. 2001)

In the simulation a similar situation happens when the robot is given few landmarks to localise against. The estimate can sometimes localise in the wrong position, the particles are clustered together and move like the robot but are slightly off the true position, this occurs as readings say the estimate is a plausible position. Due to the resampling method only sampling from particles currently in the distribution, this can take time to correct as other more plausible positions aren't considered when the particles cluster together.

A solution to this would be to set a portion of the resampled particles to be particles with random poses during each resampling step, this ensures that other places in the map get a chance to be resampled into the distribution even if they aren't in the main cluster. This should cause localisation failures to recover more quickly and act as a solution to the kidnapped robot problem.

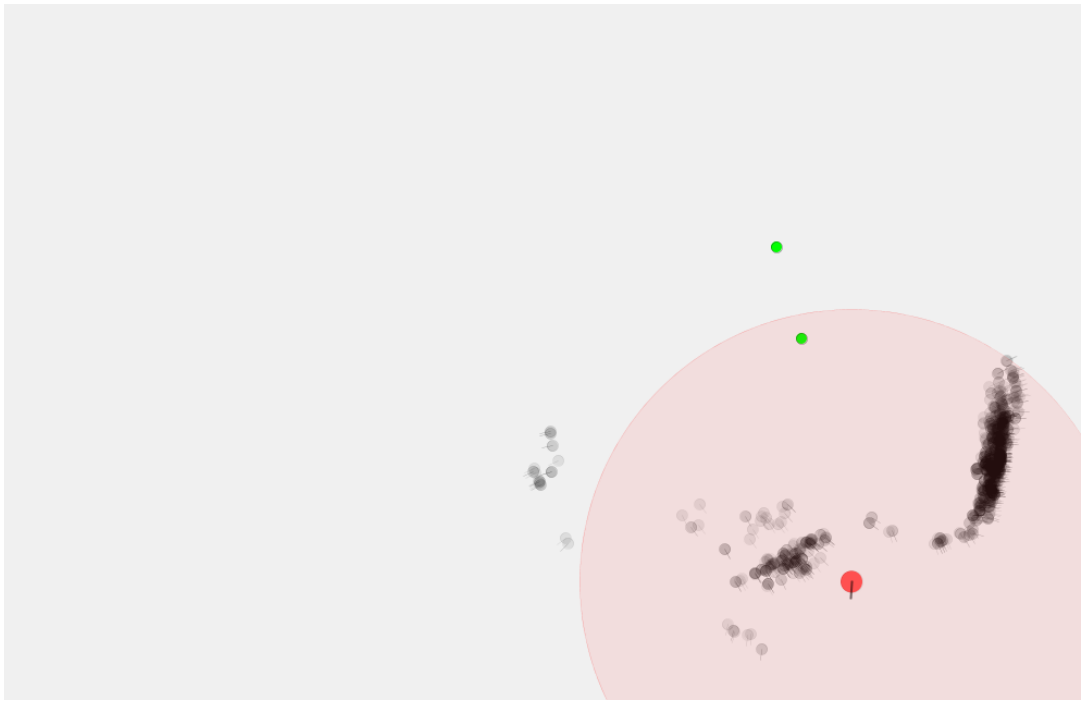


Figure 4.26 Screenshot from TDL software, the particles (black) clustering in a position slightly off from the robot (red), the landmarks are green

5 Conclusions

This section covers some of the conclusions drawn upon completion of the project.

5.1 Project as a Whole

In hindsight the project aims were too broad, it is clear that both robot building and localisation are fields with an almost overwhelming amount of depth. Working on both the robot hardware and localisation software meant that less time was available to spend on either. Should the project be redone it would benefit from focusing specifically on localisation and using a prebuilt robot, allowing exploration of more localisation methods.

Due to a lack of expertise in robot building, the majority of time was spent on getting the hardware working and was the project starting point. Focus should have been put on localisation first using simulations which would help with hardware building by indicating what physical capabilities the robot needed.

Despite these points however the project on a whole was successful as shown in the previous section about the deliverables, most of the project aims were achieved and failures were down to hardware issues.

The modular, bottom up approach to designing the hardware and software worked well combined with an iterative approach to development. This came across as a natural way to tackle a project where understanding is initially limited but grows as the time spent on it increases, lessons learnt from working on later modules were used to improve earlier work.

5.2 Hardware Deliverable

Inaccurate turning is the biggest issue present with the robot, it was caused by limitations in the IMU unit that was to be used as the orientation sensor, with the unit not being able to counter drift it made accurate IMU controlled turns unfeasible. The robot can turn using dead reckoning but reliability of this depends on the surface the robot is on.

The use of steppers as the drive motors was a mistake, this was done to keep the hardware mechanically simple, however a significant amount of time was spent dealing with issues like learning how to use driver circuits and producing enough torque to move the robot since the steppers had no gearing, this outweighed the benefits of mechanical simplicity.

Despite these issues the robot can be described as a good first prototype as it achieves most of the specification aims. Particularly what worked well is the Arduino used as the control unit, it

had plenty of room for hardware and programming it was simple, secondly the remote operation of the robot was not used to its full potential but the concept of offloading to powerful computers would be useful in future more complex applications.

5.3 Software Deliverables

A key limitation in the software deliverables lies in the realism of the localisation simulation software, it uses landmark points that the robot detects the distance to, to localise. In reality the robot constructed would make use of a beam model where the distance to any objects surrounding the robot is detected and used for localisation.

The ODL software is a good implementation of Grid Localisation with the intricacies of controlling a physical robot. It allows user control over the physical robot through a remote Bluetooth connection and provides a simple but intuitive example of grid localisation, another great feature allows changing of parameters used in the algorithm so their significance can be seen and understood easily.

The TDL software is a good simple simulation of Monte Carlo Localisation, despite it being a limited simulation important functions in the algorithm are well understood due to this initial implementation and can be easily built upon to create a more realistic simulation. The visualisation of this is very pleasant to observe and makes understanding the way MCL works very intuitive.

5.4 Suggestions for Future Work

Table 5.1 suggests future work and how it could be implemented.

Suggestion	Implementation Plan
Improve turning accuracy	Use an IMU unit that has a built in magnetometer, or interface a magnetometer to the robot with the previous IMU this would be able to correct the drift in the gyroscope. Use the IMU to control turning.
Change drive motors	Replace the stepper motors with geared continuous DC motors and encoders, this would provide much more torque, higher top speed and more existing robots using similar motors to learn from.

Use beam model in 2D localisation simulator	Create maps in the simulator using basic geometric shapes. Have the simulated robot cast rays of a given length at a number of angles in the form of lines, this would represent a distance sensor scanning at various angles. Use geometry to detect where the rays first intersect with objects in the map, the length of the ray from the robot to the intersection represents the measured distance from its virtual sensor. These distances would then be used to localise the robot.
2D localisation with the robot	Having implemented the beam model in the 2D localisation simulator, the virtual robot would be replaced with a class that allows access to real measurement data and control of the physical robot.

Table 5.1 Future work and improvements for the projects and how this could be done

5.5 Project Learning and Ethical Issues

With this project being about robot construction and localisation ethical issues have not been a major concern, but it is worth mentioning a particular issue related to the sensor technology used (LIDAR) when applying this technology to real world problems. Localisation has many applications the most obvious being helping autonomous robots find where they are so they can perform tasks better, an example being autonomous cars.

The critical ethical issue that surrounds LIDAR based localisation in autonomous cars is safety. Is the sensor technology reliable enough to ensure the safety of passengers in an autonomous car?

Petit and Shaldover (2014) present threats to autonomous vehicles that include "*electromagnetic pulse (EMP), map poisoning, radar confusion, LIDAR confusion, infection of in-vehicle devices, and manipulation of in-vehicle sensors*."

Confusion in the LIDAR used in the project was observed, it reported false readings (zero distance) for objects that are a pale off white colour, likely due to the way the colour reflects light. Should the robot be used in a situation where safety is of concern, solutions to LIDAR confusion would need to be considered and added.

Regarding learning, this project fostered development of a range of skills. It allowed learning about basic mechanics and electronics through the construction of the robotic platform, and getting experience with different types of software development through the creation of embedded software on the robot and application software that resides on traditional computers. It also provided the opportunity to learn how different pieces of technology can be combined together to form a coherent whole and solve a non-trivial problem.

The “full-stack” nature of the project meant that the difficulties in constructing and controlling reliable robot hardware were made explicit through first-hand experience with common robotics issues and the core steps in two methods of localisation could be deconstructed, understood and applied in a real world situation.

This project also exposed the depth of robotics as an area made up of many distinct disciplines and exposed a number of gaps in knowledge that would prove extremely useful in robotics if developed, in particular certain branches of mathematics and physics.

As mentioned earlier, the project’s aims were arguably too broad but a benefit of this has been that it gave a broad introduction to the field of robotics, any future work would firstly involve filling in knowledge gaps. Then a path to developing understanding in a particular area of robotics can be created.

References

ARDUINO LLC (2015). *Download the Arduino Software.* [online]. https://www.arduino.cc/en/Main/Software

ARDUINO.CC (2016). *Arduino Mega*. [online]. https://www.arduino.cc/en/Main/arduinoBoardMega

ARDUINO.CC (2016). *MPU-6050 Accelerometer + Gyro*. [online]. http://playground.arduino.cc/Main/MPU-6050

DSS CIRCUITS (2014). *Arduino I2C Master Library*. [online]. http://dsscircuits.com/articles/86-articles/66-arduino-i2c-master-library

Ep 7: Comparing Arduino and Rasberry Pi. (2013). [YouTube Video]. Directed by James Lewis. AddOhms.

FOX, Dieter, et al. (1999). Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. *AAAI Conference on Artificial Intelligence,*.

FOX, Dieter, et al. (2001). Particle Filters for Mobile Robot Localization. In: *Sequential Monte Carlo Methods in Practice*. Springer New York, 401-428.

FUENTES-PACHECO, Jorge, et al. (2012). Visual Simultaneous Localization and Mapping: A Survey. *Artificial Intelligence Review*, **43** (1), 55-81.

GEORGIA INSTITUTE OF TECHNOLOGY (2013). *Artificial Intelligence for Robotics.* [online]. https://www.udacity.com/course/artificial-intelligence-for-robotics--cs373

GI CENTRE (2015). *giCentre Utilities.* [online]. http://www.gicentre.net/software/#/utils/

HOBBYCOMPONENTS (2014). *HC-05 Master Module*. [online]. http://hobbycomponents.com/wireless/432-hc-05-master-slave-bluetooth-module

JONES, Joseph, et al. (1998). *Mobile Robots: Inspiration to Implementation*. 2nd Edition ed., A K Peters.

MCCAULEY, Mike (2010). *AccelStepper library for Arduino.* [online]. http://www.airspayce.com/mikem/arduino/AccelStepper/index.html

MCCOMB, Gordon (2011). *Robot Builder's Bonanza*. 4th Edition ed., McGraw-Hill.

