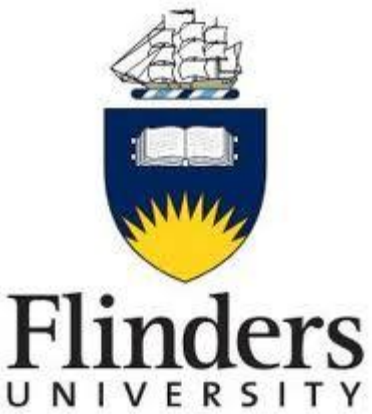
ROBOT LOCALISATION AND MAPPING

Localisation & Mapping on an Autonomous Robot

Flinders University

James Mead 2012

Localisation & Mapping on an Autonomous Robot



Author: James Mead

Student ID: 2055504 **FAN: mead0042**

Degree: Bachelor of Engineering (Computer Systems)

Team: James Mead, Scott Penley, Ken Birbeck

Supervisor: Dr Nasser Asgari

Submission Date: November 2012

Submitted to the School of Computer Science, Engineering, and Mathematics in the Faculty of Science and Engineering in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Computer Systems) at Flinders University – Adelaide Australia.

I certify that this work does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

16/11/2012

Acknowledgments

A big thank you must firstly go out to the team, Scott Penley & Ken Birbeck. Without them this project would have been vastly different.

Secondly to our supervisor Dr Nasser Asgari, your insight and guidance was much appreciated and helpful.

Third and finally to Flinders University, and the school of Computer Science, Engineering, and Mathematics for providing the technology and infrastructure that made this project possible.

Abstract

This report is not a research paper, but rather an investigation and practical application report of the work done in the fields of mobile robots, mapping and localisation. It specifically looks at the mapping and localisation methods used in this project. While localisation and mapping can be achieved in a number of different ways, it is important to choose a method suited to the problem at hand. This can be done by asking “what do I want this robot to achieve?” and then comparing this to both the environment it will be working in and the sensors available to the robot. This report will look into the investigations made of certain sensors and hardware, and then the application of this research into a practical robot model in a real world scenario.

Contents

1. Introduction

The goal of this project was to create an autonomous robot capable of mapping an indoor environment. With the goal not being specific, the solution could be achieved through hundreds of different approaches. With the approach being so open, it was left to the team to decide which approach would be the most suited to them and the work environment.

The problem of SLAM (simultaneous mapping and localisation) is one that has been looked at in depth in the robotics community, and there has been a lot of work done in solving it. The problem is that the SLAM problem is a very broad one, and can apply to many different situations. A robot can be localised within a known environment from a known position, it can be localised in a known environment from an unknown position, or it may need to be localised in an unknown environment from an unknown position. The final case is the most difficult one to solve, and is also the one that applies to this project. For this reason the different methods of localisation need to be looked at and the appropriate one needs to be implemented into a working scenario. From this point the project is to be taken to the next step to implement autonomous control of the mobile robot within the environment, with the final goal of having a robot capable of driving around autonomously whilst simultaneously creating an image of the environment it is seeing. The idea is that this data may then be used by other robots with different sensors to navigate the previously mapped environment.

2. Proposed Project Plan

At the beginning of the project we looked at how the project could be broken down into smaller individual tasks, and the skills each member had for tackling these goals. We looked at the robot we had to work with, the sensors it had on it, and took into account any future modifications that may have been possible or extra hardware that would be added to the robot.

The work was done by a team of three; James Mead, Ken Birbeck and Scott Penley. Because of this the project was divided into three main sections, the first section was sensor interfacing, data retrieval, and programming of the Microsoft Kinect. This was assigned to Scott as he had previous experience in programming the Kinect and had a good base knowledge of the sensors.

The second section was navigation and path planning. This section was assigned to Ken as he had experience with programming the PIC board and driving the robot. His role was to take the map produced by the mapping and localisation section and use it to navigate around the environment as well as avoiding obstacles.

The final area was mapping and localisation. This would involve using the sensor data to produce the map and position the robot within the environment. This was assigned to me, and the bulk of this thesis will be on the work undertaken to achieve this goal.

3. Hardware

3.1 Lynx Robot

The robot base provided for this project is the Lynx robot. It is a four wheeled robot, featuring a range of sensors.

Figure 1: Initial proposition of Lynx robot (minus the LIDAR)

The robot contains a FitPC compact PC for controlling the operation of the robot. It has wireless connectivity so it can be remotely controlled via remote desktop over the CSEM network. The driving is controlled by a dsPIC board, which is programmed through MPLAB and the on board ICD 2 programmer. It also has a Microsoft Kinect, display screen, 2 VGA cameras, a Sparkfun IMU and infrared range sensors. The robot is also planned to be modified to have a LIDAR attached and an Apple iPhone.

3.2 FitPC 2

The FitPC on board the robot contains an Intel Atom 1.6GHz CPU and 1GB DDR2 RAM. It has 6 USB 2.0 ports and 802.11b/g wireless connectivity, giving it enough functionality and power to run everything needed on the robot.

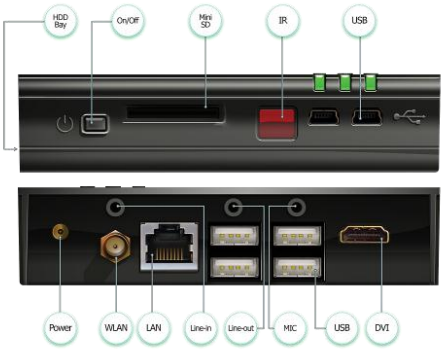


Figure 2: FitPC 2 Features and Specifications

The operating system originally loaded on the FitPC was windows XP, but this was upgraded to Windows 7 to allow Kinect functionality on the FitPC.

3.3 Sensors

The Lynx robot came with an array of sensors attached to it. While not all of them were used in the project, investigation into each one carried out to determine how each could be used. Then the final decision was made about which sensors were best suited to the project.

3.3.1 RB-Pol-125 Pololu 12V, 50:1 Gear Motor w/Encoder

Integrated quadrature encoder: two-channel Hall Effect encoder provides 64 counts per revolution of a magnetic disk attached to the motor shaft. The Gearbox output counts per revolution are determined by multiplying this by the gearbox ratio, resulting in 3200 counts per revolution. When coupled with the wheels this results in 4.22 encoder counts/mm.

Figure 3: Wheel Encoder motor

3.3.2 Sparkfun Razor 9DOF IMU

The 9DOF Razor IMU incorporates four sensors - an LY530AL (single-axis gyro), LPR530AL (dual-axis gyro), ADXL345 (triple-axis accelerometer), and HMC5843 (triple-axis magnetometer) - to give you nine degrees of inertial measurement. The outputs of all sensors are processed by an on-board ATmega328 and output over a serial interface. (Sparkfun Electronics)

Figure 4: Sparkfun Razor IMU

3.3.3 Microsoft Kinect

- Microsoft Kinect drivers (from SDK)
- Depth sensor
- Skeletal tracking
- Xbox360 version: accurate within range 0.8m to 4.0m

Figure 5: Microsoft Kinect

The Kinect was a sensor that was used in early development, but later removed from the robot.

For more information on the work done with the Microsoft Kinect, refer to Scott Penley's thesis.

3.3.4 Sharp GP2Y0A02YK0F Distance Measuring Sensor (Infrared distance sensor)

Measuring distance: 20 to 150 cm

Analog output type: Distance detection uses triangulation and is not easily affected by reflectivity of the object, environmental temperature or operating duration.

These sensors were to be used as an emergency stop mechanism for the robot. If an object came within a threshold distance seen by the infrared sensors the robot would stop, as to avoid unwanted collisions. They were never planned to be used for mapping since their limited range, accuracy and viewing angle would inhibit their usefulness for this purpose.

The 6 sensors were spaced evenly around the robot to maximise their field of view.

Figure 6: Infrared distance sensor

3.3.5 Hokuyo utm-30lx laser rangefinder

The laser rangefinder (or LIDAR for short) is the main sensor on the robot. It is to be used for range detection around the robot in a 2D plane. At the onset of the project it was important to research all possible rangefinders suitable for the project, as one was yet to be purchased.

Laser rangefinders can vary from around $1000 up to about $400 000, depending on the quality of the LIDAR and its requirements. As for commercial grade, robotics application based LIDAR's the choice was of two manufacturers, SICK and Hokuyo. SICK manufacturer LIDAR's for this sort of application ranging from about $5000 up to $10 000, with the devices offering high end specs beyond the needs of this project. Hokuyo on the other hand, target their products at learning facilities and hobbyist robotics consumers. This made their product a better choice when looking at a suitable LIDAR.

The list of possible LIDAR's was narrowed down to three.

Hokuyo URG-04LX-UG01 (entry level)

- 10Hz scan frequency
- 240° sweep angle,
- 4 meters maximum range
- Approximately $1100.

Hokuyo UGH-08LX (mid-level)

- 67ms scan
- 270° field of view
- 8 meter maximum range
- 0.36 angular resolution
- Approx. $4000

Hokuyo UTM-30LX (High level)

- 40Hz scan frequency
- 270° sweep angle
- 30 meters effective range
- High speed USB 2.0 serial connection
- 0.25° Angular resolution (1081 scans per sweep)
- Semiconductor laser diode(λ=905nm)
- Approx. $5500.

The UTM-30LX was accepted for the project and purchased. Unfortunately, there was a long delay between submitting a request for the LIDAR and it arriving, leaving a reduced timeframe to implement the LIDAR on the robot platform. The full technical document can be seen in Appendix D.

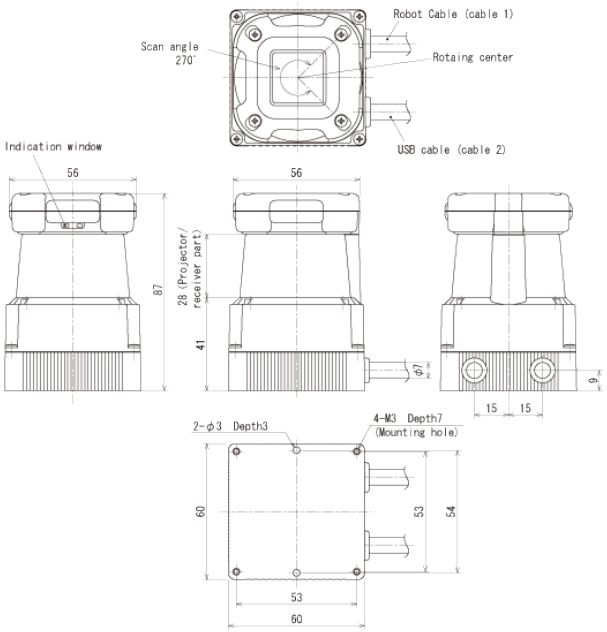


Figure 7: Hokuyo UTM-30LX Dimensions

3.3.6 Apple iPhone

The iPhone was a sensor added later in the development. It was not part of the initial plans for the robot, but due to Scott's previous experience with iPhone programming he was able to see a use for it on the robot. For that reason, it was initially added as a secondary option to the Sparkfun IMU, but later, other helpful features were discovered. An application which ran on the phone could be accessed on the mobile robot, and contained many functionality options.

These options included an emergency stop button and a connect button that allowed the robot to connect to the base PC over the network without being connected to a screen. A second iPhone was later implemented during testing that could drive the robot. This allowed a user to wirelessly drive the robot by using an application on the phone which used screen tilt for movement and acceleration.

For the full details on its functionality and implementation refer to Scott Penley's thesis.

4. Software

When looking at the project overview, nearly all the work to be done is in code. This means it is important to choose an appropriate software platform to build upon.

When comparing operating systems, it appears that a lot of work done on mobile robots is on a Linux operating system. This is mainly due to the flexibility and control the operating system gives, along with the performance increases due to the fact that a Linux operating system is more efficient and lighter on memory usage.

Unfortunately no one in the team had enough experience with the Linux operating system to be comfortable with the work needed to be undertaken with it. The other problem faced is that the robot may use an Xbox Kinect, which being a Microsoft product means the SDK released for it is Windows 7 exclusive. For these reasons the platform chosen for the robot and base PC is a Windows 7 operating system and Visual Studio 2010 IDE.

4.1 Packages/libraries

When looking at the code that needed to be written, it would be almost impossible to achieve the goal of the project without the use of previously developed code libraries. If libraries weren't used, the work produced would be years behind current work being produced. For this reason, available libraries and software packages were explored.

4.1.1 ROS.org

*ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license.****Invalid source specified.***

The ROS.org website offers a large number of open source libraries, amongst which are some very cutting edge ones. The libraries require the installation of the software packages on an Ubuntu system, with only limited support for a Windows machine. For this reason any ROS packages went unused.

4.1.2 MRPT

> ***"Mobile Robot Programming Toolkit (MRPT)*** *provides C++ developers an extensive, portable and well-tested set of libraries and applications which cover the most common data structures and algorithms employed in a number of mobile robotics research areas:* ***localisation****,* ***Simultaneous Localisation and Mapping*** *(SLAM),* ***computer vision*** *and* ***motion planning*** *(obstacle avoidance).*

> *Key points in the design of MRPT are **efficiency** and **reusability of code**. The libraries include classes for easily managing 3D(6D) geometry, probability density functions (pdfs) over many predefined variables (points and poses, landmarks, maps), Bayesian inference (Kalman filters, particle filters), computer vision, SLAM, path planning and obstacle avoidance, 3D visualization of all kind of maps (points, occupancy grids, landmarks, graphs, ...), etc. Gathering, manipulating and inspecting **very large robotic datasets** (Rawlogs) efficiently is another goal of MRPT, supported by several classes and applications".*
>
> (MRPT, 2012)

The MRPT software package formed the backbone of the mapping and localisation section of the project. The software packages are structured in a way that higher level modules are dependent on lower level ones, as they may use mathematical equations from these lower level modules. This means the user can use only the modules they need rather than installing the entire library in their project, which would vastly increase program size and compile time during development.