MICROCONTROLS.ORG (2015). *Ultrasonic Range detector using Ultrasonic Sensor & Arduino – Tutorial & Sorce Code*. [online]. Last accessed 18 September. http://www.microcontrols.org/ultrasonic-range-detector-using-ultrasonic-sensor-arduino-tutorial-sorce-code/

MODMYPI (2013). *What's The Difference Between DC, Servo & Stepper Motors?* [online]. Last updated 25 August. http://www.modmypi.com/blog/whats-the-difference-between-dc-servo-stepper-motors

NEHMZOW, Ulrich (2003). *Mobile Robotics, A Practicle Introduction*. 2nd Edition ed., Springer-Verlag London.

PETIT, Jonathan and SHLADOVER, Steven (2014). Potential Cyber Attacks on Automated Vehicles. *IEEE Transactions On Intelligent Transportation System*, **16** (2), 546-566.

POLOLU (2012). *A4988 Stepper Motor Driver Carrier*. [online]. https://www.pololu.com/product/1182

PULSED LIGHT (2015). *LIDAR Lite Basics.* [online]. https://github.com/PulsedLight3D/LIDARLite_Basics

QUIGLEY, Morgan, et al. (2009). ROS: an open-source Robot Operating System. In: *ICRA Workshop on Open Source Software*, Open Source Robotics Foundation, p.6.

REAS, Casey and FRY, Benjamin (2016). *Processing Environment and Standard Libraries.* [online]. Last updated 13 February. https://processing.org/

ROBOTSHOP (2014). *RPLIDAR 360° Laser Scanner*. [online]. http://www.robotshop.com/uk/rplidar-360-laser-scanner.html

ROCKWELL AUTOMATION (2016). *Encoders - Incremental Versus Absolute Encoders*. [online]. http://www.ab.com/en/epub/catalogs/12772/6543185/12041221/12041235/Incremental-Versus-Absolute-Encoders.html#

RUSSELL, Stuart and NORVIG, Peter (2009). *Artificial Intelligence, A Modern Approach*. 3rd Edition ed., Pearson.

SEIGWART, Roland, NOURBAKHSH, Illah and SCARAMUZZA, Davide (2011). *Introduction to Autonomous Mobile Robots*. 2nd Edition ed., MIT Press.

SPARKFUN (2014). *Arduino Uno - R3*. [online]. https://www.sparkfun.com/products/11021

SPARKFUN (2014). *Lidar Lite*. [online]. https://www.sparkfun.com/products/retired/13167
SPARKFUN (2014). *Raspberry Pi - Model B+*. [online]. https://www.sparkfun.com/products/retired/12994
SPARKFUN (2014). *Stepper With Cable*. [online]. https://www.sparkfun.com/products/9238
THRUN, Sebastian, FOX, Dieter and WOLFRAM, Burgard (2005). *Probabilistic Robotics*. The MIT Press.

6 Appendix 1 – Hardware Specifications & Decision Info

LIDAR Lite	
Operating Voltage: 4.75 – 5.5V DC Current Consumption: 100mA Control: I2C or Pulse Width Modulation Max Range: 40m Accuracy: +/- 2.5cm Default Rate: 50Hz	The LIDAR Lite is a cheap LIDAR ranger finder that can be operated from any control unit easily through I2C built into an Arduino board, its range means that it is suitable for indoor operation and not as limiting as ultrasonic sensors, also it is very accurate given its price. Acts as the range sensor that will be used to get distance information, it will be placed on a rotating platform so that the robot can measure all around it without having to move the platform.

Stepper Motors	
Operating Voltage: 12V Current Consumption: 400mA Control: driver circuit Holding Torque: 2100 gm-cm	Stepper motors will be used as the drive system and to operate the spinning LIDAR platform. A pair of stepper motors will be used in a differential wheel configuration and used to drive the robot. Another stepper will use a simple gearing system to control that LIDAR platform and spin it to any given heading to take measurements. The low current draw of these motors means that multiple can be run at the same time from a small power source like a battery.

HC-05 Bluetooth Module	
Operating Voltage: 3.3-5V Current Consumption: 20 - 40mA Default Baud Rate: 9600	A HC-05 Bluetooth module will be used to communicate with a remote computer running the localisation software. Since this module makes use of receive and transmit pins, this is interfaced simply with the Arduino by connecting it to the transmit and receive pins on the Arduino board. The module also has pins that give information about the state of connection, which is useful to have.

MPU-6050 IMU	
Operating Voltage: 3.3-5V Current Consumption: <5mA 3 axis Gyroscope 3 axis Accelerometer	An InvenSense MPU-6050 IMU will be used to track the robot orientation, with it having both an accelerometer and gyroscope it can be used to get data about the robot's motion on many axes, it is cheap, accurate and interfaces with an Arduino using I2C communication.

A4988 Stepper Driver	
Operating Voltage: 8-35V Max Current: 2A Logic Voltage: 3- 5.5V Micro stepping available	Allegro 4988 Stepper Drivers will be used to control the stepper motors used in the robot, one for each motor. It provides a current limiting functionality; this functionality will be used to drive the motor higher than the rated voltage (safely) to produce extra torque from the motors

Arduino Mega 2560	
Microcontroller: ATmega2560 Operating Voltage: 5V Clock Speed: 16MHz SRAM: 8KB Digital GPIO Pins: 54 (15 with PWM) Analog Input Pins: 16 Flash Memory: 256KB	The Arduino Mega will be used as the robots control unit, the large number of pins that it has make it ideal for an application like this, the larger program storage space and SRAM compared to other Arduino allows lots of room for extending the on-board software in future or performing operations which require a relatively large amount of stack space or dynamic memory.

7 Appendix 2 – The Robot's Power System

The robot makes use of two separate power systems, one to power the Arduino control unit itself and another to supply power to the stepper motors used for the drive system and to spin the LIDAR. The Arduino board is powered from a collection of six 1.2V AA batteries in a battery pack, everything connected to the Arduino such as the LIDAR, Bluetooth module, stepper drivers and LCD screen are also power from this source as they get their power from the Arduino board. The motors are powered from a 7.4V 1300mAh Lithium Polymer battery pack with a discharge rate of 20C.

The main reason behind having a separate power supply for the motors is the amount of current the motors use, each motor can use up to 400 milliamps of current, technically the boards 5V supply can supply up to 1 Amp however the AA batteries powering the Arduino cannot continuously supply high amounts of current for very long, they could over heat or melt wires as this is not what they are designed for. It is more likely that the motor simply will not move, as the AA batteries cannot continuously supply enough current to power it. Lastly with the robot having 3 stepper motors the potential current draw could be 1.2 Amps which would exceed the rating of the 5V supply given by the Arduino board.

The Lithium Polymer battery used to power the motors is designed for high discharge high current situations and the 20C rating means it can technically supply 20 times the "C" rating which is the battery capacity, essentially over 20 Amps this can be done continuously for long periods of time as this is specifically what the battery is designed for.

The other reason is that large electrical devices such as motors can case electrical noise that can interfere with smaller delicate components, isolating the power supplies ensures that any electrical noise stays in a circuit without sensitive components.

A final point is that a component called a boost converter is used to up the voltage from the Lithium Polymer batteries to approx. 30V to power the motors, this produces increased torque.

Figure 7.1 The Arduino power supply

Figure 7.2 The motor power supply

8 Appendix 3 – Operation of One Dimensional Localisation Software

These series of annotated images show the operation of the one-dimensional localisation software. The ODL software has simple to use interface shown in figure 8.1, a number of option are down the right hand side, a histogram representing the probability distribution is in the centre with the control buttons below it, below that is a console for displaying messages.