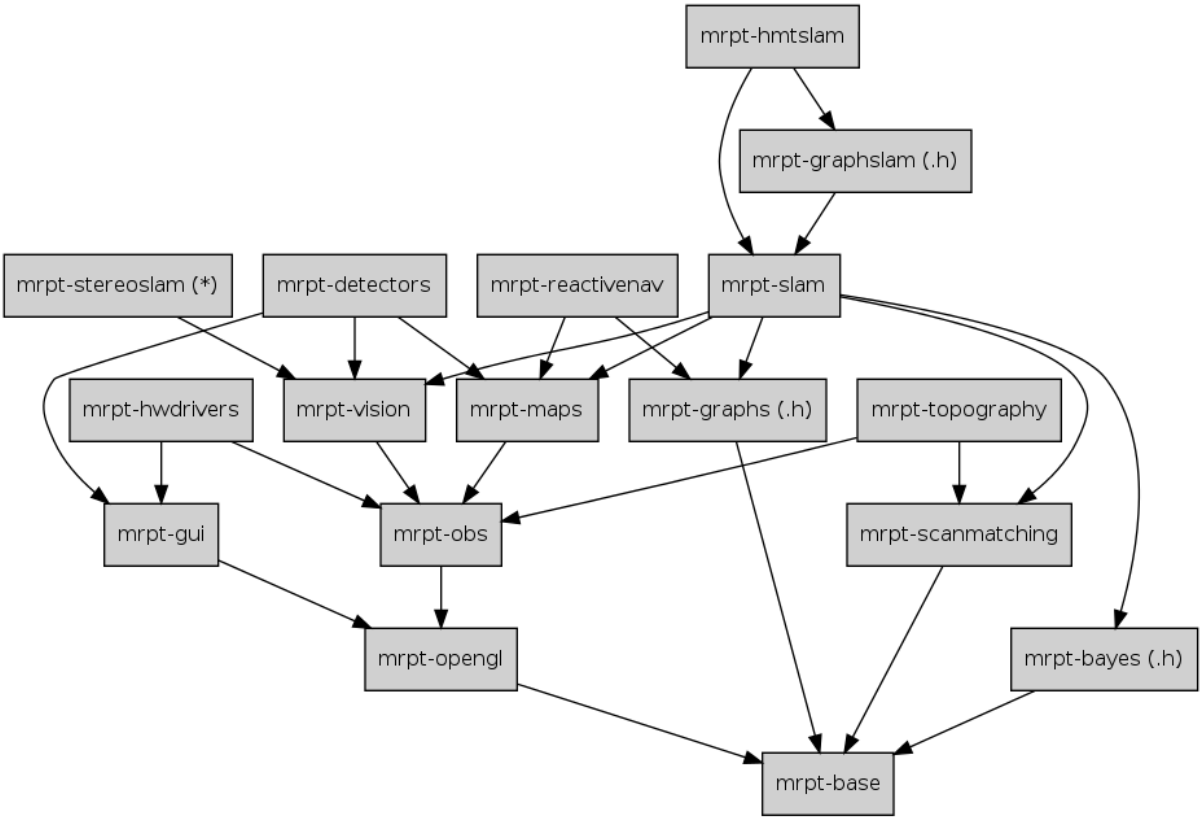


Figure 8: MRPT Library Structure

Each library contains a list of functions and these can be seen in the MRPT API. The libraries used within this project were mrpt-base, mrpt-bayes, mrpt-graph, mrpt-gui, mrpt-maps, mrpt-obs, mrpt-opengl, mrpt-scanmatching, and mrpt-slam. These were required because at least one or more functions from each library were used in the code.

Installing the libraries was quite a large task on its own. Once the library package was downloaded, third party software was required, as a few of the MRPT libraries made use of them.

4.1.3 Third party Software

4.1.3.1 OpenCV

> ***"OpenCV** (**Open** Source **C**omputer **V**ision) is a library of programming functions for real time computer vision. OpenCV is released under a BSD license, it is free for both academic and commercial use. It has C++, C, Python and soon Java interfaces running on Windows, Linux, Android and Mac. The library has >2500 optimized algorithms. It is used around the world, has >2.5M downloads and >40K people in the user group. Uses range from interactive art, to mine inspection, stitching maps on the web on through advanced robotics".*
>
> (OpenCV, 2012)

OpenCV is heavily integrated with ROS.org along with MRPT libraries, and it contains a large amount of optimised algorithms in the field of mathematics and robot operations.

4.1.3.2 wxWidgets

> *"wxWidgets is a C++ library that lets developers create applications for **Windows, OS X, Linux and UNIX** on 32-bit and 64-bit architectures as well as several mobile platforms including Windows Mobile, iPhone SDK and embedded GTK+. It has popular language bindings for Python, Perl, Ruby and many other languages. Unlike other cross-platform toolkits, wxWidgets gives its applications a truly native look and feel because it uses the platform's native API rather than emulating the GUI. It's also* extensive, free, open-source and mature".
>
> (wxWidgets, 2012)

wxWidgets is not necessary to run the MRPT libraries but it is strongly recommended, as it is used heavily for creating GUI's in a number of example projects. While it was not used in the final program it would be beneficial for **future work**.

4.1.3.3 CMake

> *"CMake is a family of tools designed to build, test and package software. CMake is used to control the software compilation process using simple platform and compiler independent*

> *configuration files. CMake generates native makefiles and workspaces that can be used in the compiler environment of your choice”*.
>
> (CMake, 2012)

As the MRPT libraries are cross platform, they are packaged as the raw .cpp files and a .makefile. From here CMake is used to compile the libraries for a specific platform, in our case this platform is Visual Studio 2010. Along with creating the .sln and all other required files the CMake program allows a number of other options to be changed when compiling, such as allowing support from other third party software or CUDA support for example.

5. Mapping

As the data is gathered from the sensors it needs to be stored in some form of map. The map is needed to both display the environment seen by the robot, while also being used to help the robot position itself in the environment. It can also be used for creating a path for the robot to traverse in path planning, and in this project that is the case. For these reasons it is important to choose an appropriate mapping method that is best suited to this project.

Three fundamental relationships must be understood when choosing a particular map representation:

1. The precision of the map must appropriately match the precision with which the robot needs to achieve its goals.
2. The precision of the map and the type of features represented must match the precision and data types returned by the robot's sensors.
3. The complexity of the map representation has direct impact on the computational complexity of reasoning about mapping, localisation, and navigation.

Traditionally there are two main types of mapping for mobile robots, they are as follows.

5.1 Topological Mapping

Topological mapping is a method of mapping where "places" in the world such as hallways and junctions are represented by nodes, while paths correspond to arcs. This means that in large open environments, only the important features are saved into the map. The benefit of this type of mapping is that large scale environments can be easily mapped without increasing computation time or memory usage. Path planning is also much simpler as the computation of arcs isn't greatly affected by the size of the map. Objects in the environment can be further simplified to be represented by simply polygons, allowing for even less memory consumption when generating the map.

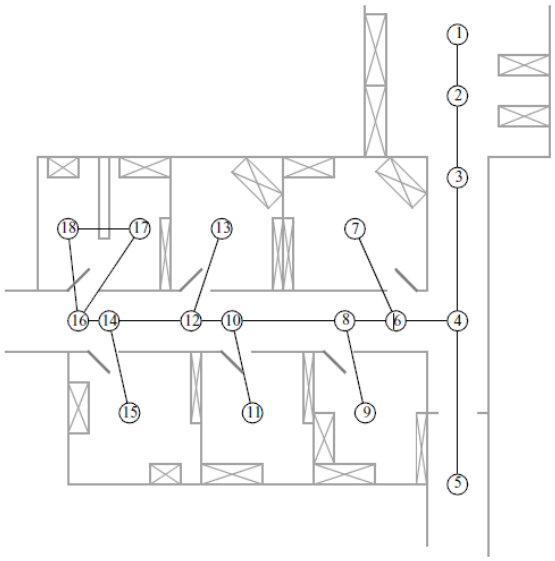


Figure 9: A topological representation of an indoor environment with nodes

In order for the robot to navigate a topological map accurately it must satisfy two constraints. First, it must be able to detect its current position in terms of the nodes of the topological map, and second it must have a means for traveling between nodes using robot motion.

The downside to this form of mapping is that it is not as accurate as other methods, and only partial metric information is stored in regards to the map. So while it would be beneficial to use this method when mapping an outdoor environment, it would be far less effective to use it for a smaller scale indoor environment where accuracy and more exact dimensions are important.

A key concept for this method of mapping is feature detection, and it is this method that keeps this type of mapping popular. In the case where the robot is traversing an already known map unique features can be detected and used to locate the robots exact position in the environment with a very high belief. Unfortunately this method cannot be used when traversing an unknown environment for the first time, as is the case in this project.

5.2 Metric mapping

Metric mapping is the most common method used for map representation in robot localisation. Objects within the map are placed with exact coordinates and the robots position within the map can always be given by x,y,θ. The most common representation of metric mapping is an occupancy grid. In this method the environment is represented by a grid, with each cell representing whether that area in the environment is occupied or not.

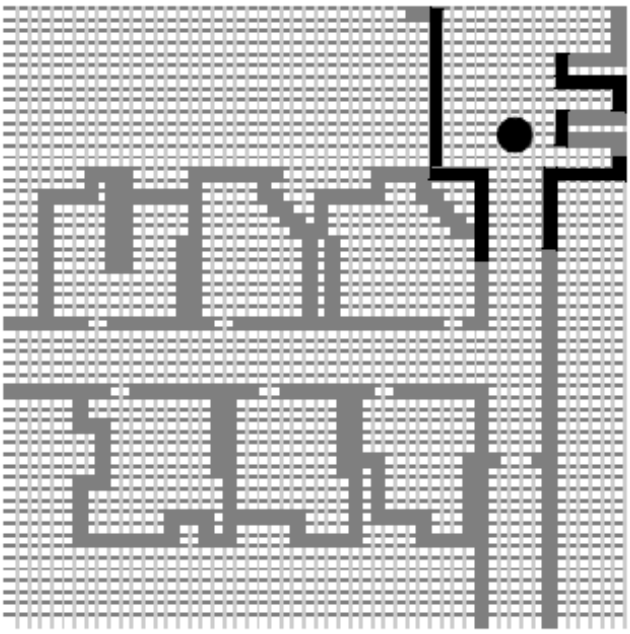


Figure 10: Example of an Occupancy Grid Map

The benefits of this representation are that the map is always a true metric representation of the environment. This method is generally more accurate than topological, and it is very suited to the use of laser range finders, as is the case with this project. This is because the cells are easily updated by taking the range data from the laser rangefinder, then using the robots current position update the corresponding cells in the map. The disadvantage of this mapping method is that is can have high memory consumption as the map grows larger, as every cell in the grid needs to be stored in memory, even if the cells represent empty space. Often path planning complexity grows in a non-linear way as the map grows, leading to problems.

5.3 Hybrid Mapping

While both metric mapping and topological mapping have advantages and disadvantages, a lot of work in recent years has been done to combine the benefits of both mapping methods into one. A hybrid mapping method is one that uses both. This method can be used to reduce the memory usage of the map via the use of oc-trees to combine cells of a grid map, creating a variable resolution partitioning of the map.

While this method does represent a more efficient method of mapping, it also adds another layer of complexity to the project, and for this reason it was not attempted. For more information refer to **Future Work**.

5.4 Deciding on a method

When looking at what we want the robot to achieve, along with the environment it will be operating in and the sensors and hardware available to us, occupancy grid mapping is the most suitable method.

As occupancy grid mapping is more suited to completely unknown environments and the use of laser range finder sensors, it evidently a more appropriate choice than topological mapping. As the data not being processed on the robot but instead on a base PC, memory usage and processing complexity are not going to be an issue.

6. Localisation

Localisation is the process of determining the exact position of the robot within the environment. There are a number of different methods that can be used solely, or in conjunction with other methods. Therefore it was up to the group to determine which method would be the best suited to this project. In theory, positioning within the map can be done without any form of localisation algorithm, by simply passing odometry data straight into the map. In practice, this results in an extremely poor map being drawn due to sensor errors or noise. To overcome this, the estimated sensor data is passed through an algorithm which cleans it up and results in a much more accurate value for the robots position in the map. The robots position within the map is also now given as a probability rather than an exact value.

When choosing a localisation method another factor that needs to be taken into account is the SLAM scenario. Often localisation needs to be done on a robot in a known environment. This could be the case of a tour guide robot, where it already has a map of its environment stored and all it needs to do is determine its exact position within that map. Another case could be a robot in a known position, with an unknown environment. The final case is a robot in an unknown environment, and in an unknown position. This means the robot needs to map the environment it can see, along with tracking its position within the environment, which is the case in our project.

6.1 Kalman Filter

The Kalman filter works by taking a series of measurements over time from the sensors, (odometry and IMU) and produces estimates of the robots pose that tend to be more accurate than using those measurements alone.

The algorithm works in a two-step process: in the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the next measurement is observed those estimates are updated. This is done using a weighted average, with a higher weighting being given to estimates with higher certainty. Because of the algorithm's recursive nature, it can run in real time using only the present input sensor data and the previously calculated position, no additional past information is required. A standard Kalman filter only works on a linear system, which in this case the robot is not a linear system. For this reason the Extended Kalman Filter was developed (EKF) and is what would be used in the project. It is basically the same as a standard Kalman Filter, but it takes in differential functions rather than linear ones.

The downside with the Kalman filter is that the sensory data being passed to it needs to be relatively accurate due to the linear nature of the algorithm. This means that if it is not then the algorithm can quickly breakdown and diverge from the correct results.

The MRPT library contains everything needed to implement an EKF through the mrpt-bayes library, and contains a large amount of data and information on implementing it. The problem is that the

odometry data being passed up from the robot locomotion is extremely inaccurate when turning due to slippage, which means there is a good chance that the Kalman Filter may break down. An attempt was made to implement an EKF into the program for localisation but due to time constraints and programming skills it wasn't a success, so its effectiveness in real world tests is unknown. It is definitely a good option for **future work**.