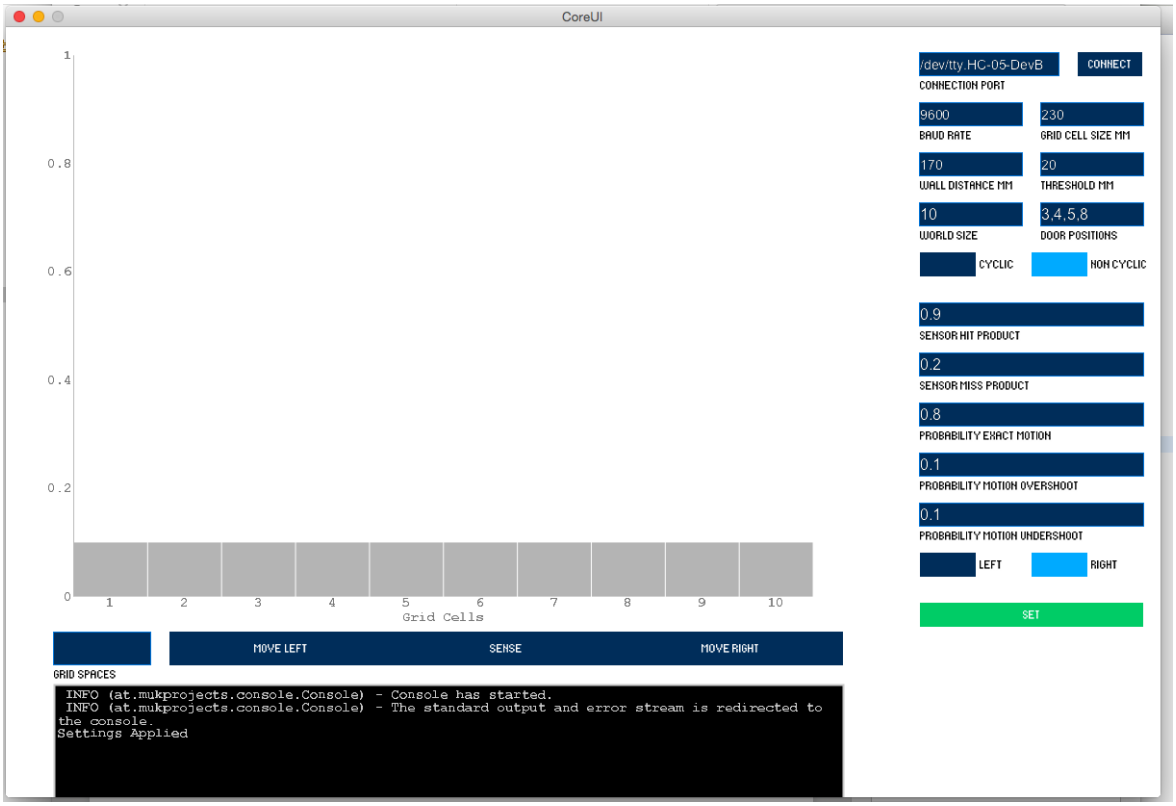


Figure 8.1 The one-dimensional localisation software's user interface

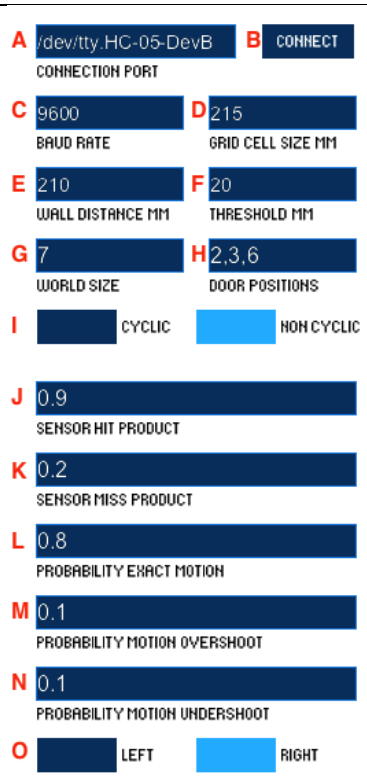


Figure 8.2 Labelled Settings panel

Settings Explained
A- string name of the Bluetooth port to connect to B- button to attempt connection to the robot C- connection baud rate to use D- The physical length of each grid cell in millimetres E- How far the robot is away from the cells with walls F- A threshold for detecting doors or walls G- The amount of physical grid cells there is H- The positon of each door in the grid (1 based index) I- Decide if the world is cyclic (rolls over) or not J- Value to multiply a cell's belief by if the cell properties matches the measurement taken by the sensor. K- Value to multiply a cell's belief by if the cell properties doesn't match the measurement taken by the sensor. L- The probability the robot will move exactly to the intended cell M- The probability the robot will move and overshoot the intended cell by 1 N- The probability the robot will move and undershoot the intended cell by 1 O- In the physical world are the door to the right or left of the robot The set button applies all of these settings to be used in the localisation algorithm.

Figure 8.3 The robot's one dimensional world

Figure 8.3 shows the one-dimensional world that the robot is in each cell has a number and doors are the empty spaces where as walls are represented by the coloured cardboard. Figure 8.4 shows how the ODL software is once initialised with the settings in figure 8.2.

Figure 8.4 The ODL software UI once being initialised with used provided settings

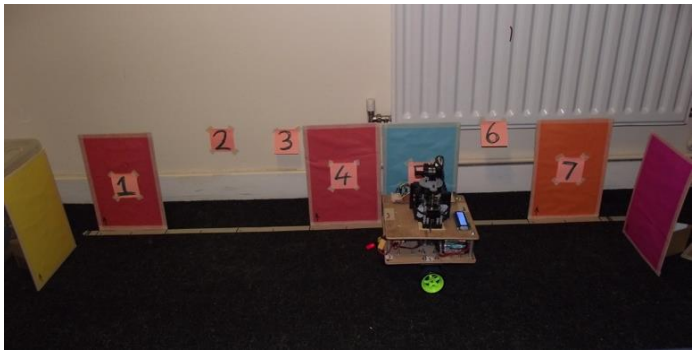


Figure 8.5 Robot on initialisation

Figures 8.5 and 8.6 show the robot and the probability distribution that represents the location estimate on initialisation. Note how the distribution is initially uniform showing that the software in uncertain of the robot's positon, meaning there it is equally likely that the robot is in any cell at that point in time.

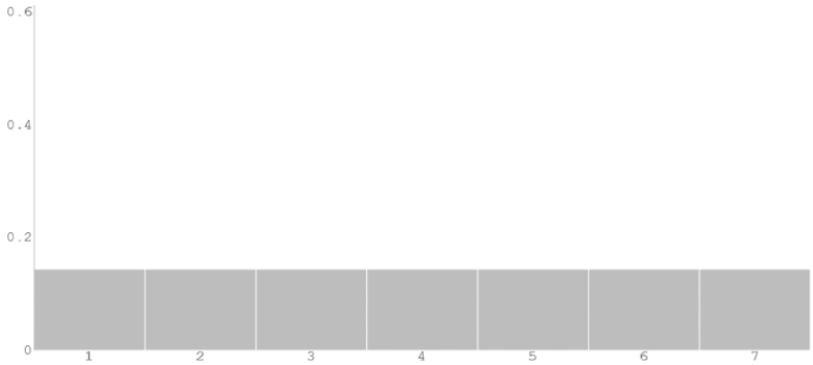


Figure 8.6 Distribution on initialisation

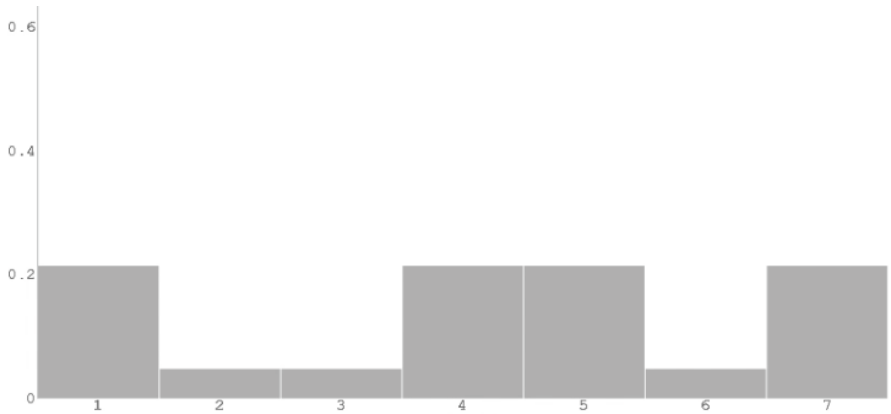


Figure 8.7 Distribution after the first sensor measurement

The first action made is to take a sensor reading, the sensor detects that the robot is next to a wall and raises the probability of the cells which have a wall. Figure 8.7 shows this.

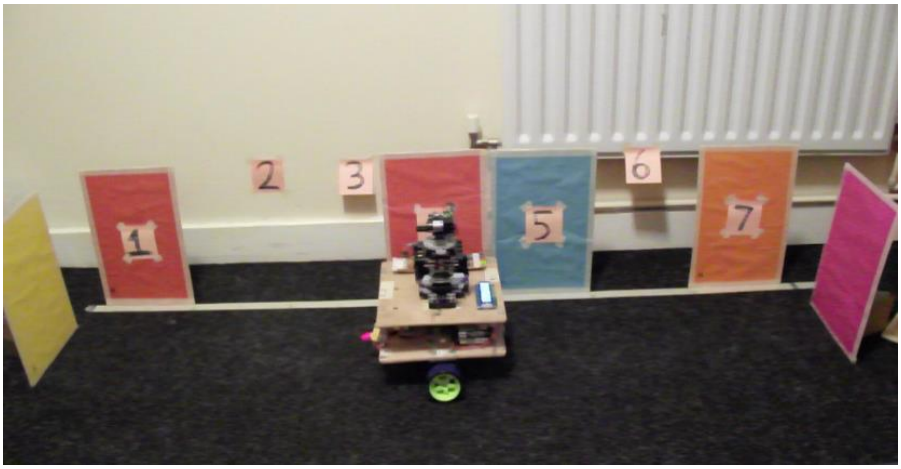


Figure 8.8 The robot after moving a single grid space to the left

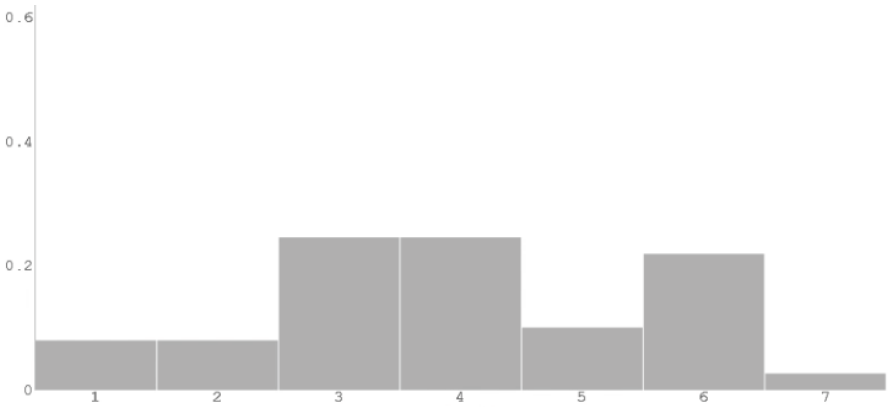


Figure 8.9 The probability distribution after moving one space to the left

Figures 8.8 and 8.9 show the robot and the probability distribution after the robot has moved to the left by one. Note how the probability distribution has shifted in the direction of motion.

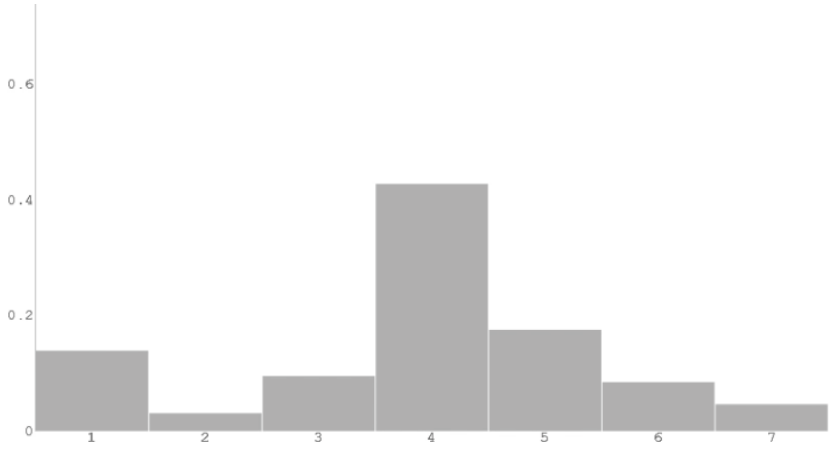


Figure 8.10 Distribution following motion to the left and a sensor measurement

Figure 8.10 shows the distribution after a sensor measurement was taken following the motion to the left, the sensor detects a wall again so the probability of cells with a wall rises. The software has a best estimate as to where the robot might be.

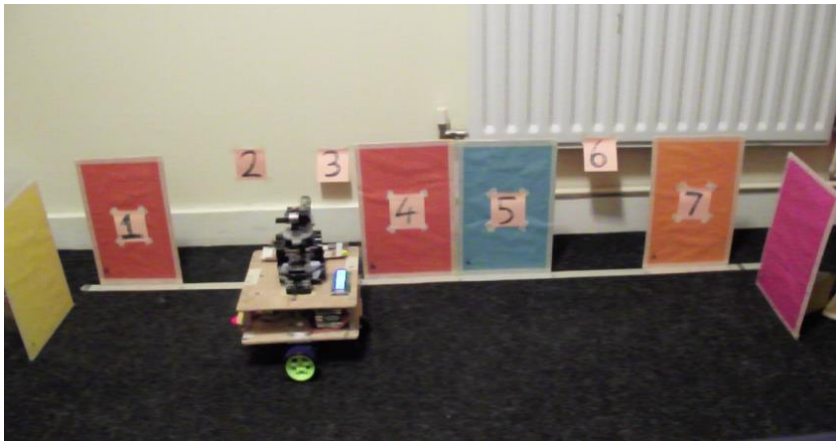


Figure 8.11 Robot after moving another grid space to the left

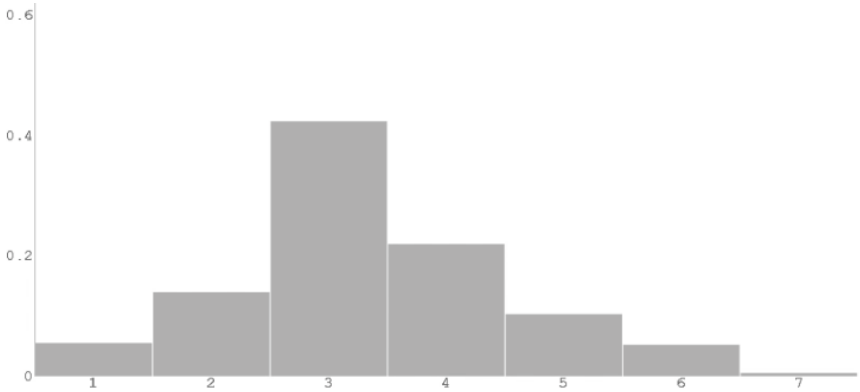


Figure 8.12 The probability distribution after the movement to the left

Figures 8.11 and 8.12 show the robot and the probability distribution after moving to the left again by one grid space. Note how the distribution has shifted and the robot has become less certain about where it is.

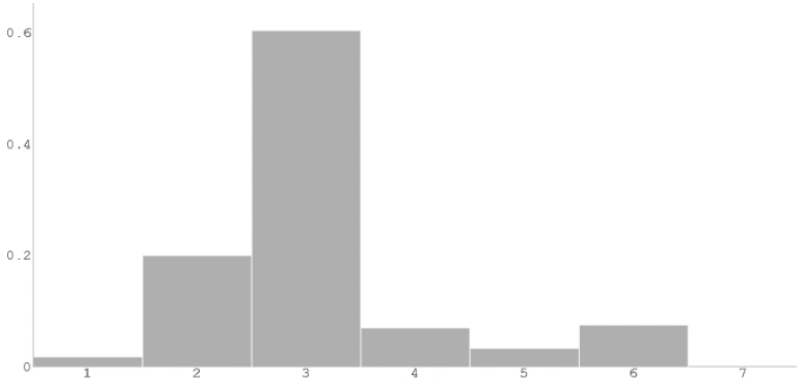


Figure 8.13 The probability distribution after motion and a sensor measurement

Figure 8.13 shows distribution after another taking sensor measurement after the previous motion, a door was detected this time so the probability of all cells with doors is raised. These events significantly increase the robot's certainty of where it is.

9 Appendix 4 – The Finished Robot

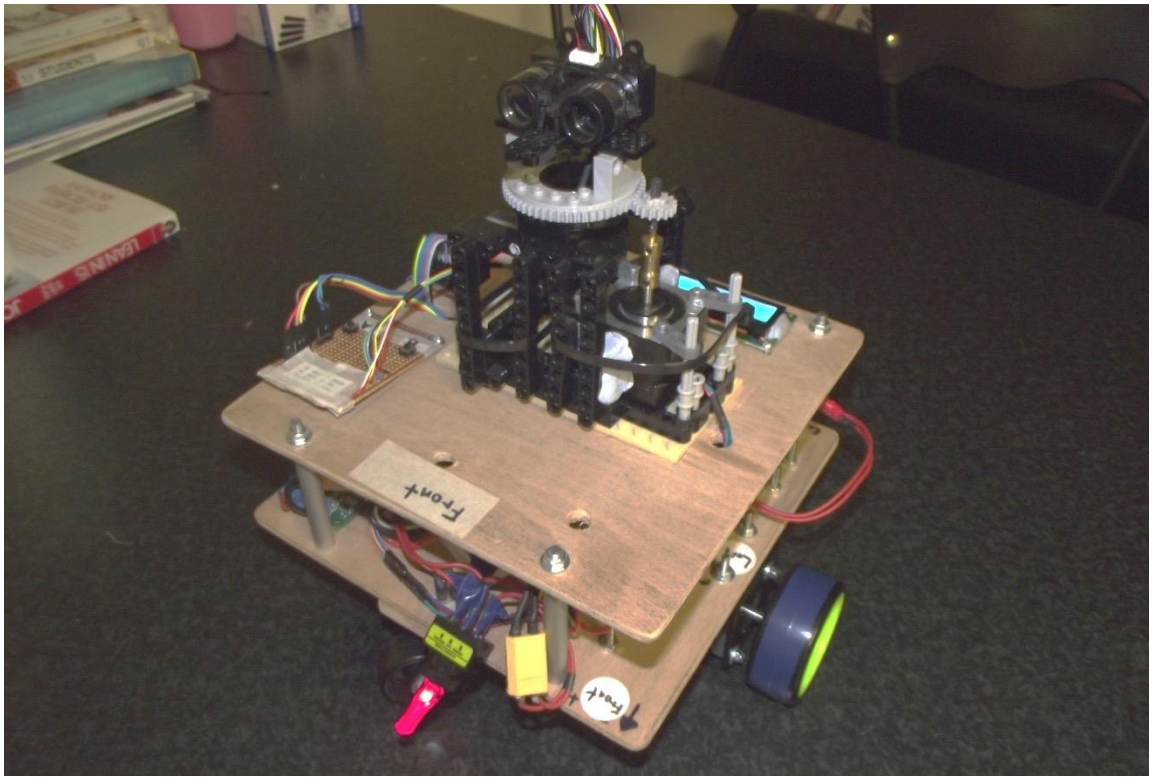


Figure 9.1 The robot from a high angle

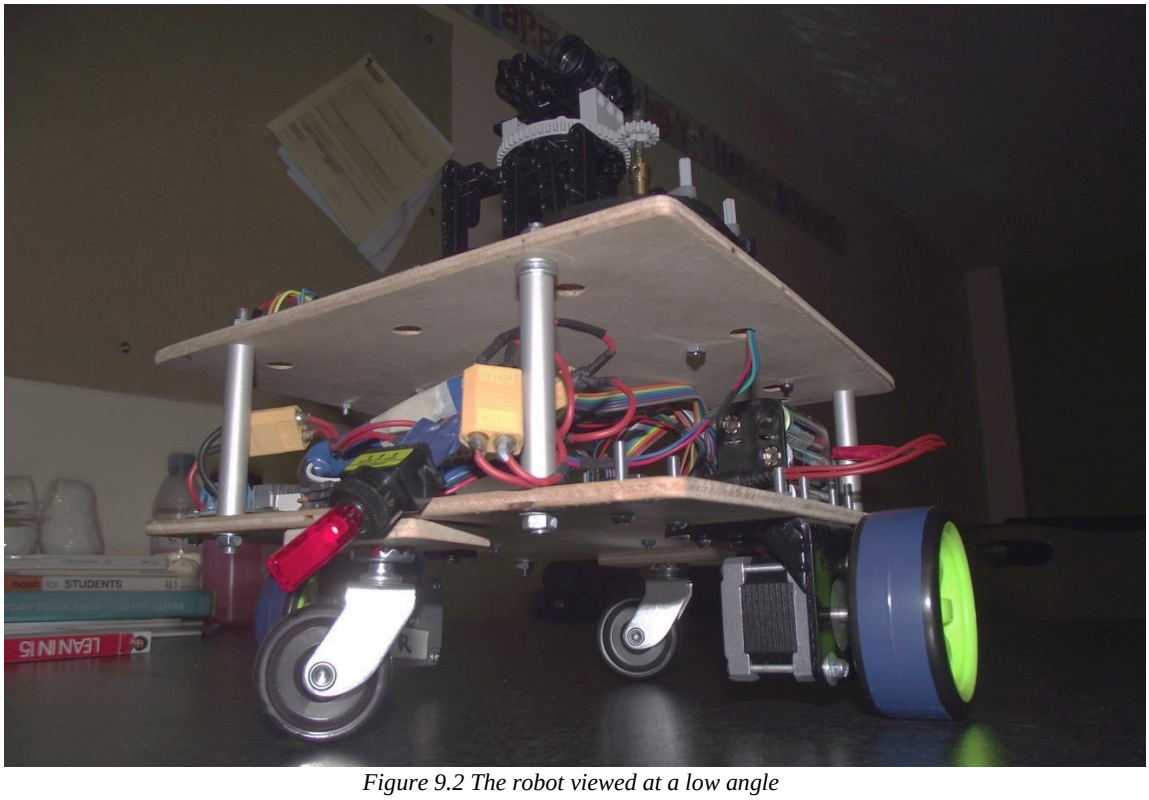


Figure 9.2 The robot viewed at a low angle

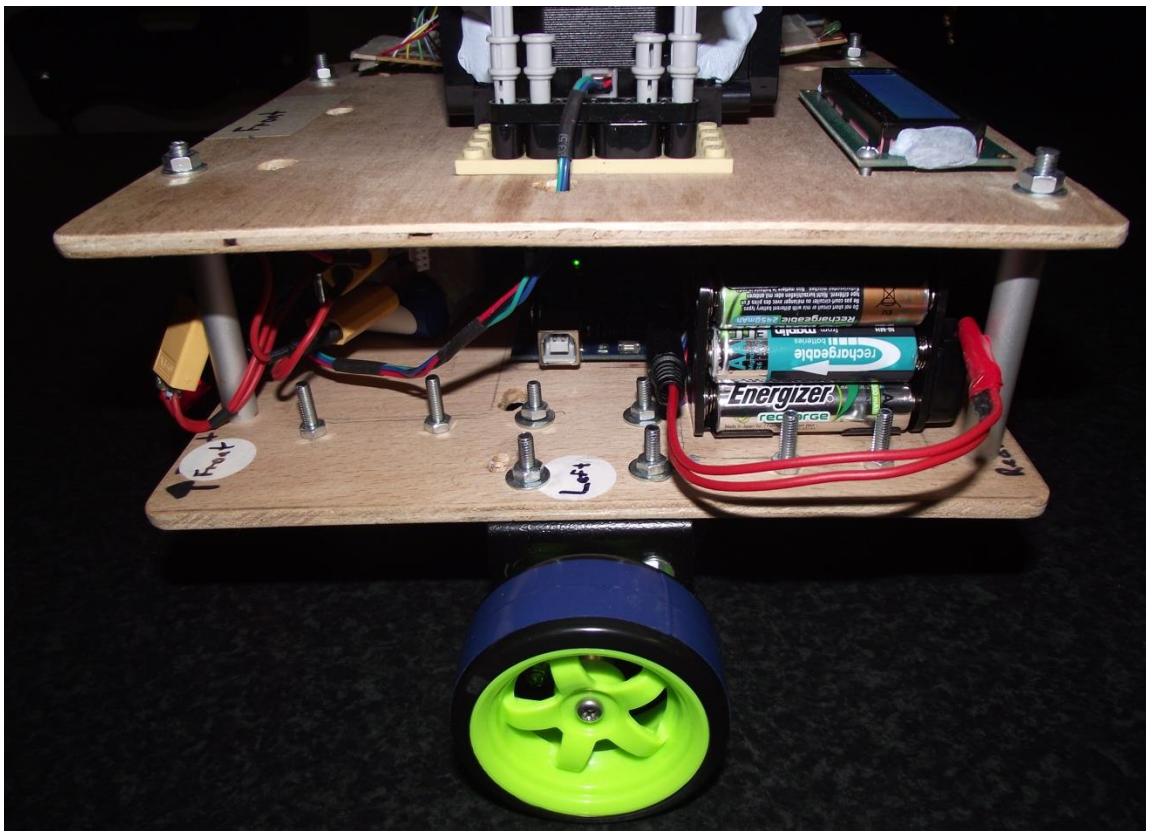


Figure 9.3 Left side of the robot battery pack that powers the Arduino is visible as well as the programming port