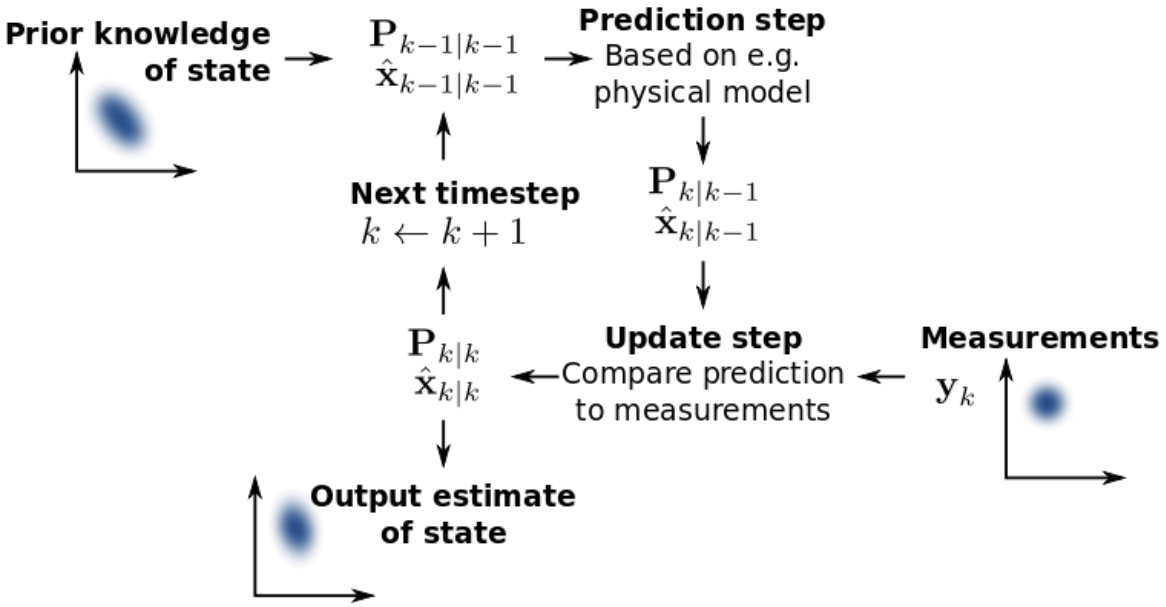


Figure 11: Kalman Filter Process

6.2 Particle Filter

A particle filter is a sophisticated model estimation technique based on simulation. It tracks the robots belief using an arbitrary probability density function to represent the robot's position. It does this by first tessellating the map into a finite number of possible robot positions, and then narrowing these possibilities down according to sensory data received in. This means it can localize a robot in a known environment with an unknown starting position. The benefit of particle filter localisation is that with enough samples, the robots true position in the environment will almost certainly be found. The problem with this method is that it is not effective if the map is unknown to begin with, and it is also extremely computationally expensive. While this method can be modified to work within an unknown environment, it is not an easy task. Particle filter localisation is best suited to the situation where sensor accuracy is reduced (using sonar instead of laser range finder for instance) and the robot is in an already known environment.

6.3 Scan Matching

Scan matching is a method by which a robots change in position can be found by examining the data received from the LIDAR. This can be done by taking two successive LIDAR scans and comparing the two to determine the transformation between the two. This can also be done using a single point cloud and an existing map.

One such method is known as ICP (iterative closest point). In this method the two point clouds are taken and then pass an initial estimation of their transformation between the two point clouds (calculated from the odometry and IMU values) and then the transformation is found by minimising the square errors between the corresponding entities. The "iterative" of ICP comes from the fact that the correspondences are reconsidered as the solution comes closer to the error local minimum (MRPT, 2012). The output of the equation is a PDF of the relative pose between the two maps.

The benefit of using this method is that it reduces the reliance on accurate odometry data, and can even be implemented to use no odometry data, but it is important to have an accurate sensor such as a high quality laser range finder to produce accurate point clouds.

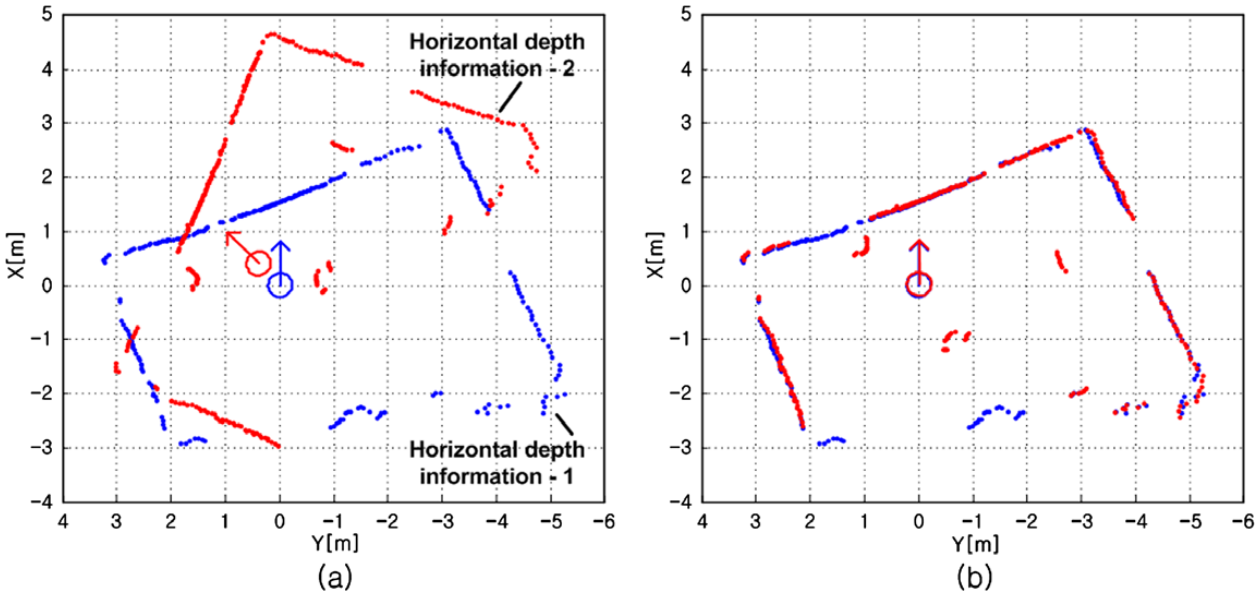


Figure 12: Example of Scan Matching method

The disadvantage of this method is that it can lose its accuracy when matching linear scans like a hallway or round room, and if this happens it has no real way of recovering it position. For this

reason it is normally used in conjunction with another method such as a Kalman filter to further increase accuracy and robustness.

When assessing the project and the equipment available, it was found that scan matching was the best option for localisation. This was decided due to the main factor of the robots design. The odometry of the robots wheels is too inaccurate for use with a Kalman filter, but the LIDAR on the robot is very accurate. For this reason it makes sense to use scan matching, while providing room for improvements and implementation of a different method in **future work**.

7. Robot Kinematics

As previously mentioned the robot base is the Lynx robot. The base comprises of 4 rigid wheels with the following design:

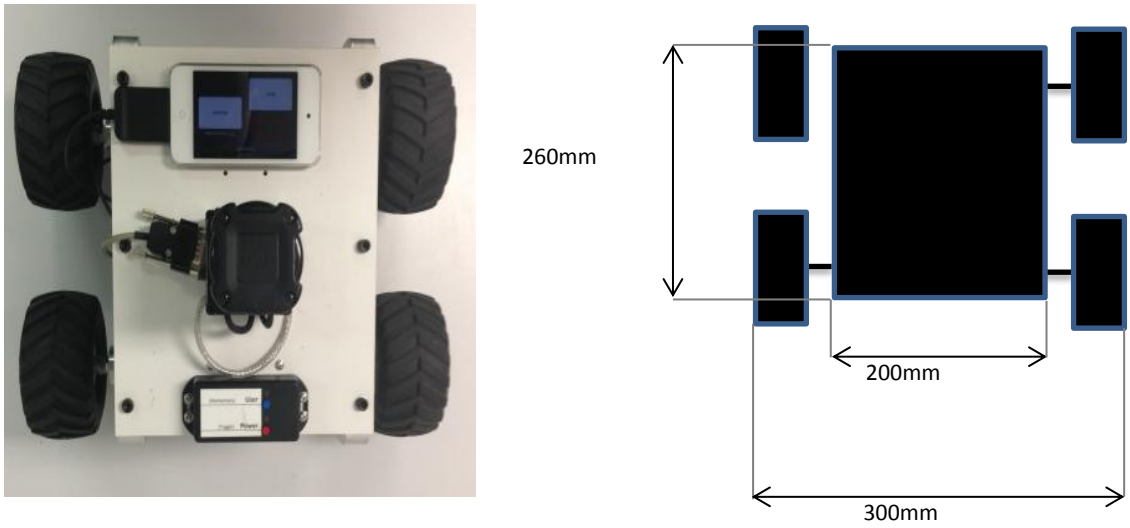


Figure 13: Lynx Robot & Dimensions

The robot was designed without this project in mind, and because of that it isn't purposely built to suit mapping and localisation. The robot turns by driving the left and right side wheels in opposite directions, but this leads to severe wheel slippage.

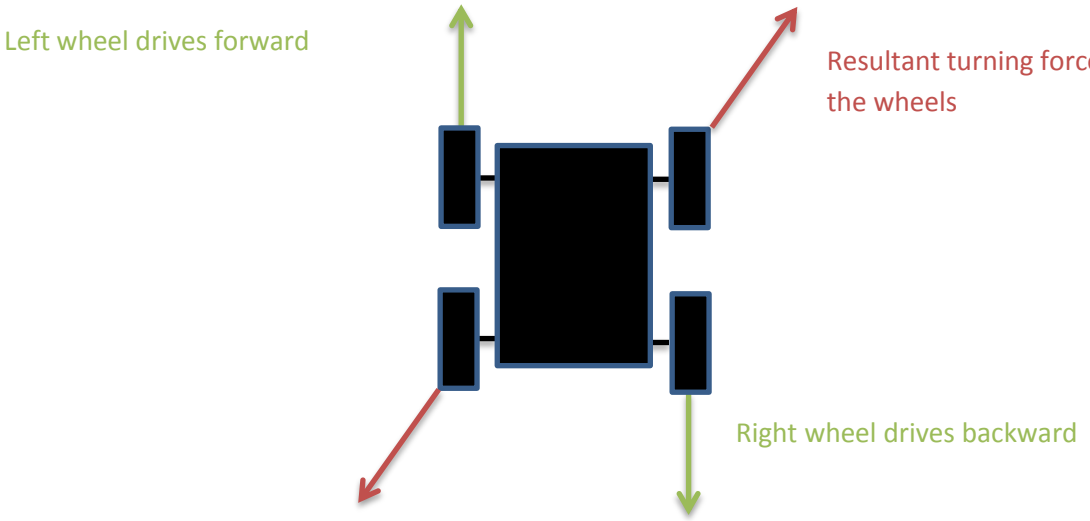


Figure 14: Illustration of wheel slippage on the robot

When performing point turns it was found that there was no exact pivot point on the robot. If there was then the LIDAR could be mounted on this point which would reduce calculation complexities and measurement errors.

Even the wheels themselves have some aspect of random slippage due to their wide dimensions.

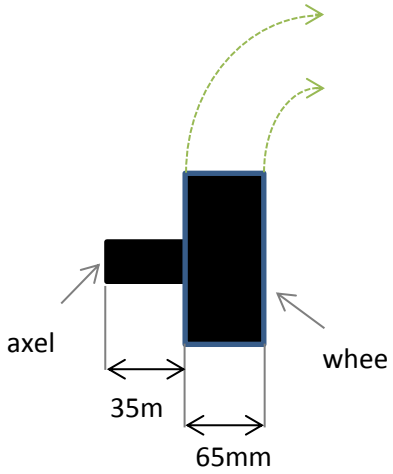


Figure 15: Illustration of wheel slippage on individual wheels

For the reasons illustrated above, the odometry values returned from the robot were less than desirable, making the localisation difficult from the start as a key component (odometry data) was rendered less reliable. Even testing in a straight line found odometry errors. Over the course of 4 metres the robot was continually driving about 3.89m, demonstrating an error of about 2.75%. This could possibly be attributed to the sponginess of the wheels, as their dimensions would skew depending on the robots weight. For this project the most suitable wheel type would be a thin rigid wheel, so that it would offer very little slippage and retain odometry accuracy.

A possible solution was to modify the design of the robot. A concept was put forward to remove the back two wheels and replace them with a single free spinning omnidirectional wheel at the rear.

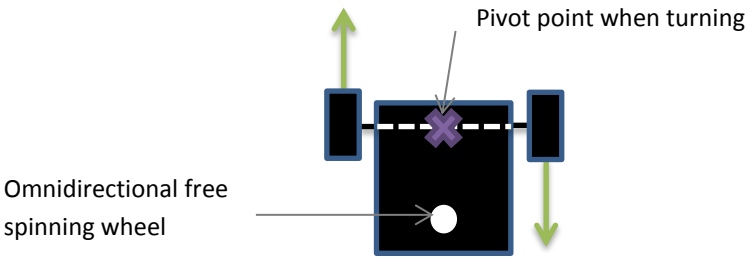


Figure 16: Proposed robot design to reduce wheel slippage

By modifying the robot to the design in Figure 16 then the odometry from the wheels when turning could also be used to calculate the θ angle of the robot when turning, which in turn could be combined with the IMU data and scan matching results to get a more accurate value.

The new pivot point between the front tyres could also be used as a point to mount the LIDAR, reducing calculation complexity and error margin when turning.

Unfortunately the workshop were not willing to modify the robot to this extent, which meant the initial design of the robot had to remain. The design does leave room for improvement in **future work**.