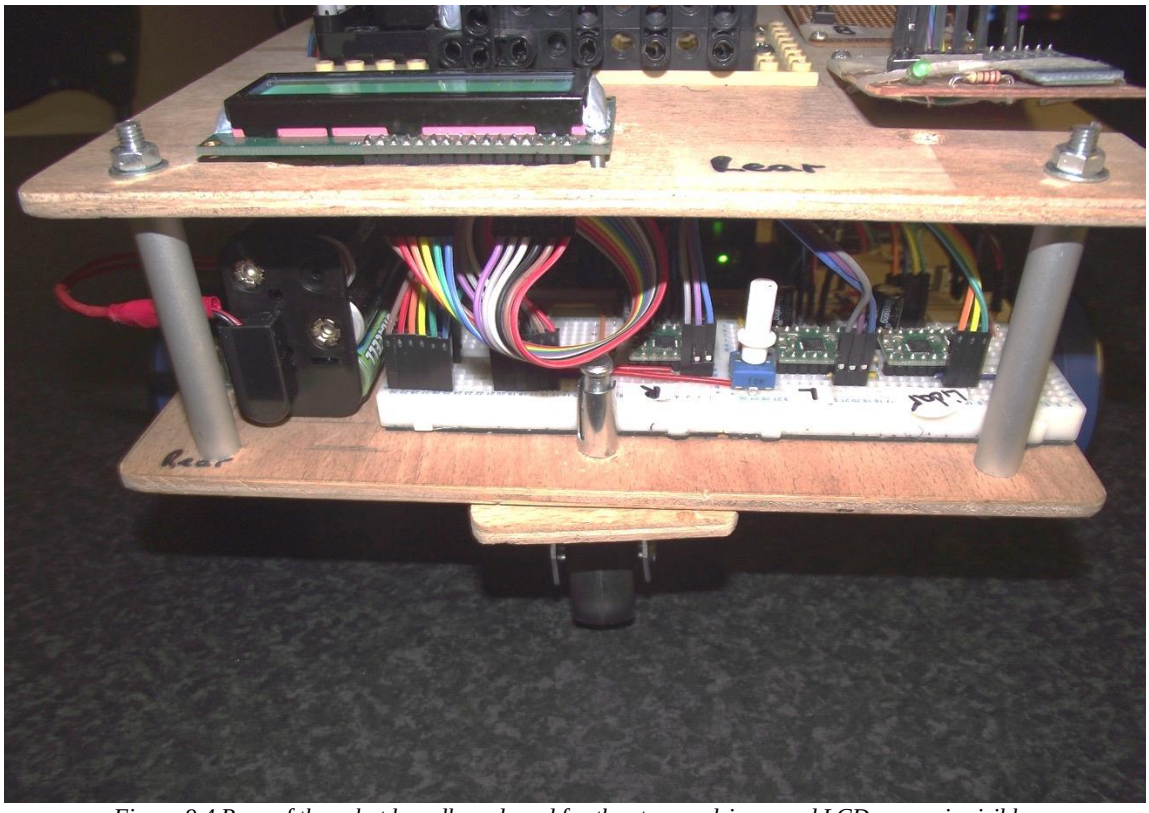


Figure 9.4 Rear of the robot breadboard used for the stepper drivers and LCD screen is visible

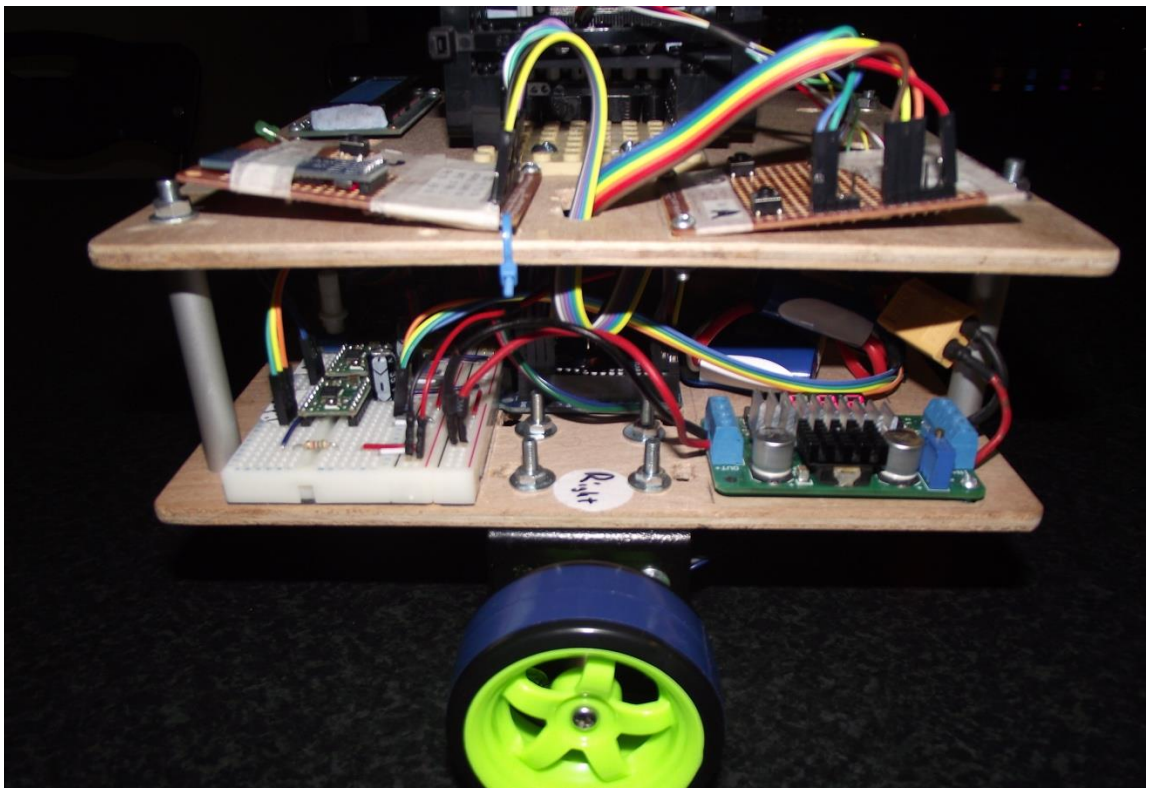


Figure 9.5 Right side of the robot, note the boost converter in the bottom right that regulates power for the steppers

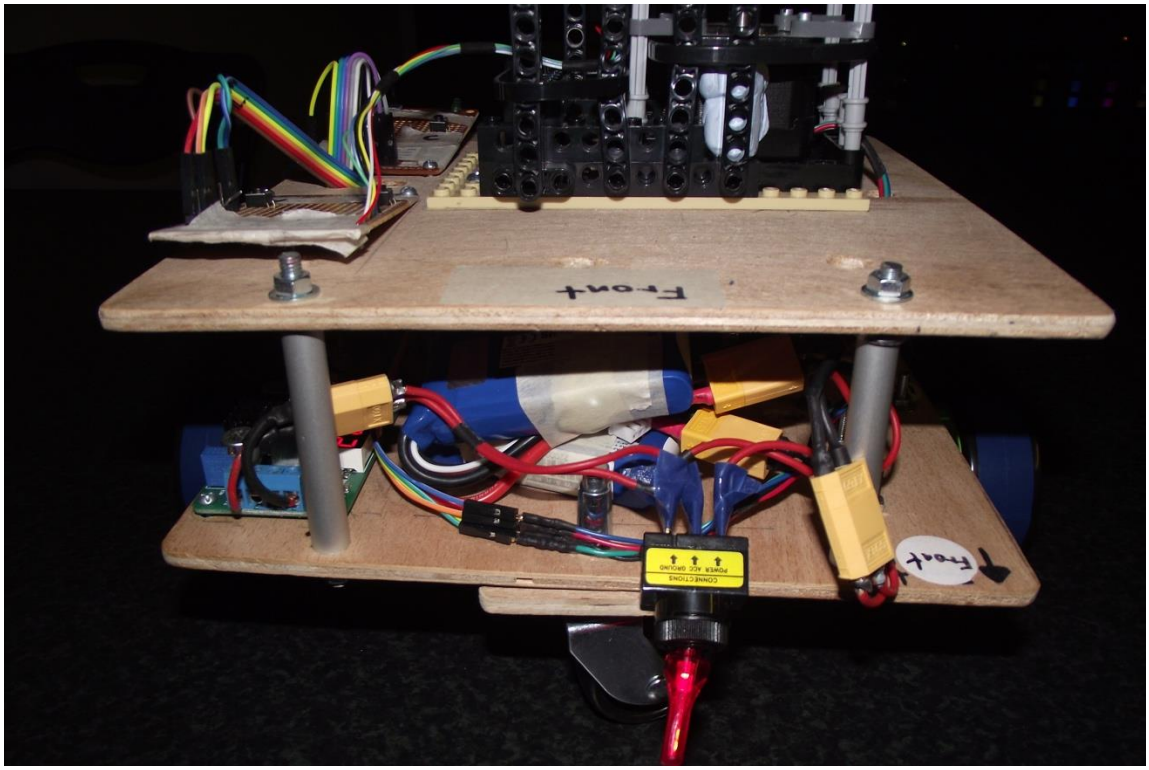


Figure 9.6 Front of the robot, note the Lithium Polymer battery packs and power switch use to power the motors