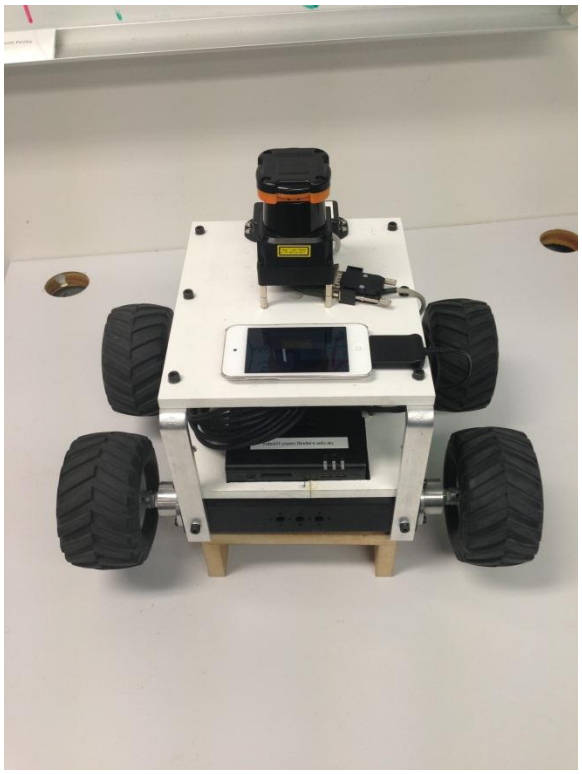


Figure 17: Final robot design (front view)

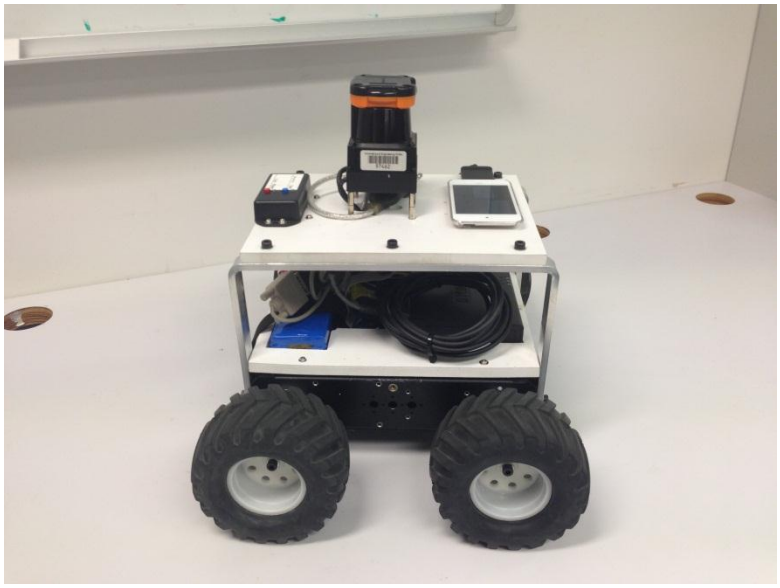


Figure 18: Final robot design (side view)

8. Program Structure

At the outset of the project there was no program structure to begin with, due to the fact that this was a brand new project. This means that an initial program plan needed to be drawn up, but as development continued this was changed as circumstances changed.

One common method and a method heavily integrated into the MRPT libraries is to drive the robot around, record all the data seen by the sensors on the robot and then process the data later offline. Unfortunately this method wasn't an option in this project as the robot needed to process all data online in real-time so that it could react to the environment it was seeing.

So the initial plan drawn up was for the robot to gather all the data and process it on the one machine (FitPC 2). There was also the possibility of adding a second FitPC to the robot to run just the Kinect since this sensor was resource hungry. However, during initial testing we could see the FitPC was struggling with even simple mapping, let alone any algorithms for localisation. This meant the plan had to be changed. One suggestion was to use the FitPC on the robot to gather all the sensor data, package it together and then transmit it wirelessly to a base PC where all the processing was done, and then from there the drive commands would be transmitted back to the robot. This would solve the problem of doing heavy processing on the FitPC, and at the same time solve another issue. The program that runs on the FitPC is written in C# code, as this was Scott's choice of language and the work done on the Kinect was in C#. The mapping program that uses the mrpt libraries is written in C++ and to have the two programs running on one machine and communicating would mean an addition wrapper program would be needed to allow the two programs to communicate, adding another layer of complexity. By sending the data wirelessly from the robot to a base PC this problem was removed.

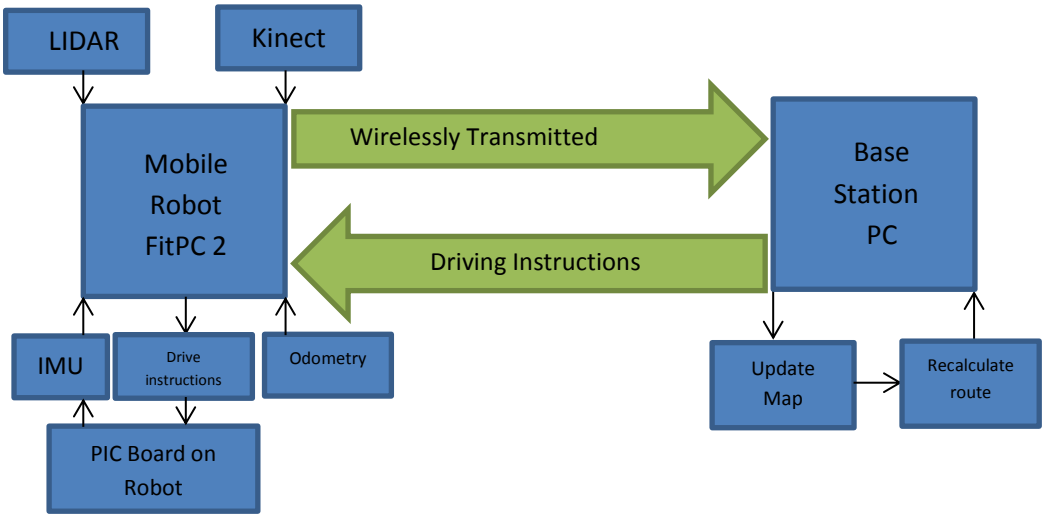


Figure 19: Simple block diagram showing overall structure of the program

Figure 19 shows the outline of how the program works overall. A more detailed and in depth diagram of the above block diagram can be seen in Figure 20.

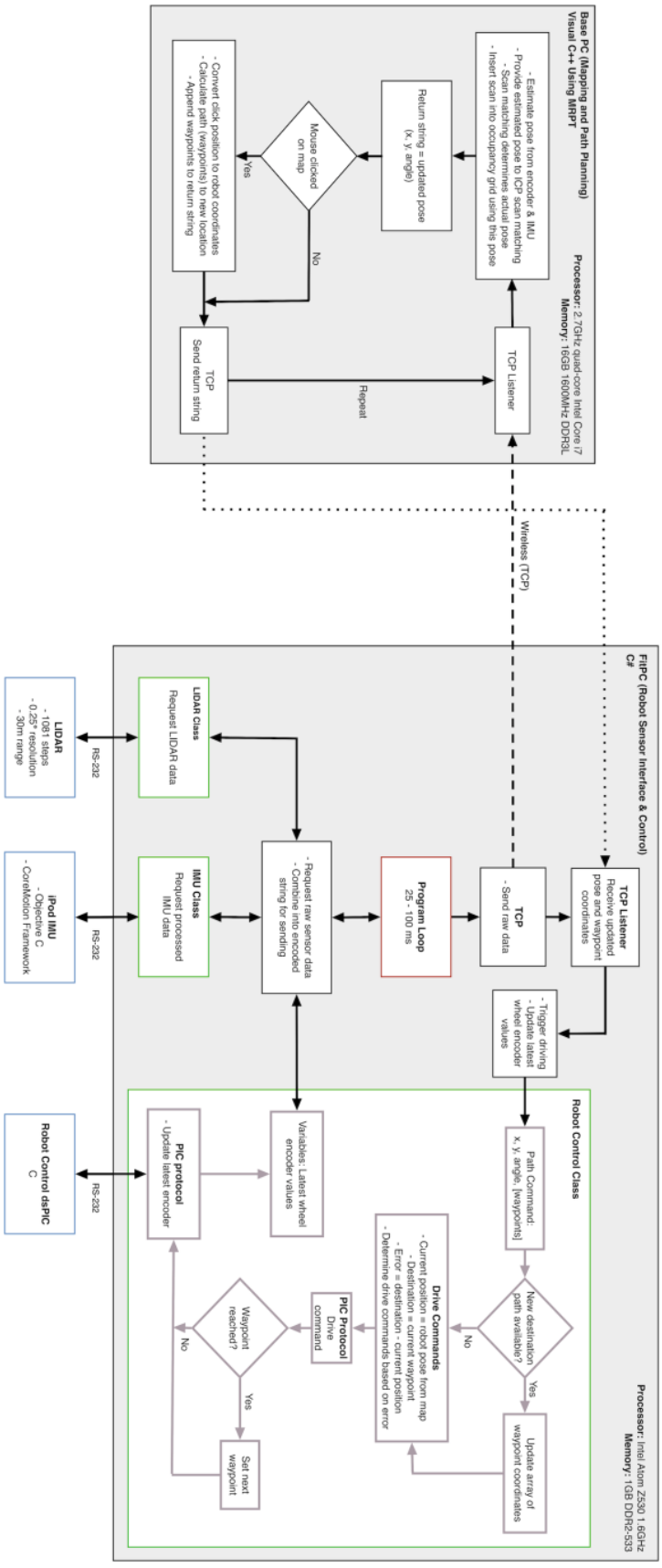


Figure 20: Detailed diagram of how the program is structured

9. Program/Mapping Process

This chapter will delve into a detailed account of how the program actually works, looking at all the important functions written to create the project.

The programming was broken down into two main programs, with one running on the robot and one on the base-PC. The program running on the robot was written by Scott and Ken in C#, with its purpose being to collect the sensor data, transmit it via TCP connection to the base PC, then decode returned instructions and pass these down a level to the PIC board. Since I had little part in writing this program it will only be referred to in a general sense, for a deeper look at how it functions refer to Scott and Ken's thesis or Appendix B.

The base PC program was written from the ground up, based around the MRPT libraries. It was written in a linear, easy to follow way so errors were avoided. The program runs on a single thread, but it is possible to modify it so that it multi-threaded to improve performance but this is another area for **future work**. A general flowchart of how the program works is in Figure 21: Base PC program block diagramFigure 21 and the full program code is in Appendix A.

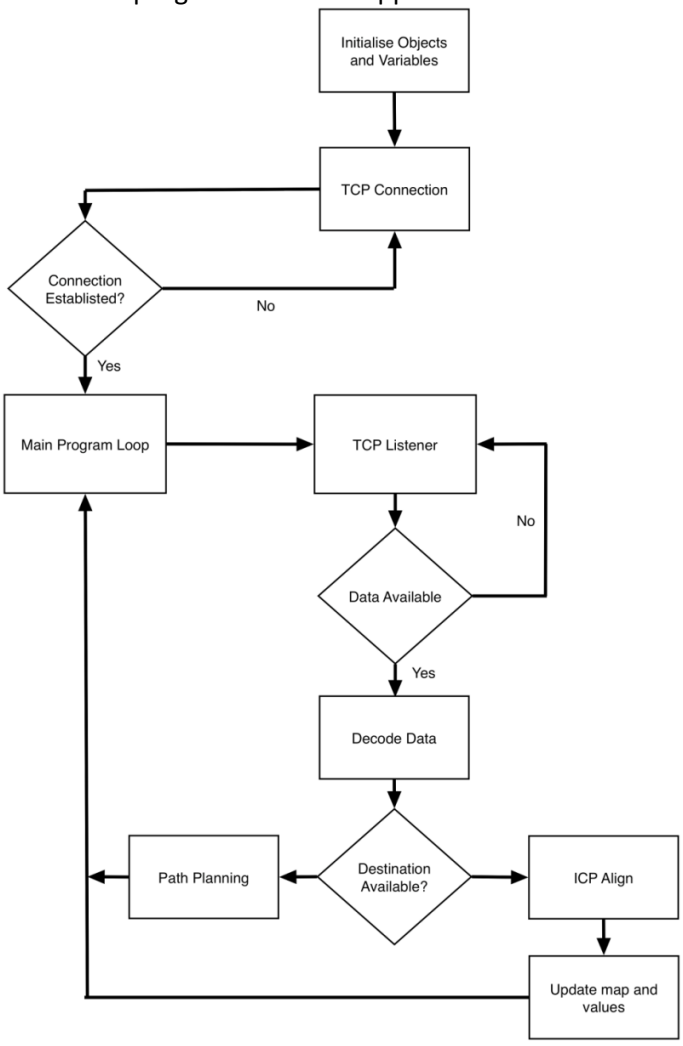


Figure 21: Base PC program block diagram

9.1 Initialise Objects and Variables

The program works by first defining all objects and setting any required parameters and options. Objects like the grid map and point clouds are defined here along with their respective resolutions.

9.2 TCP Connection

The program then calls a function call `TCPconnection()`. The purpose of this function is to setup and establish a TCP connection with the robot. The program idles here until a connection is established with the robot.

9.3 Main Program Loop/TCP Listener

Once a connection is established the program enters a continuous main loop. It calls the function `TCPlistener()`. In this function the program monitors the TCP connection and waits for data to become available. The code for setting up and monitoring the TCP connection in `TCPconnection()` and `TCPlistener()` was taken from the MSDN .NET Framework examples and modified to suit our application.

The data being sent from the robot can be at a maximum speed of 40Hz, so new data can be arriving up to every 25ms. A buffer is defined in this function to store any data being sent, but the buffer is defined as only 3258 bits long so that it will only store the last set of data sent by the robot. The purpose behind this is so that if the base PC is not processing the data as fast as it's coming in it will just skip any scans it misses rather than falling behind trying to process a large buffer of data before it overflows. In practice the base PC processes a set of data in about 70ms, so it is only receiving every third set of data being sent. The processing time is very dependent on the setting defined at initialisation like map resolution, ICP accuracy and number of iterations, and point cloud detail. Reducing all these to a minimum will reduce processing time to well below 25ms but reduces accuracy too much, resulting in a relatively poor map.

It is also during the TCPlistener function that a response is sent back to the robot. This response will compose of the robot's current pose, but it will also contain drive instructions in the form of waypoints if a destination on the map has been clicked. For more information on how these waypoints and drive instructions work refer to Scott or Ken's thesis.

9.4 Decode Data

The `TCPlistener()` function makes calls to the function `DecodeLine()` which receives the string of data from the robot and splits it up into segments of 3 bytes. This is done because of the way the data is encoded on the robot before it is sent to the base PC.

The robot sends 1086 float values, 1081 of which are range values from the LIDAR, 1 value for the yaw reading from the IMU, then 4 values from the encoder wheels (2 for encoder count of left and right, and 2 for overflow flag). In the Visual Studio IDE a float value is 4 bytes long, or 32 bits. To send it value in its original state would require 34752 bits, and while this isn't overly excessive it can definitely be simplified. By taking the binary representation of each value, then splitting it into 3 sections of length 6 and getting the hexadecimal equivalent then converting to ASCII the amount of data sent is reduced.

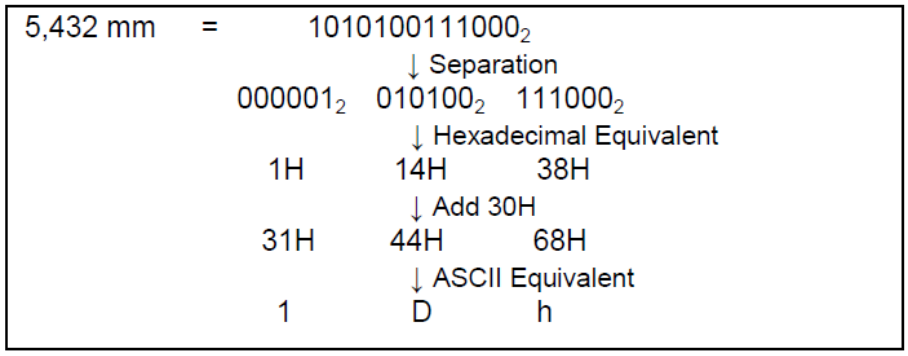


Figure 22: Three character encoding example

This method means that every string sent will be 3258 bits long, and this can then also be used as a check at the other end to ensure all the data has been received before continuing.

The `DecodeSegment()` function is directly responsible for taking the 3 bit segment and decoding it back into a float value.

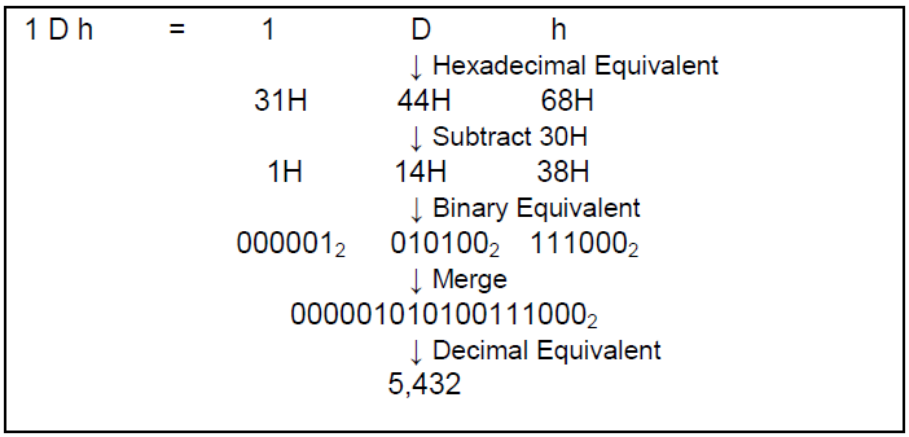


Figure 23: Three character decoding example

9.5 Destination Available?

Once the data has been decoded it checks if the screen has been clicked to generate a path for the robot to travel. If it has then it goes to path planning, if not then it goes to localisation.

9.6 Path Planning

If the map has been clicked then this function runs instead of the localisation and map updating functions. Once the path is calculated and drawn into the map, it returns to the main loop ready to receive the next lot of data from the LIDAR. This section was handled by Scott and Ken, for more detailed information on this function refer to Appendix A or Scott and Ken's theses.

9.7 ICP Align

Using the newest data received the ICP align algorithm is run. It is passed through the previous point cloud, the most current point cloud, and an estimation of the transformation between the two. It then outputs a PDF representing the calculated transformation along with data regarding run time, number of iterations executed, and accuracy of the alignment. If convergence of the two point clouds corresponds to a 1 to 1 match then a goodness value of 1 is given. With each scan we aim to get a value close to or equal to 1. Code was implemented in earlier versions to look at the goodness value and if it was below a certain threshold to exclude it from the map, but this would sometime lead to the case of the map breaking down and never recovering is position without any visual indication of what had gone wrong.

The ICP align algorithm is implemented through the MRPT library. Using the function from the library it is possible to pass either two points clouds (as was done in this case) or a point cloud and a reference grid map. While it would make sense to reference the newest scan against the whole map being built, in practise it didn't function as expected and gave very poor results.

The implementation of the align algorithm was one area fundamentally flawed in the code. In theory if each scan was referenced to the whole map the results produced would be more accurate along with map being more robust. Even if a single scan was slightly off, the next scan should be able to realign itself to the map. This will be discussed further in **future work**.

Another flaw with the algorithm came with the estimated transformation between point clouds. The estimation being passed was actually the transformation calculated on the previous loop (output of the previous iteration of ICP align). This means the estimated transformation being passed is actually not the current estimation. In testing this as found to still work, and work quite well but it would never be possible to get a perfect estimation using this method. Attempts were made to calculate the estimated transformation using the Yaw from the IMU and the odometry from the wheel encoders, but the calculated transformations were far too inaccurate to work well. Possible methods for correcting this will also be discussed in **future work**.

9.8 Update map and values

Once the transformation between point clouds is calculated it is added to the robots pose. An absolute value of the pose is also calculated using the robot pose PDF, which is done by taking the mean value of each Gaussian distribution (x,y,θ). The grid map is then updated by inserting the current scan into the map from the calculated pose. The current scan is then saved over the previous scan and the loop returns to receive new data and run the process again.

Like in the ICP align function, this method is not performed in the most efficient way. The grid map is simply updated from the robots pose, but the pose is then never checked using data from the grid map. This means when mapping an area and then going back over what has already been mapped the program is never checking the pose against data that is already in the map, leading to ghosting in the map and an overall poor looking map. While efforts were made to correct this, time constraints and code failure prevented it from happening, this is then another section that will be looked at in **future work**.

10. Results

Early in development the first step was to get the mapping working. This was achieved pretty early on, allowing work to move onto the localisation method. After a number of attempts to implement the EKF and just as many failures, work went to implementing localisation using Scan matching.

In early development with the ICP algorithm implemented the program was producing some very unexpected and undesired results. Quite often when turning the robot the mapping process would break down and become irreversibly broken.

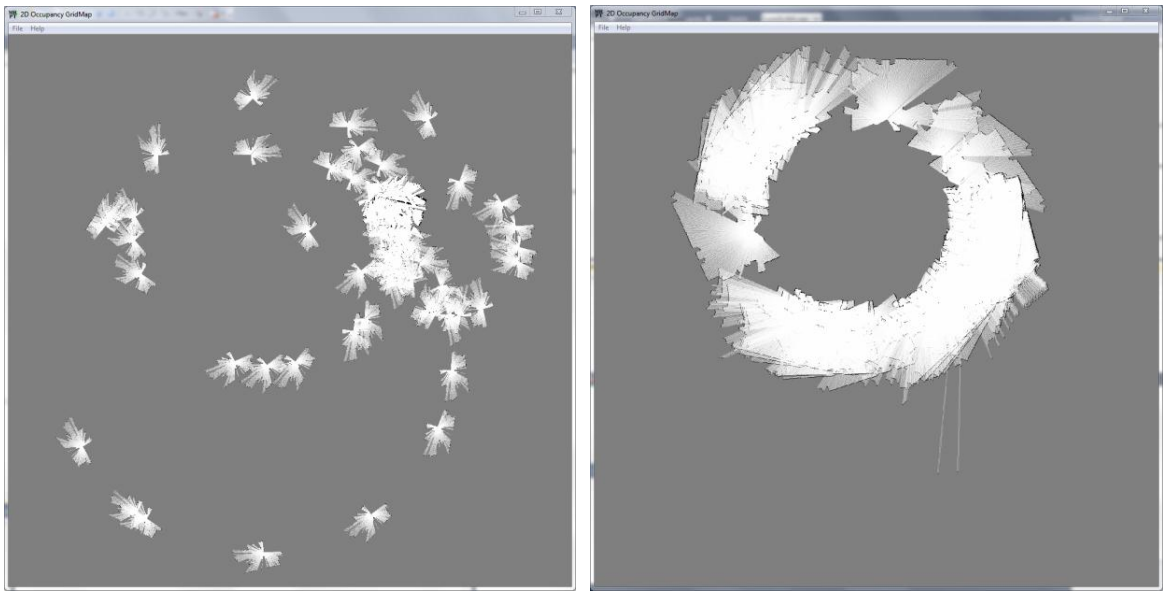


Figure 24: Screenshot of bugged mapping

After around three weeks, the problem was eventually found to be in the estimated pose being passed to the ICP algorithm, rather than the estimate of the transformation between point clouds being passed to it. This meant the algorithm was being initially told the consecutive point clouds were further and further apart as the robot travelled along.

Once this was fixed the mapping worked much better and very rarely broke down, giving a testament to the robustness of the ICP algorithm in the MRPT library. From here it was about tweaking the mapping to get improved results. If the robot was left in one position for long enough it was seen that the map would slowly start to drift around, due to the compounding minute errors calculated in the pose. This was fixed by tweaking configurations in the ICP algorithm and increasing the number of points in the point clouds being aligned.

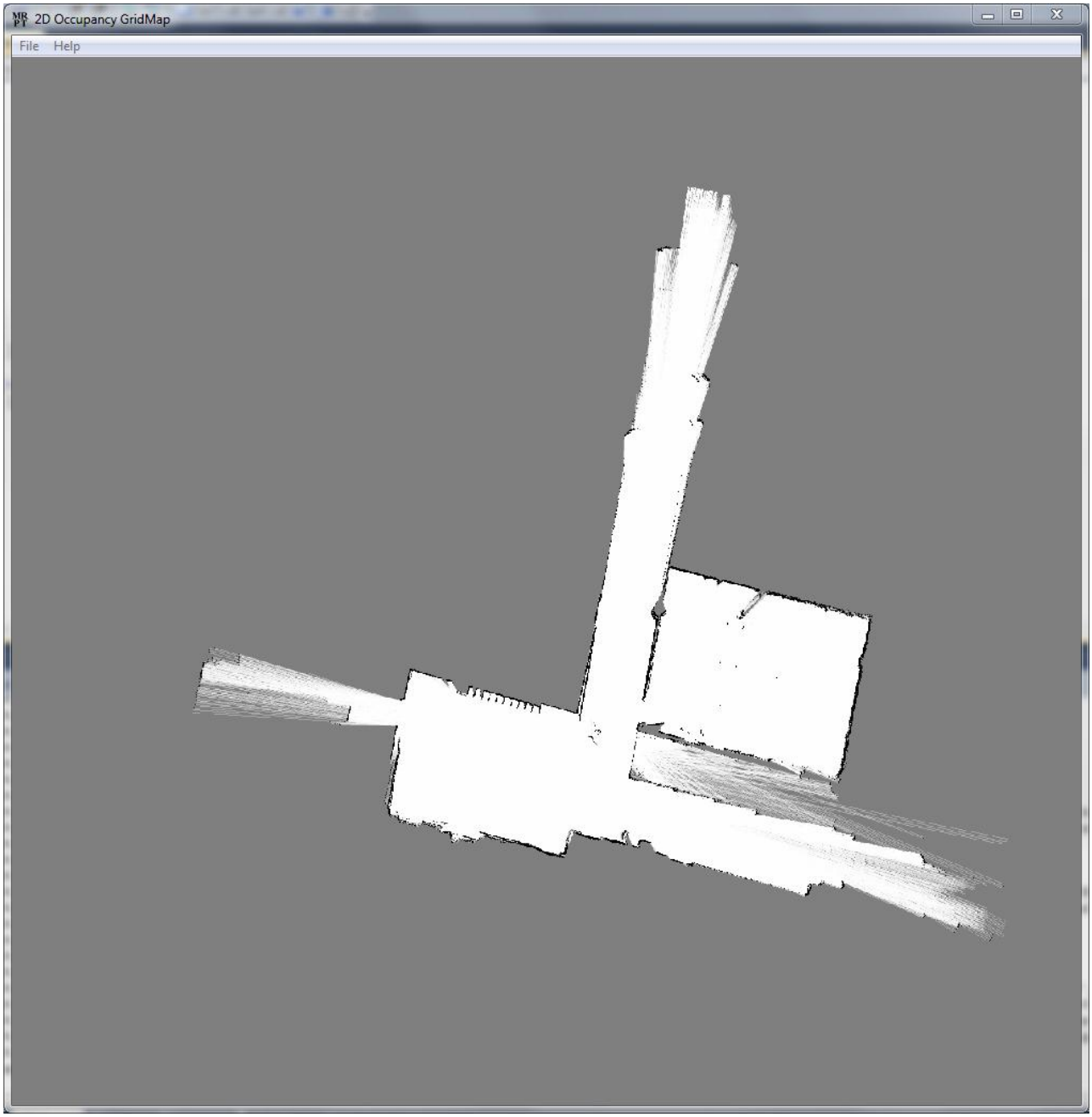


Figure 25: Good example of mapping when working correctly

Figure 25 shows one of the better maps produced during testing. The map shows up a lot of detail due to the high resolution of the LIDAR, you can notice the pillar in the hallway.

Figure 26 illustrates how the ICP algorithm struggles on corners or linear areas where it finds it difficult to converge the point clouds seen. It was found that if doors were exited backwards and corners were taken at a wide angle the scan matching was far more successful, as the point clouds recorded were far more unique and easier to match.