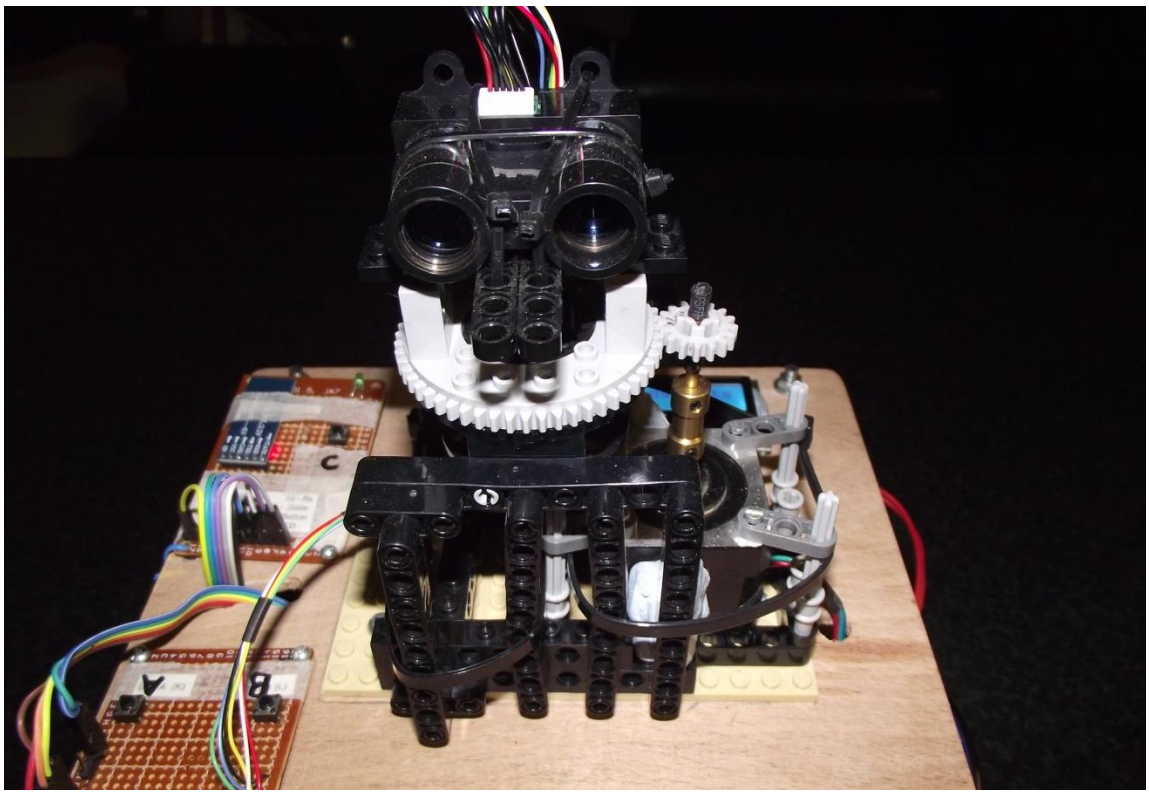


Figure 9.7 Close up of the robot's LIDAR system on the top platform

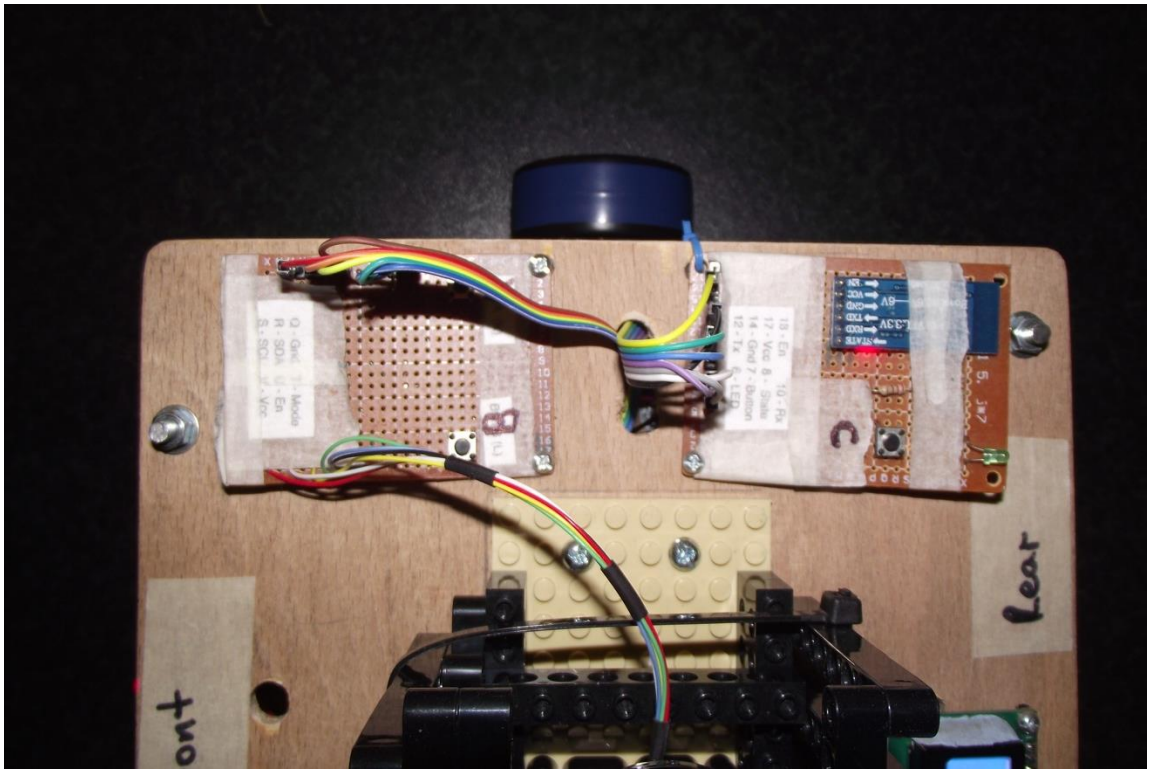


Figure 9.8 The LIDAR proto-board interface on the left, and the Bluetooth proto-board interface on the right

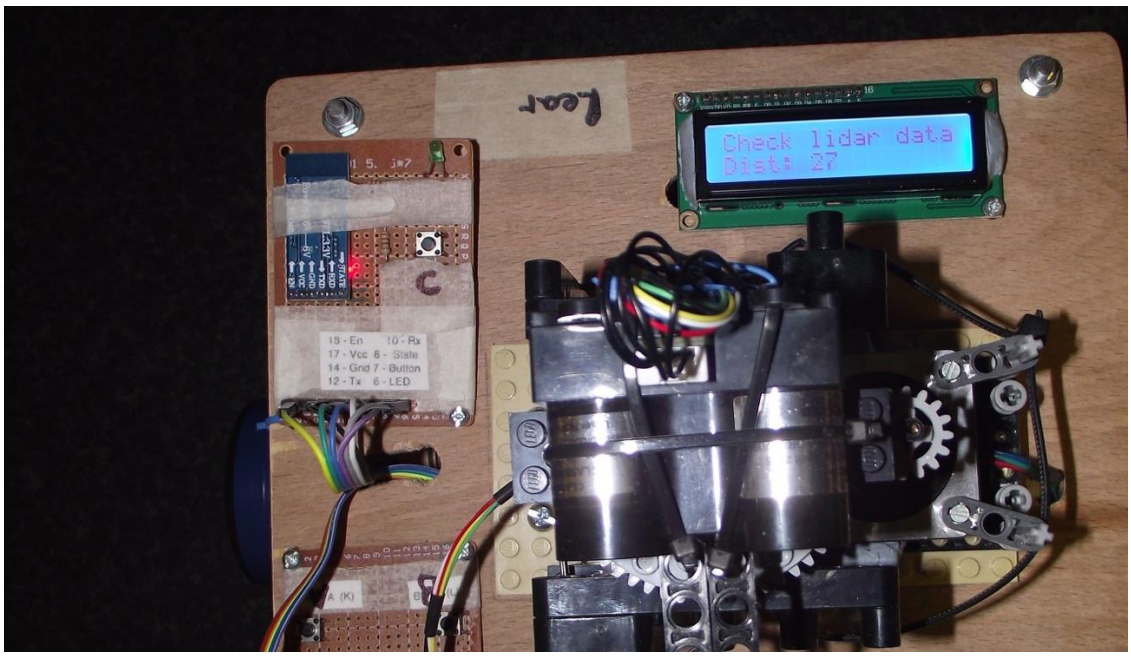


Figure 9.9 Top of the robot note the LCD screen in the top right

10 Appendix 5 – Operation of Two Dimensional Localisation Simulator

This short series of images show the operation of the two dimensional simulation software, that uses the Monte Carlo localisation algorithm. The user interface is simple with customisable options that affect the way the algorithm operates. The visualisation of the simulation is in the centre of the screen and the options down the right hand side of the screen. This is shown in figure 10.1