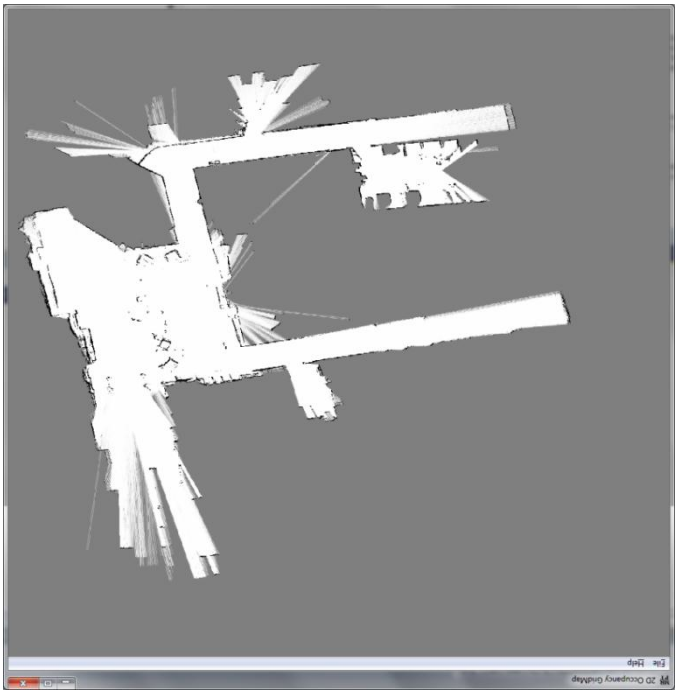


Figure 26: Example of how scan matching faults on corners

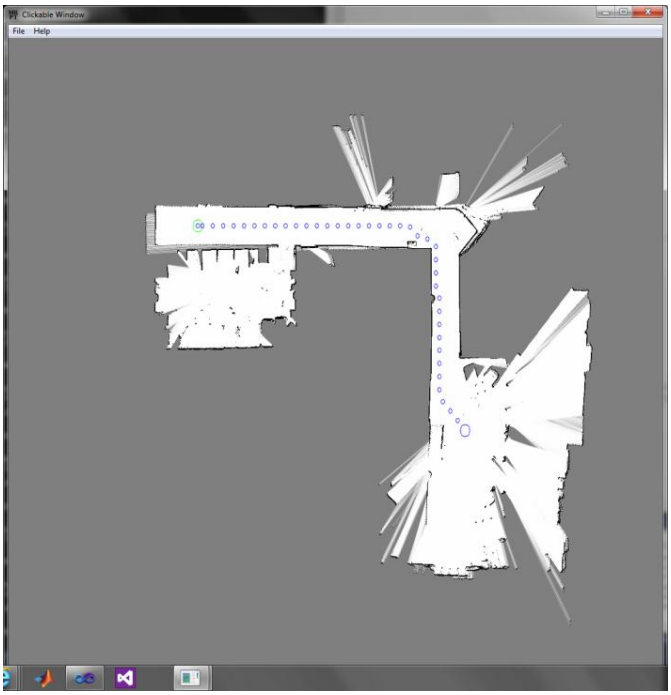


Figure 27: Mapping with Path Planning

Figure 27 shows the final program that handles simultaneous localisation and mapping along with path planning. When clicking on a point on the screen the path planning function will generate a path to that point and plot it on the map if possible, and then send the corresponding drive instructions to the robot to get it there.

11. Future Work

While the project was an overall success, the initial goals set out to be achieved were not met. There was also a lot of information learnt throughout the project that would have been extremely beneficial to know at the onset of the project. In this chapter we will detail all the recommendations for any future work that may be done if the project is picked up by someone else.

11.1 Robot Design

As detailed in Chapter 7 Robot Kinematics, it was seen that the robot base was a poor design. This is one major aspect holding the project back from achieving much greater accuracy in mapping and localisation. For this reason it would be strongly recommended to use a completely different base.

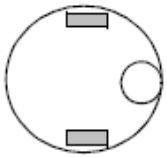
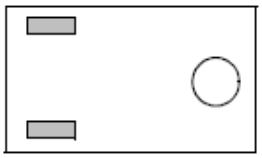
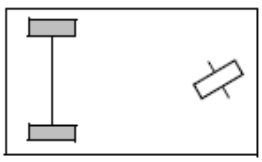
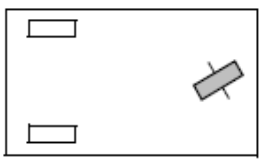
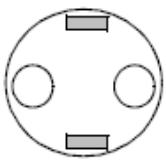
Arrangement	Description	Typical examples
	Two-wheel centered differential drive with a third point of contact	Nomad Scout, smartRob EPFL
	Two independently driven wheels in the rear/front, 1 unpowered omnidirectional wheel in the front/rear	Many indoor robots, including the EPFL robots Pygmalion and Alice
	Two connected traction wheels (differential) in rear, 1 steered free wheel in front	Piaggio minitrucks
	Two free wheels in rear, 1 steered traction wheel in front	Neptune (Carnegie Mellon University), Hero-1
	Two-wheel differential drive with 2 additional points of contact	EPFL Khepera, Hyperbot Chip

Figure 28: Possible wheel configurations for robot base

Any of the designs or a derivative of them in the above table would provide an excellent base for mapping and localisation. It is important to have a layout that reduces or eliminates slippage. It is also important for the wheel motors to retain high encoder count ratios too, as this will help produce a more accurate reflection of the correct odometry.

Similarly, the wheel design is also very important. If the robot is going to be used outside on rough terrain then it is important to have wide, well-treaded tyres like those present on the robot currently. Although if it's intended use is going to be indoors as would be expected, then the tyres used should be rigid and reasonably thin. This way they will not deform under weight or use, and reduce slippage of the wheel when turning to again increase odometry accuracy.

The use of a different, more accurate IMU would be another improvement. The IMU data went unused in this project from both the Sparkfun IMU and iPhone as the accuracy of the magnetometer was inefficient. Testing on the Sparkfun IMU found that anything conductive interfered with the magnetometer, including walls, power lines and even table legs. This made the Yaw readings completely useless, which was the most desired reading from the IMU. The other problem was that the data was fused together into roll, pitch and Yaw before it is sent to the robot. The only way to change this would be to reprogram the IMU's Firmware. While the iPhone didn't present as much interference it still showed drift and a lack of accuracy. The iPhone IMU is also not designed for this form of use, and was therefore not the best choice of hardware, but it did provide an easier way to send back the data from each sensor individually.

For future work it would be worth taking the raw data from the IMU's gyroscope, accelerometer, and magnetometer and passing these straight to the base PC. There are functions present in the MRPT libraries for storing this data as observations, which can then be used in other functions for calculating robot movements. More work would need to be done into working out exactly how to best use this data.

There are also some interesting developments in IMU technology available. There are fibre optic gyroscopes available that return extremely accurate data, while being free of interference from electrical sources, but carry a larger price tag. If work was to continue on the Lynx platform it would definitely be worth investing in a powerful and accurate IMU.

11.2 Kalman Filter implementation

With more time and understanding of C++ code the next step would have been to implement an Extended Kalman filter as the main algorithm for localisation. The EKF offers an accurate yet reasonably easy to use and implement localisation algorithm. Using the EKF as the main algorithm and then using scan matching at a fine tuner would help produce a much more accurate and repeatable mapping process. Looking at the MRPT libraries, there is a lot written on implementing the EKF, but it required the implementation of landmarks in the mapping process, so this would require a lot of additional work to get it implemented, or a possible change in the mapping process

(topological or hybrid). Another choice would be to look at other libraries available, such as those from ROS.org.

11.3 Operating system to Linux or ROS

If starting over from scratch one change would be the software platform the programs were built on. It may have proved advantageous to use Linux as the operating system on the base PC. This would have allowed use of libraries from the ROS.org packages, as well as possibly improving performance and speed of calculations during the ICP algorithm and mapping process.

The Windows platform also posed problems when programming the TCP connection, as Winsock functions had to be used over general socket programming.

The robots FitPC may have to remain on the windows platform though as the code for most of that program was written in C#, a .NET based language.

11.4 Improvements in scan matching

As mentioned in chapter 9.7 ICP Align, the alignment algorithm and scan matching wasn't optimal. With more time the program would have been altered so that scan matching was always done in a way that referred back to the map being built. That way, localisation would be reliant on the map, and the map reliant on the localisation, helping to produce a more stable map.

Issues were faced when using the MRPT libraries to convert between point clouds and occupancy maps. If these issues were resolved the map could be saved as an overall point cloud, which in turn could be used directly in the scan matching algorithm. From there the occupancy map could be built straight from the point cloud and updated whenever it was needed, rather than on every iteration of the program loop.

Another option would be to use different functions within the MRPT libraries. There is a function called ICPmapBuilder, which handles a lot of the hard work. All that needs to be done is passing the function the scans along with movement observations and then the map is calculated from there. It stores all previous scans and pose data within the function so other algorithms that use previous data can also be implemented such as a particle filter. The problem was that there was no work done on implementing this method in real time, and while the comments in the MRPT documentation suggest it would be possible, there was no evidence of it being done. The example program of it ran offline using logged data in a specific format. The other problem with this method is that it may be too simplified and you don't get to see how the operations are working below it, meaning a good understanding of how the method works may be missed.

More accurate sensors or better use of current sensors, combined with a refined scan matching process would definitely aid in creating a more accurate map.

11.5 Microsoft Kinect

A lot of early work was done by Scott with the Microsoft Kinect, but this work was not used and the Kinect was removed from the robot. This is because we found there wasn't anything we could use the Kinect for that the LIDAR couldn't already do within our time constraints. The Kinect did have some definite advantages though, with its ability to see in 3D, track people or function in a similar way to the LIDAR it would be beneficial to explore methods to implement it. With SLAM (simultaneous localisation and mapping) the more sensor data that can be fed into the mapping program, the more accurate the output will be, and the Kinect is a very powerful sensor.

It was also proposed to use the Kinect to detect object the LIDAR wouldn't, such as stairs or edges hanging out of the 2D plain, seen by the LIDAR and implement these into the map so the robot would be able to avoid them.

11.6 Hybrid mapping

Another possibility for future work is to look into the method of hybrid mapping. Looked at in chapter 5.3, it is the process of using both topological and metric mapping in the one map. It offers the benefits of both, while reducing the disadvantages of both. It is much more difficult to implement and understand though, so it would require a large amount of work. The benefits it would offer may tie in with improved scan matching too, as scans could be done against nodes of the map, rather than the entire map, which would increase the processing complexity. It may also go a long way to helping the implementation of landmarks and landmark recognition.

11.7 Increased functionality of mapping program

With more time it would have been good to implement more polished features into the mapping program. If the program lost connection or the map broke down there was no way to reset the map or re connect the robot on the GUI, instead the program had to be restarted. With more work on this project it would be good to implement some of these features, or the ability to record or screenshot the current map with a key press.

11.8 Path Planning and Autonomous exploration

The path planning and following was only implemented very late in the project due to time constraints, and therefore wasn't fully functional or optimised. With more time these methods would definitely need to be improved and implemented correctly. Refer to Ken and Scott's theses for more details on exactly what else needs to be done in this area.

12. References

Beevers, K. R., & Huang, W. H. (n.d.). *Loop Closing in Topological Maps.* New York: Rensselaer Polytechnic Institute, Department of Computer Science.

CMake. (2012). *Welcome Page*. Retrieved from cmake.org: http://www.cmake.org/

Gutmann, J.-S., & Konolige, K. (n.d.). *IncrementalMapping of Large Cyclic Environments.*

Konolige, K., Marder-Eppstein, E., & Marthi, B. (n.d.). *Navigation in Hybrid Metric–Topological Maps.* Willow Garage, Inc.

MRPT. (2012). *Mobile Robot Programming Toolkit*. Retrieved from www.mrpt.org

OpenCV. (2012). *About OpenCV*. Retrieved from willowgarage: http://opencv.willowgarage.com/wiki/Welcome

Park, S., Kim, S., Park, M., & Park, S.-K. (2008). Vision-based global localization for mobile robots with hybrid maps. *ScienceDirect*, 4174-4198.

Riisgaard, S., & Blas, M. R. (n.d.). *SLAM for Dummies.*

Siegwart, R., & Nourbakhsh, I. R. (2004). *Introduction to Autonomous Mobile Robots.* London, England: The MIT Press.

Sparkfun Electronics. (n.d.). *9 Degrees of Freedom - Razor IMU - AHRS compatible*. Retrieved 2012, from Sparkfun Electronics: https://www.sparkfun.com

Tomatis, N., Nourbakhsh, I., & Siegwart, R. (n.d.). *Simultaneous Localization and Map Building: A Global Topological Model with Local Metric Maps.*

wxWidgets. (2012). *about wxWidgets*. Retrieved from wxwidgets.org: http://www.wxwidgets.org/

Appendix A – Base PC Program Code

LynxSLAM.cpp

```
#include "LynxSLAM.h"

#include <msclr\marshal.h>
using namespace msclr::interop;

#define WINDOW_WIDTH 1000
#define WINDOW_HEIGHT 1000

#define ROBOT_COLOR TColor::blue
#define DESTINATION_COLOR TColor::green
#define PATH_COLOR TColor::red

#define SCAN_SIZE 1086
float Scan_Ranges[1081];
char Scan_Valid[1081];
vector<float> LIDAR_VEC;
float Yaw, deltaYaw, prevYaw, deltaX, deltaY, Dist, prevDist, deltaDist;
int WheelEncoderLeft, WheelEncoderRight, WheelOverflowLeft, WheelOverflowRight, WheelPrev, deltaWheel;
CTicTac tictac;
bool TimerRunning = false, yawRunOnce = false, WheelMoved = true;

string sendData = "nodata";//" + "Cat";
bool Planning = false;

int robotRadius = 0.25f;
int R;

public ref class Globals abstract sealed {
public:
   static TcpClient^ client;
   static System::String^ scanData;
};

class LynxMap : CObserver
{
public:
          int mouseX;
          int mouseY;

          LynxMap()
          {
                    newMouseClick = false;
                    window.resize(WINDOW_WIDTH, WINDOW_HEIGHT);
                    window.setWindowTitle("Clickable Window");
                    observeBegin (window);
          }

          bool NewDestinationAvailable()
          {
                    return newMouseClick;
          }

          int* GetDestination()
          {
                    int *ret = new int[2];

                    ret[0] = mouseGridX;
                    ret[1] = mouseGridY;

                    newMouseClick = false;
                    return ret;
          }

          void SetPath(std::deque<poses::TPoint2D> newPath)
          {
                    thePath = newPath;
          }

          void UpdateWithGridMapAndPose(COccupancyGridMap2D gridmap, CPose2D *pose)//double poseX, double poseY)
          {
                    double poseX = pose->x();
                    double poseY = pose->y();

                    // Store latest gridmap size to be used by mouse click event
                    xCells = gridmap.getSizeX();            //number of cells in x direction
                    yCells = gridmap.getSizeY();            //                                          y

                    // Get gridmap as image: resolution: 1 cell = 1 image pixel
                    gridmap.getAsImage(gridMapImage, true, true, false);
```

```
                    // Annotate robot pose on image
                    int poseCellX = gridmap.x2idx(poseX);//transform xPose coordinate (from meters) into a cell index = pixel
index
                    int poseCellY = gridmap.y2idx(poseY);                                                    //repeat for y
          // yCells - 1 -
                    //int R = round(robotRadius / gridmap.getResolution());
                    gridMapImage.drawCircle(poseCellX, poseCellY, R, ROBOT_COLOR);
                              //Draw position of robot over map
                    gridMapImage.drawCircle(poseCellX, poseCellY, 8, ROBOT_COLOR);

                    //Window has been scaled, therefore 1pixel != 1cell
                    //get mouse in world coordinates to match pose (meters)
                    mouseGridX = ((float)xCells/WINDOW_WIDTH) * mouseWindowX;
                    mouseGridY = ((float)yCells/WINDOW_HEIGHT) * mouseWindowY;
                    gridMapImage.drawCircle(mouseGridX, mouseGridY, R, DESTINATION_COLOR); //Draw position of cursor over map
                    gridMapImage.drawCircle(mouseGridX, mouseGridY, 8, DESTINATION_COLOR);

                    for (std::deque<poses::TPoint2D>::const_iterator it=thePath.begin();it!=thePath.end();++it)
                              gridMapImage.drawCircle( gridmap.x2idx(it->x), gridmap.y2idx(it->y), R, PATH_COLOR );

                    gridMapImage.scaleImage(WINDOW_WIDTH,WINDOW_HEIGHT); //TODO: Look into interpolation method
                    //-------------------------------------------------------------------------------------------------------
----------------------

                    window.showImage(gridMapImage);
          }

          void OnEvent(const mrptEvent &  e)
          {
                    if (e.isOfType<mrptEventMouseDown>())
                    {
                              const mrptEventMouseDown* ev = e.getAs<mrptEventMouseDown>();

                              mouseWindowX = ev->coords.x;
                              mouseWindowY = ev->coords.y;
                              newMouseClick = true;

                              //Window has been scaled, therefore 1pixel != 1cell
                              //get mouse in world coordinates to match pose (meters)

                              mouseGridX = ((float)xCells/WINDOW_WIDTH) * mouseWindowX;
                              mouseGridY = ((float)yCells/WINDOW_HEIGHT) * mouseWindowY;
                    }

          }

private:
          int mouseWindowX, mouseWindowY;
          int mouseGridX, mouseGridY;
          bool newMouseClick;

          CImage gridMapImage;
          std::deque<poses::TPoint2D> thePath;

          unsigned int xCells, yCells;            //updated each with each gridMap update
          CDisplayWindow window;
};

          //function to update the pose from odometry data, currently unused.
void UpdatePose()
{
                    //records current distance then subtracts previous distance value to get a delta distance from last scan
                    //can probably be achieved more acurately using MRPT functions that do a similar task
          Dist = (WheelEncoderLeft / 4220.0f) + ((WheelOverflowLeft/3) * 15.5296);
          deltaDist = Dist - prevDist;
          deltaX = (deltaDist * cos (Yaw));
          deltaY = (deltaDist * sin (Yaw));
          cout << "Distance: " << Dist << endl;
          prevDist = Dist;
}

          //converts the 3 byte characters into a float value
int DecodeSegment(System::String^ codedData)
{
    // encoding technique:
    // separate data into groups of 6 bits
    // 30H (48) added to convert to ASCII
    int bit, nBitEncoding = codedData->Length;
    int val = 0;
    for (int i = 0; i < nBitEncoding; ++i)
    {
        bit = nBitEncoding - i - 1;
        val += pow((double)2,6*i)*(Convert::ToInt32(Convert::ToChar(codedData->Substring(bit, 1))) - 48);
    }
    return val;
}

          // This will be an ASCII string of length 1080 * 3 bytes
void DecodeLine(System::String^ scan)
{
```

```
		//get IMU value from first 3 bytes
		WheelEncoderLeft = DecodeSegment(scan->Substring(0, 3));
		WheelEncoderRight = DecodeSegment(scan->Substring(3, 3));
		WheelOverflowLeft = DecodeSegment(scan->Substring(6, 3));
		WheelOverflowRight = DecodeSegment(scan->Substring(9, 3));

			//Yaw value given in radions from 0 - 2PI
		Yaw = DecodeSegment(scan->Substring(12, 3))/1000.0f;
		Yaw -= DEG2RAD(180.0f);
		UpdatePose();
		if (!yawRunOnce)
		{
			prevYaw = Yaw;
			yawRunOnce = true;
		}
		deltaYaw = prevYaw - Yaw;
		if (deltaYaw < -180)
			deltaYaw += 360;

		for (int i = 0; i < 1081; i++)
	{
			Scan_Ranges[i] = DecodeSegment(scan->Substring(((i*3)+15), 3))/1000.0f;
	}
}

		//this function establishes the initial TCP connection to the robot
void TCPconnection()
{
		try
		{
				//Set the TcpListener on port 13000.
			Int32 port = 13000;
			System::String ^flindersIP = "172.16.253.133";
			System::String ^csemIP = "129.96.58.109";
			IPAddress^ localAddr = IPAddress::Parse(csemIP);

				//TcpListener* server = new TcpListener(port);
			TcpListener^ server = gcnew TcpListener( localAddr,port );

				//Start listening for client requests.
			server->Start();

				// Enter the listening loop.
			Console::Write("Local IP: ");
			Console::WriteLine(localAddr);
			Console::Write("Waiting for a connection... ");

				//Perform a blocking call to accept requests.
				//You could also user server.AcceptSocket() here.
			Globals::client = server->AcceptTcpClient();
			Console::WriteLine("Connected!");
		}
		catch (SocketException^ e)
		{
			Console::WriteLine( "SocketException: {0}", e );
		}
}

		//function for recieving in and decoding data sent from the robot
bool TCPlistener()
{
		bool allDataReceived = false;
		try
		{
				//Buffer for reading data
			array<Byte>^bytes = gcnew array<Byte>(SCAN_SIZE*3);
			String^ sData = nullptr;

				// Get a stream Object* for reading and writing
			NetworkStream^ stream = Globals::client->GetStream();

			if (TimerRunning == true)
				cout << "Single Run time: " << tictac.Tac() << endl << endl;

			while (!(stream->DataAvailable))
			{
				stream = Globals::client->GetStream();
			}
			tictac.Tic();
			TimerRunning = true;

			Int32 i;

			i = stream->Read(bytes, 0, bytes->Length);

				//Translate data bytes to a ASCII String*.
			sData = Text::Encoding::ASCII->GetString(bytes, 0, i);

			Globals::scanData = Globals::scanData + sData;
			int len = Globals::scanData->Length;
```

```
                    if (len >= SCAN_SIZE*3)
                    {
                              DecodeLine(Globals::scanData->Substring(0, SCAN_SIZE*3));
                              if (len > SCAN_SIZE*3)
                              {
                                        Globals::scanData = Globals::scanData->Substring(SCAN_SIZE*3);
                                        cout << len << endl;
                                        cout << Globals::scanData->Length + SCAN_SIZE*3 << endl;
                              }
                              else
                              {
                                        Globals::scanData = nullptr;
                              }
                              allDataReceived = true;
                    }
          // received data is in a managed string (System::String), need to Marshal as unmanaged string (std::string).
          //Process the data sent by the client.

//SendData is the data being sent back to the FitPC which includes information about the robots pose and any new path data
                    String^ SendData = "";
                    SendData = marshal_as<String^>(sendData);

                    array<Byte>^msg = Text::Encoding::ASCII->GetBytes(SendData);

                              //Send back a response.
                    stream->Write(msg, 0, msg->Length);
          }
          catch (SocketException^ e)
          {
                    Console::WriteLine( "SocketException: {0}", e );
          }
          return allDataReceived;
}

int main(int argc, char ** argv)
{
          LynxMap lynxMap;
                    //create an array of 1's for use in validating laser scan later.
          for(int i = 0; i < sizeof(Scan_Valid); ++i)
          {
                    Scan_Valid[i] = 1;
          }
                    //define image that will contain the gridmap
          CImage
          bmpImg;
                    //define the gridmap
          COccupancyGridMap2D                                                                   gridmap;
                    //create 2 points maps for holding laser range scan data
          CSimplePointsMap                                                                      m1,m2;
          m1.insertionOptions.minDistBetweenLaserPoints     = 0.00f;
          m2.insertionOptions.minDistBetweenLaserPoints     = 0.00f;
          m1.insertionOptions.isPlanarMap                                             = true;
          m2.insertionOptions.isPlanarMap                                             = true;
          float
          runningTime;
          CICP::TReturnInfo                                                                     info;
          CICP                                                                                                                  ICP;
          // ---------------------------------------------------
//        ICP.options.ICP_algorithm                                           = icpLevenbergMarquardt;
          ICP.options.ICP_algorithm                                           = icpClassic;
                    // The maximum number of iterations to execute if convergence is not achieved before
          ICP.options.minAbsStep_trans                                = 1e-6;
          // If the correction in all rotation coordinates (yaw,pitch,roll) is below this threshold (in radians), iterations
are terminated:
          ICP.options.minAbsStep_rot                                          = 1e-6;
                    // The maximum number of iterations to execute if convergence is not achieved before
          ICP.options.maxIterations                                           = 80;
                    // An angular factor (in degrees) to increase the matching distance for distant points.
          ICP.options.thresholdAng                                            = DEG2RAD(5.0f);
                    // Initial maximum distance for matching a pair of points
          ICP.options.thresholdDist                                           = 0.3f;
          // After convergence, the thresholds are multiplied by this constant and ICP keep running (provides finer matching)
          ICP.options.ALFA                                                                      = 0.8f;
                    // This is the smallest the distance threshold can become after stopping ICP and accepting the result.
          ICP.options.smallestThresholdDist                           = 0.05f;
          // 1: Use the closest points only, 0: Use all the correspondences within the threshold (more robust sometimes, but
slower)
          ICP.options.onlyClosestCorrespondences            = true;

          ICP.options.doRANSAC                                                = false;
          ICP.options.corresponding_points_decimation = 5;
          ICP.options.dumpToConsole();
          // ---------------------------------------------------
                    //define a 2D scan for holding the lIDAR data
          CObservation2DRangeScan                                     LIDARscan;
                    //maximum range of URG-30LX is 30m accurately
          LIDARscan.maxRange                                          = 30.0f;
                    //scan range is 270 degrees, which is 4.71 radians
          LIDARscan.aperture                                          = DEG2RAD(270);
          LIDARscan.rightToLeft                                       = true;
```

```
	LIDARscan.stdError                                          = 0.03;
	LIDARscan.validRange.resize(1081);
	LIDARscan.scan.resize(1081);


		//set gridmap width, height, resolution and set all cells to unknown(0.5)
	gridmap.setSize(0,500,0,500,0.025,0.5);
	gridmap.clear();
		//create the TCP connection to the robot
	TCPconnection();
		//define a counter so the gridmap can clear it's layers every 5 scans
	int clearcount = 0;
		//listen for incoming data from the robot
	TCPlistener();
		//define the robot pose @ 0,0,0
	CPose2D             pose(0,0,Yaw);
		//define a Gaussian 2D pose
	CPosePDFGaussian posePDF(pose);

	ASSERT_( sizeof(Scan_Ranges) == sizeof(float)*1081 );
	memcpy(&LIDARscan.scan[0], Scan_Ranges, sizeof(Scan_Ranges));
	memcpy(&LIDARscan.validRange[0], Scan_Valid, sizeof(Scan_Valid));

	m1.insertObservation( &LIDARscan );
	CPosePDFGaussian  deltaPDF;

	R = round(robotRadius / gridmap.getResolution());

	while(true)
	{
		if(TCPlistener())
		{
			sendData.clear();

			char buffer [33];
			string x_y ="";
			int s;
			string str;

			if(lynxMap.NewDestinationAvailable())
			{
				int *destination;
				destination = lynxMap.GetDestination();

					// Find path:
				CPathPlanningCircularRobot    pathPlanning;
					// approximate radius of the robot if we treat it as a circle is about 20
cm, so made it 25 cm to be on the safe side
				pathPlanning.robotRadius = 0.25f;
					// consecutive coordinates that form the path data are at least 0.75 m
apart to reduce the amount of redundant information being sent back
				pathPlanning.minStepInReturnedPath = 0.75f;
					// a cell on the map must have an occupancy value of greater than 0.5 for
the robot to think it can go there
				pathPlanning.occupancyThreshold = 0.5f;

				std::deque<poses::TPoint2D>             thePath;
				bool      notFound;

				CPoint2D  origin( pose.x(), pose.y() );           // robots current pose

					//get from gui
					// where the robot is trying to get to
				CPoint2D  target( gridmap.idx2x(destination[0]), gridmap.idx2y(destination[1]) );


					// prints the destination corrdinates
				cout << target.x() << ", " << target.y() << endl << endl;

				pathPlanning.computePath( gridmap, origin, target, thePath, notFound, -1.0f /* Max.
distance */ );

				float originValue = gridmap.getPos(origin.x(), origin.y());
				float clickValue = gridmap.getPos(target.x(), target.y());

				lynxMap.SetPath(thePath);

					// prints true if no path was found
				cout << notFound << endl << endl;

					// constructs the string to be sent back to the FitPC
				s = gridmap.x2idx(pose.x());  //robots current x pose
				itoa(s,buffer,10);
				str = buffer;
				x_y = str;

					//robots current y pose
				s = gridmap.y2idx(pose.y());
				itoa(s,buffer,10);
				str = buffer;
```

```
                                        x_y = x_y + ',' + str;

                                                //robots current angle timed by 1000 to perserve the decimal point
                                        s = pose.phi()*1000;
                                        itoa(s,buffer,10);
                                        str = buffer;
                                        x_y = x_y + ',' + str;

                                                //the length of the path calculated
                                        s = thePath.size();
                                        itoa(s,buffer,10);
                                        str = buffer;
                                        x_y = x_y + ',' + str + ",";

                                        for (std::deque<poses::TPoint2D>::const_iterator
it=thePath.begin();it!=thePath.end();++it)      // the x and y corrdinates that form the path data
                                        {
                                                s = gridmap.x2idx(it -> x);
                                                itoa(s,buffer,10);
                                                str = buffer;
                                                x_y = x_y + str + ',';

                                                s = gridmap.y2idx(it -> y);
                                                itoa(s,buffer,10);
                                                str = buffer;
                                                x_y = x_y + str + ',';
                                        }
                                                //SEND X_Y STRING TO TCP LISTENER
                                        sendData = x_y;
                                        Planning = false;
                                }
                                else
                                {
                                        ASSERT_( sizeof(Scan_Ranges) == sizeof(float)*1081 );
                                        memcpy(&LIDARscan.scan[0], Scan_Ranges, sizeof(Scan_Ranges));
                                        memcpy(&LIDARscan.validRange[0], Scan_Valid, sizeof(Scan_Valid));

                                                //insert newest scan into pointmap m2
                                        m2.clear();
                                        m2.insertObservation( &LIDARscan);

                                                //run ICP algorithm on previous scan m1 and the new scan m2
                                        CPosePDFPtr pdf = ICP.AlignPDF(&m1, &m2, deltaPDF, &runningTime, (void*)&info);
                                        deltaPDF.copyFrom(*pdf);
                                        //update the reference point of the gaussian distribuiton by adding the deltaPDF
                                        posePDF.operator+=(deltaPDF);
                                        cout << "Scanmatch Time: " << runningTime << endl;
                                        cout << "Goodness: " << info.goodness << endl;
                                        cout << "posePDF Yaw: " << RAD2DEG(posePDF.mean.phi()) << endl;
                                                //create absolute pose value from posePDF
                                        pose.x(posePDF.mean.x());
                                        pose.y(posePDF.mean.y());
                                        pose.phi(posePDF.mean.phi());
                                        CPose3D                         pose3D(pose);
                                                //draw the observation in the map
                                        gridmap.insertObservation(&LIDARscan, &pose3D);
                                        lynxMap.UpdateWithGridMapAndPose(gridmap, &pose);

                                                //save current pointcloud as the old scan for the next iteration
                                        m1.clear();
                                        m1.insertObservation( &LIDARscan);

                                                // constructs the string to be sent back to the FitPC
                                        s = gridmap.x2idx(pose.x());  //robots current x pose
                                        itoa(s,buffer,10);
                                        str = buffer;
                                        x_y = str;
                                                //robots current y pose
                                        s = gridmap.y2idx(pose.y());
                                        itoa(s,buffer,10);
                                        str = buffer;
                                        x_y = x_y + ',' + str;

                                        //robots current angle timed by 1000 to perserve the decimal point
                                        s = pose.phi()*1000;
                                        itoa(s,buffer,10);
                                        str = buffer;
                                        x_y = x_y + ',' + str;

                                //the length of the path calculated which is 0 as there is no path calculated at this point
                                        s = 0;
                                        itoa(s,buffer,10);
                                        str = buffer;
                                        x_y = x_y + ',' + str + ",";
                                        sendData = x_y;

                                }
                        }
                }
}
```