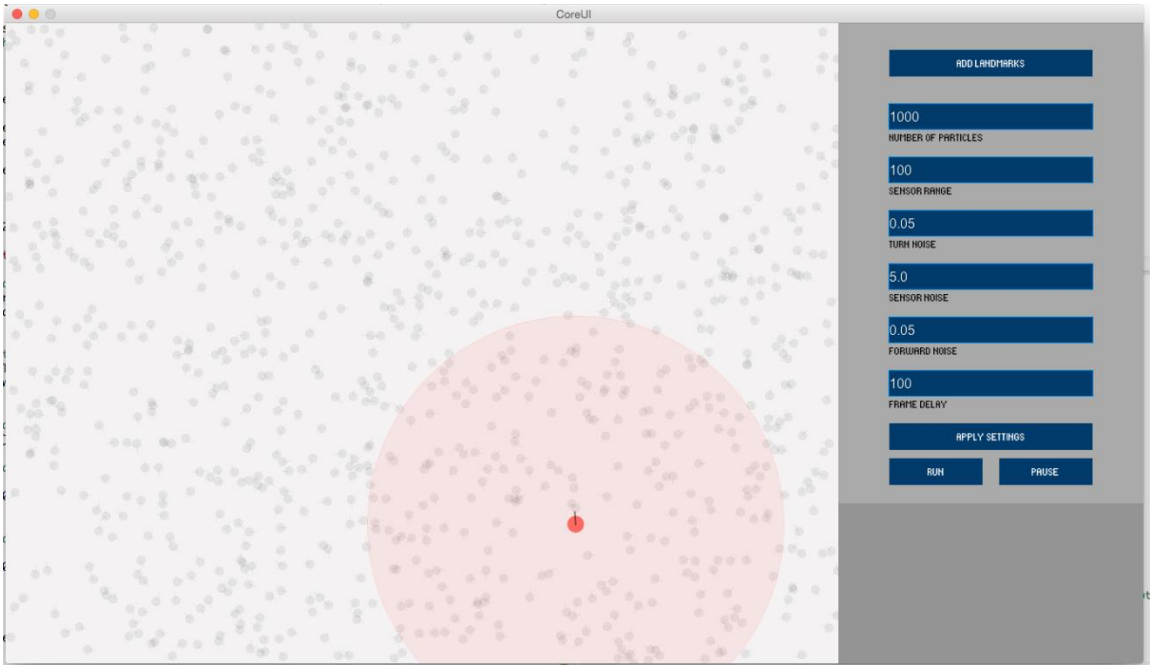


Figure 10.1 The user interface for the 2D localisation simulator

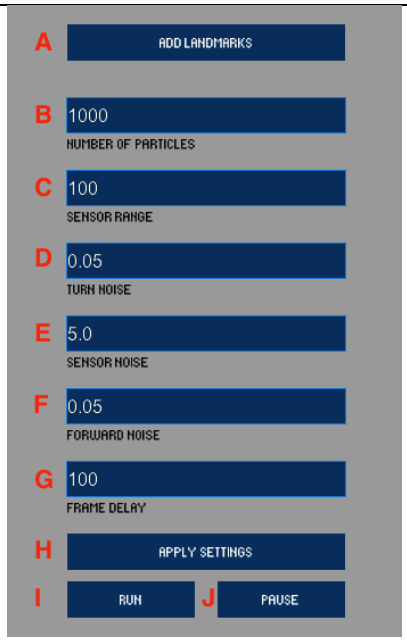


Figure 10.2 The simulation settings

Settings Explained

- A- Button to enable placing landmarks into the simulated world
- B- The number of particles to use in the simulation
- C- The range of the landmark sensor
- D- The amount of variance for the random Gaussian turning noise
- E- The amount of variance for the random Gaussian sensor noise
- F- The amount of variance for the random Gaussian forward noise
- G- The amount of milliseconds to wait before updating the simulation
- H- Use the given setting to the simulation
- I- Let the simulation play out
- J- Stop the simulation from operating

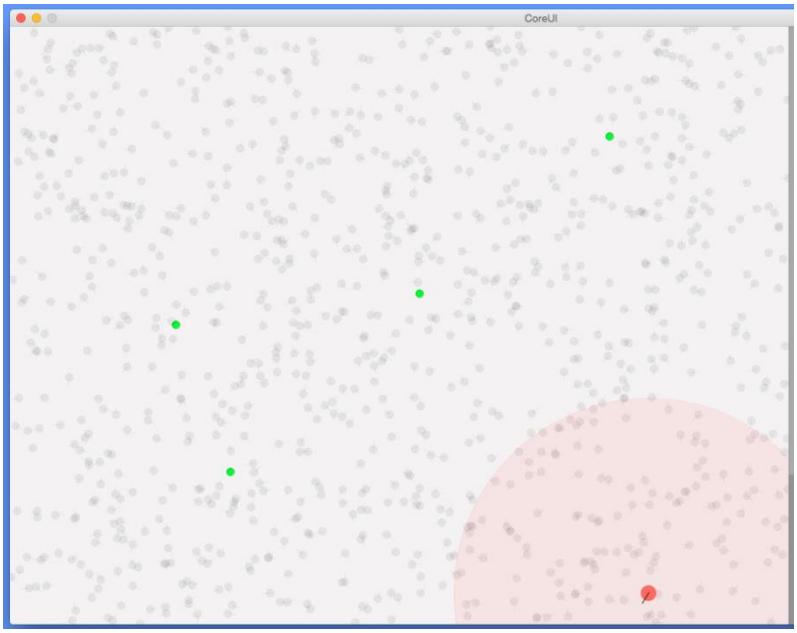


Figure 10.3 The robot is lost, particles are evenly distributed

Figure 10.3 shows the simulation at the beginning, there is no estimate for the robot's position as the particles are evenly distributed through the map.

Note that the landmarks are green, the robot red, the sensor detection radius of the robot surrounds it in light red and the particles are grey, the heading of a particle and the robot is indicated with a line coming from the front of it.

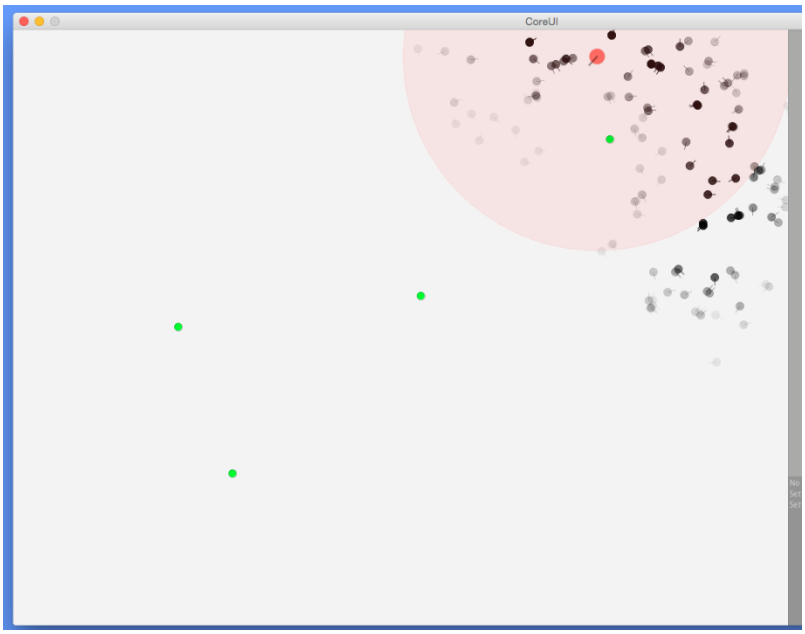


Figure 10.4 Measurements begin to eliminate very unlikely particles

Figure 10.4 shows progression of the simulation, very unlikely estimates are eliminated and there are small clusters of particles each with a different heading

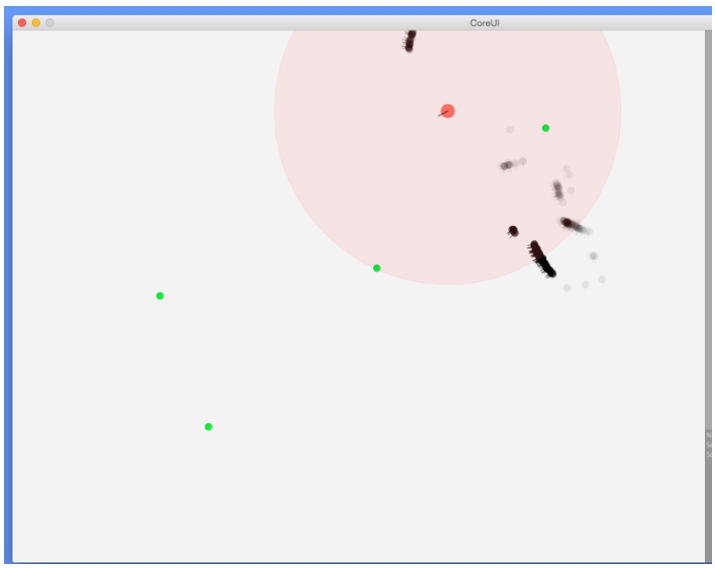


Figure 10.5 shows further progression of the simulation, larger separate clusters of particles are beginning to form, this time the heading of all the particles in each cluster is the same.

Figure 10.5 Clusters of particles are beginning to form

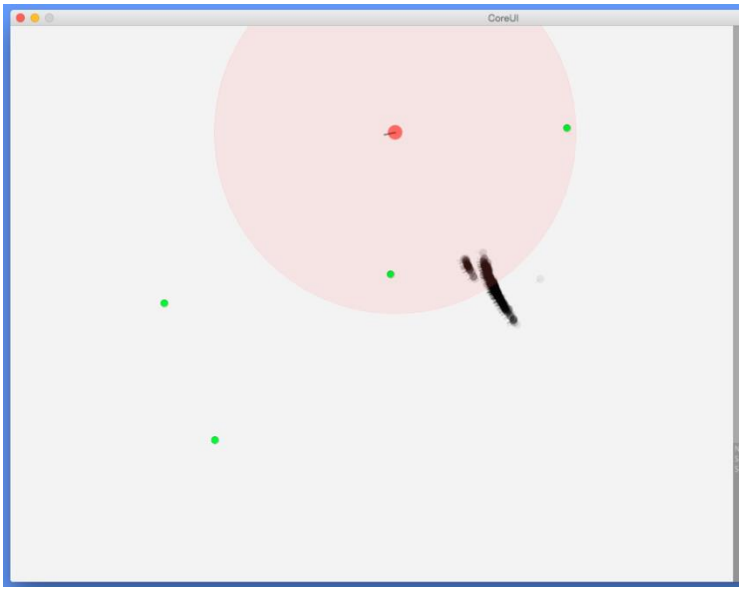


Figure 10.6 shows the formation of a single main cluster heading in the same direction. Note the cluster is heading in the wrong direction from the robot and a tiny cluster is beginning to split off from it.

Figure 10.6 Main particle cluster has formed going in same direction

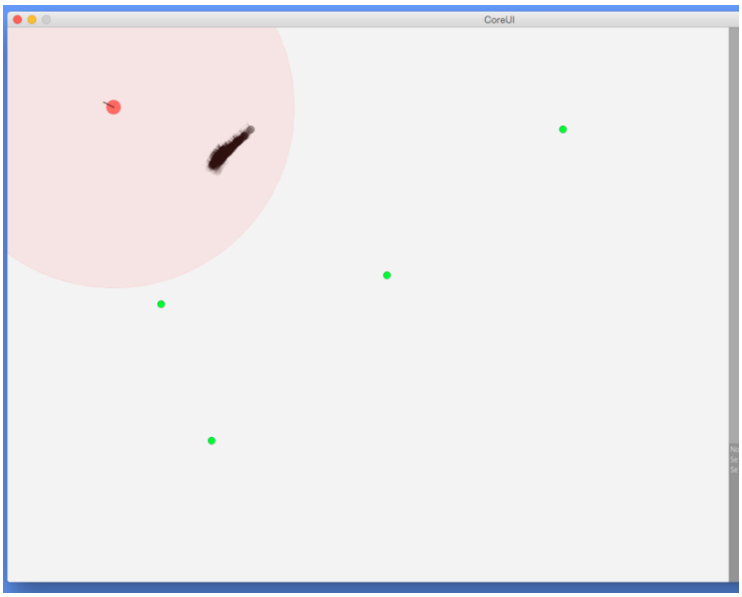


Figure 10.7 shows how the cluster that split off has followed the robot in the correct direction, but is still slightly off its position.

Figure 10.7 The particle cluster is beginning to correct its position

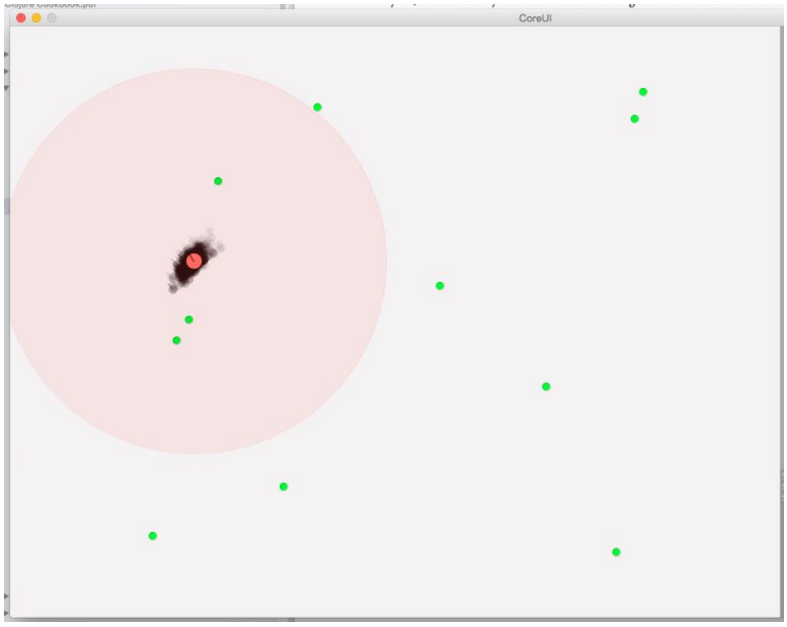


Figure 10.8 The particle cluster eventually arrives at the robots location

Figure 10.8 shows the simulation once the particles have an accurate estimate of where the position of the robot is, more landmarks have been added which increased the speed and accuracy of the of the localisation estimate.

11 Appendix 6 – Project Specification

PROJECT SPECIFICATION - SEGM 2015/16

Student: **Kevin Charles**
Date: **19th October 15**
Supervisor: **Alessandro Di Nuovo**
Degree Course: **BSc Computer Science**
Title of Project: **Simultaneous Localisation and Mapping, a 2D LIDAR based implementation**

Elaboration

Self-driving cars represent an interesting area of applied computer science, which has inspired me to investigate the use of similar technology for indoor navigation – in this case mapping out a finite indoor space, to perform this mapping some method of localisation will also be necessary. This problem is known as simultaneous localisation and mapping, this project will be a simple implementation of a solution to this problem, what will make this a simple implementation is the fact that the environment will be *finite, indoors, static and be modelled in only two dimensions.*

For this final year project, I will be developing a mobile robot capable of mapping its environment in two dimensions and knowing where it is in this generated map. The robot will be mechanically simple only using motors to move around in its environment and using a LIDAR (**LI**ght-**D**etection-**A**nd-**R**anging) sensor combined with a 360 degree rotating mirror to measure the distance of objects relative to the position of the robot, using this data to create a map of its environment.

In this project there will be two software systems, the software operating the hardware on the mobile robot and a control system running on another computer remotely, the two systems will communicate with each other via a simple Bluetooth connection. The robot will send data about its environment to the control software running on the computer, which will put this data together to create the model and be used to send instructions to the robot about where to move to explore next. Essentially the robot acts as the "eyes" and the remote control software acts as the "brain".

The software running remotely on a computer will have a UI that displays the model of the environment in a 2D top down floor plan so that a human operator can see what the system "sees", this software will also be used to control the robot remotely. The human operator will use the generated map created to view the robot's environment and move it around in this environment to create a full map

This project should:

- Investigate the suitability of LIDAR as an obstacle sensing technology for an indoor navigation system
- Investigate methods of environment representation in computer memory
- Investigate algorithms that can be used to estimate an agent's location in its generated map

Project Aims

- Design and build robotic platform that has the following facilities
  - can move on 2 axes (Forward/ Backward , Left / Right)
  - can observe/sense 360° of its environment
  - can be controlled wirelessly via Bluetooth communication
- Develop the software the will operate the hardware and run on the robot
- Design and implement a well-defined interface that will allow two-way communication between the robotic platform and the control software

- Develop an application that can remotely connect to the hardware and send commands to it
- Develop the application so that it can visually display the robot's immediate environment model in two dimensions (*possibly in real time*)
- Implement at least 1 algorithm that can compute the robot's location and map its environment

Project deliverable(s)

-A mobile robot that has the facilities to sense its environment and be remotely controlled
-An application that can remotely control the robot and that implements a SLAM algorithm to visually map the environment and estimate the robot's location in it. (with the environment being *finite, indoors, static and be modelled in only two dimensions)*

Action plan

Task	**Done By Date**
1. Research into hardware & gathering of parts	23rd October 2015
2. Building and testing of the mobile robot	14th November 2015
3. Design and implement the robot's communication and control interface	14th November 2015
4. Implement skeleton version of the robots control software	26th November 2015
5. Produce a complete design of how the control SW will work	23rd December 2015
6. Investigation & Research into environment representation models and SLAM algorithm to implement	23rd December 2015
7. Decide on an environment representation method and SLAM algorithm to implement	23rd December 2015
8. Implementation of the control SW and the SLAM algorithm that will be used to create the map visualisation.	28th February 2016
9. Write up of the project report	31st March 2016

Task descriptions

Research into hardware & gathering of parts:
At this stage I will be focussed on collecting information about the parts I will need and learning about their specifications and how to go about operating them, the circuitry that will be needed to control them and the software that will run on the Arduino to control these devices – in particular I will be looking into any libraries that I can use to control each part. I will also be considering what the chassis design will be like.

Building and testing of the mobile robot:
At this stage I will go about actual construction of the robot I will be using the information about each part I have collected to put the various parts together into one unit. I will either be using the basic software I wrote to operate the robot's parts as a starting point or the device libraries I found to go about writing the software that will operate the robot's hardware. Testing of all the robot's features will be performed to ensure the platform works properly.

Design and implement the robot's communication and control interface:
At this point I will be looking at designing the way in which the robot will communicate with the remote software. The connection will be via Bluetooth but at this point I will be considering how is the data sent from the mobile platform (e.g. every second, when a new data point is created), how this data will be packaged up and sent, the speed of communication and what controls I will be able to send from the control software to the robot.

Implement basic version of the robots control software:
At this point I will be implementing a skeleton version of the control software with minimal functionality the control software will only be able to receive data from the robot (but not yet do anything with the data just to test that it can receive the data) and send instructions about how to move – essentially the robot will just be able to be remotely driven at this stage.

Produce a complete design of how the control software will work:
Here my aim will be to have fleshed out a pretty much complete design of how the control software will function using the understanding I have developed from my research into environment modelling and SLAM algorithms. These designs will likely contain class diagrams that reflect all of the components that will be in the program and their relationships to one another, with sequence diagrams explaining how a user will interact with the control software and how it interacts with the robot. There will also be documents that go into detail about critical parts of the control software: how the environment model is stored / operated on, the way the SLAM algorithm is implemented and how a visualisation will be created on screen.

At this point the intention is to try to have an understanding of more than one SLAM algorithm and environment modelling method and compare how they could be implemented and use this comparison to choose one to implement in the actual control software. The design will be open to modification should I find any flaws through either more research or during implementation but I will try to make sure I thoroughly evaluate my designs to catch any such flaws early on.

Investigation & Research into environment representation models and SLAM algorithm to implement:
This will be on going, I am currently watching lectures about the topic of SLAM to develop my understanding of the area, after the construction of the robot is complete I will focus solely on research and design of the control software. My aim is to preferably have an understanding of at least two SLAM algorithms and environment modelling methods so I can perform a comparison between these and make a decision based on which one I believe to be the most suitable.

Decide on an environment representation method and SLAM algorithm to implement:
I will use my research that I have collected on these, preferably performing a comparison between environment representation method and SLAM algorithms, I will make a decision on which to use based on this information.

Implementation of the control SW and the SLAM algorithm that will be used to create the map visualisation:
This is the largest stage where I implement the control software fully using my research and software designs, I've given myself at least 2 months to go about doing this implementation – I don't believe it will take this long should I follow my designs as it should be simply implementing the design but I would like to leave slack time should I run into any problems. I'd want to have it completed by the end of February.

Write up of the project report:
With this being another major part of the project I have given myself the target of a month to

complete the report, as I am working on the project I intend to create logs for each day of what I've been working on and the decisions that I have made through out development – I will be able to use these logs and the other documents I have created throughout project development to create my final report. I'll be using speech to text software so I can create these logs quickly while I am working so that they aren't an intrusive clumsy disruption to working on the project.

Project Risks

One risk to delaying this project is the interface between control software and mobile platform, because I am building the mobile platform and its software first followed by the control application whilst researching into the problem of SLAM. This means that my knowledge about the subject will be more substantial by the time I am building the control application and there could potentially be misalignment between the original interface I designed for controlling & receiving information from the hardware and the interface I might deem more suitable with hindsight.

To minimise this risk, I plan to implement a skeleton version of the control software a week after the robot's construction. This skeleton control software will simply only send basic motion commands to the robot and receive data which it will print to the console, the reasoning behind this is to get some empirical testing done with this basic interface, see what is effective and make any large modifications there and then. This should prevent the need to perform any large modifications when implementing the rest of the control software.

Another potential risk to the project schedule could be the Java solution that is used to perform serial communication between the Java-based control application, currently I plan to use an open source solution called "Ardulink" which provides a solution for "lower level" serial communication functionality in Java which I would then build the rest of the application on top of. Having not worked with it yet I cannot say it is 100 percent suitable so I have a contingency plan, a Java library called "jSerialComm" which would require more work done to get serial communication which would mean less time to implement high level functionality but it is another option. To put things into perspective the Ardulink solution is essentially a small framework and the JSerialComm solution is more like a library.

12 Appendix 7 – Ethics Form

Ethics Checklist – SEGM Final Year Project 55-6727-00L
If the answer to any question is 'yes' the issue **MUST** be discussed with your project supervisor.

Question		***Yes/No***
1.	Does the project involve human participants? This includes surveys, questionnaires, observing behaviour, testing etc.	No
2.	Does the project involve the use of live animals?	No
3.	Does the project involve an external organisation? (please write the name of the organisation in the box)	No
4.	Does the project require access to any private or otherwise sensitive material?	No
5.	Does the project require the reproduction (beyond normal academic quotations) of materials authored by a source other than yourself?	No

Adherence to SHU policy & procedures:

Declaration
I can confirm that: • I have read the Sheffield Hallam University Research Ethics Policy (available at http://www.shu.ac.uk/_assets/pdf/research-ethics-policy.pdf) • I agree to abide by its principles.  Signature …… Print Name…Kevin Charles……. 23 Oct. 15 Date ……..