LynxSLAM.h

```
#using <System.dll>

#include <mrpt/slam.h>
#include <mrpt/gui.h>
#include <mrpt/base.h>
#include <vector>
#include <iostream>
#include <fstream>
#include <string>
#include <stdlib.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <cmath>
#include <msclr\marshal_cppstd.h>
#include <mrpt/slam/CPointsMap.h>
#include <mrpt/utils/CObserver.h>

using namespace mrpt;
using namespace mrpt::gui;
using namespace mrpt::utils;
using namespace mrpt::slam;
using namespace mrpt::random;
using namespace System;
using namespace System::IO;
using namespace System::Net;
using namespace System::Net::Sockets;
using namespace System::Text;
using namespace System::Threading;
using namespace System::IO::Ports;
using namespace std;
```

Appendix B – Robot Program Code

Constansts.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LynxSLAM
{
    class Constants
    {
        public const int sampleRate = 100; //ms

        public const bool usingSparkfunIMU = false;

        //Encoding
        public const int encoding1st = 63;
        public const int encoding2nd = 4032;
        public const int encoding3rd = 258048;
        public const int encoding4th = 0;

        //Baud Rates
        public const int baudLynxRobot = 19200;
        public const int baudLIDAR = 19200;
        public const int baudIMU = 19200;

        //LIDAR
        public const int stepA = 0;         // Initial measurement step of detection
range
        public const int stepB = 540;       // Sensor front step
        public const int stepC = 1080;      // End point of detection range

        public const int yawDeg = 26;       // precision 2
        public const int yawDegEnc = 58;
        public const int yawRad = 14;       //precision 3 (current max)
        public const int yawRadEnc = 46;


        //Drive constants
        public const double deltaAngleThreshold = 15;
        public const int driveForwardSpeed = 30;
        public const int turnSpeed = 18;
        public const int distanceThreshold = 10;
    }
}
```

DataEncoding.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LynxSLAM
{
    public static class DataEncoding
    {
        #region encoding
        public static string EncodeTest(float value, int precision)
        {
            int valueToSend = (int)(value * Math.Pow(10, precision));//pow(10, precision);
            // split into upper, middle and lower 6-bits
            int upper = valueToSend & Constants.encoding3rd;      // upper 6  = 111111000000000000 = 258048
            int middle = valueToSend & Constants.encoding2nd;     // middle 6 = 000000111111000000 = 4032
            int lower = valueToSend & Constants.encoding1st;      // lower 6  = 000000000000111111 = 63

            // shift and add hex30 to convert to ascii
            upper = (upper >> 12) + 48;             // shift by 12, add hex30
            middle = (middle >> 6) + 48;            // shift by 6, add hex 30
            lower += 48;                            // add hex30
            string encoded = Convert.ToChar(upper).ToString() + Convert.ToChar(middle).ToString() +
Convert.ToChar(lower);
            return encoded;
        }
        #endregion
        #region decoding
        public static uint DecodeTest(string encodedData)
        {
            // encoding technique:
            // separate data into groups of 6 bits
            // 30H (48) added to convert to ASCII
            int bit, nBitEncoding = encodedData.Length;
            uint val = 0;
            for (int i = 0; i < nBitEncoding; ++i)
            {
                bit = nBitEncoding - i - 1;
                val += (uint)Math.Pow(2, 6 * i) * (Convert.ToUInt32(Convert.ToChar(encodedData.Substring(bit,
1))) - 48);
            }
            return val;
        }

        // Moved from LIDAR class
        // TODO: Check length of data provided matches encoding type
        private static uint[] DecodeLine(string scan, int nBitCoding)
        {
            int n = scan.Length / nBitCoding;
            uint[] data = new uint[n];
            for (int i = 0; i < n; ++i)
            {
                data[i] = DecodeSegment(scan.Substring(i * nBitCoding, nBitCoding));
            }
            return data;
        }
        // for decoding 2, 3 or 4 bit encoded data
        private static uint DecodeSegment(string codedData)
        {
            // encoding technique:
            // separate data into groups of 6 bits
            // 30H (48) added to convert to ASCII
            int bit, nBitEncoding = codedData.Length;
            uint val = 0;
            for (int i = 0; i < nBitEncoding; ++i)
            {
                bit = nBitEncoding - i - 1;
                val += (uint)Math.Pow(2, 6 * i) * (Convert.ToUInt32(Convert.ToChar(codedData.Substring(bit,
1))) - 48);
            }
            return val;
        }
        #endregion
    }
}
```

IMU.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.Ports;

namespace LynxSLAM
{
    class IMU : SerialConnection
    {
        private float roll, pitch, yaw;
        public string tempYaw;
        public bool tcpShouldConnect, eStop;

        public IMU() { }

        public override void serialPort_DataReceived(object sender,
SerialDataReceivedEventArgs e)
        {

            base.serialPort_DataReceived(sender, e);
            SerialPort sp = (SerialPort)sender;
            try
            {
                string dataIn = lastData[0];
                if (dataIn == "ConnectTCP")
                {
                    tcpShouldConnect = true;
                }
                else if (dataIn == "Stop")
                {
                    eStop = true;
                }
                else if (dataIn == "Start")
                {
                    eStop = false;
                }
                else
                {
                    tempYaw = dataIn;
                }
            }
            catch (Exception) { }
        }
    }
}
```

LIDAR.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;

namespace LynxSLAM
{
    struct LastScan
    {
        // implement this
        void GetStatus()
        {

        }
        private uint timeStamp;
        private bool laserOn;
        //...
    }

    class LIDAR
    {
        private SerialPort serialPort;
        private uint[] lastScan = new uint[0];
        private bool waiting = false;

        string fullScan = "";

        // configure properties or use struct??;
        public string serialPort_GetName()
        {
            string port = "";
            try
            {
                port = serialPort.PortName;
            }
            catch (Exception e)
            {
                port = e.GetType().ToString();
            }
            return port;
        }
        public bool serialPort_IsOpen()
        {
            bool open;
            try
            {
                open = serialPort.IsOpen;
            }
            catch (Exception)
            {
                open = false;
            }
            return open;
        }

        public LIDAR()
        {

        }

        #region comms
        public void InitComms(string port)
        {
            serialPort = new SerialPort(port);
            serialPort.DataReceived += new SerialDataReceivedEventHandler(serialPort_DataReceived);
            serialPort.BaudRate = 19200;
            serialPort.DataBits = 8;
            serialPort.Parity = Parity.None;
            serialPort.StopBits = StopBits.One;
```

```
            try
            {
                serialPort.Open();
            }
            catch (Exception e)
            {
                MessageBox.Show(e.Message);
            }
        }

        public string serialPort_sendCommand(string command, bool wait)
        {
            if (!serialPort_IsOpen())
            {
                return "";// new uint[0];
            }
            if (wait == true)
                waiting = true;
            else
                waiting = false;

            serialPort.WriteLine(command);
            if (waiting == true)       // Waiting is set to false after processing data
            {
                while (waiting) { } // Wait for sensor to return data
                return fullScan;
            }
            return ""; //new uint[0];     // Return received data
        }

        void serialPort_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {
            SerialPort sp = (SerialPort)sender;
            string currentLine = "";
            try
            {
                ArrayList lines = new ArrayList();
                //int byteCOUNT = sp.BytesToRead;//count the number of bytes in RX buffer
                while (sp.BytesToRead > 0)          // while data is in the RX buffer
                {
                    currentLine = sp.ReadTo("\n");    // get one line at a time
                    lines.Add(currentLine);
                }
                DecodeDataIn(lines);
                waiting = false;
            }
            catch (Exception) { }//e1
        }
        #endregion

        #region decoding
        private void ScanAddLine(string line)
        {
            fullScan = fullScan + line;
        }

        private void sendTCP(string send)
        {

        }

        // TODO: also look at status returned and sum (FIRST)
        private void DecodeDataIn(ArrayList lines)
        {
            string status = lines[1].ToString().Substring(0,2);

            if (status != "00") // 00: Command received without any error
                return;

            // TODO: Time stamp required?
            DecodeTimeStamp(lines[2].ToString().Substring(0, 4));

            // Line 0 is echo, use this to see if data is for a scan (MD, MS, GD or GS)
            int nBit;
            string command1 = lines[0].ToString().Substring(0, 1);
            string command2 = lines[0].ToString().Substring(1, 1);
            if (command1 == "M" || command1 == "G")
```

```
            {
                // Determine data encoding format
                switch (command2)
                {
                    case "D":
                        nBit = 3;   // 3 character encoding
                        break;
                    case "S":
                        nBit = 2;   // 2 character encoding
                        break;
                    default:
                        nBit = -1;  // Error
                        break;
                }
                if (nBit != -1)     // If no error
                {
                    // Collect all scan data into a single string
                    fullScan = "";

                    int count = lines.Count;
                    int maxLength = (int)(64 / nBit);

                    uint[] allData = new uint[((count - 5) * maxLength)
                                              + lines[count - 2].ToString().Length/nBit];

                    string currentLine;
                    int currentLength;

                    // Index is 1 less than count, last line is line feed.
                    for (int i = 3; i < count-1; i++)
                    {
                        currentLine = lines[i].ToString();
                        currentLength = currentLine.Length;

                        string nextLine = currentLine.Substring(0, currentLength - 1);
                        ScanAddLine(nextLine);
                    }

                    lastScan = allData;
                }
            }
            else
            {
                // TODO: Process these to update instance variables
                string command = command1 + command2;
                switch (command)
                {
                    case "BM":  // Laser on
                        break;
                    case "QT":  // Laser off
                        break;
                    case "RS":  // Reset all settings (Off, motor speed, bit-rate and timer)
                        break;
                    case "TM":  // Match host/sensor time (read spec)
                        break;
                    case "SS":  // Change bit-rate for RS232C
                        break;
                    case "CR":  // Motor speed (not compatible with UTM-30LX)
                        break;
                    case "HS":  // Switch between high/normal sensitivity
                        break;
                    case "DB":  // Simulate malfunction
                        break;
                    case "VV":  // Get version details (serial number, firmware...)
                        break;
                    case "PP":  // Get sensor specs
                        break;
                    case "II":  // Get sensor state
                        break;
                    default:
                        break;
                }
            }
        }

        private uint DecodeTimeStamp(string data)
        {
            return DecodeSegment(data);
```

```
        }

        // TODO: Check length of data provided matches encoding type
        private uint[] DecodeLine(string scan, int nBitCoding)
        {
            int n = scan.Length / nBitCoding;
            uint[] data = new uint[n];
            for (int i = 0; i < n; ++i)
            {
                data[i] = DecodeSegment(scan.Substring(i * nBitCoding, nBitCoding));
            }
            return data;
        }
        // for decoding 2, 3 or 4 bit encoded data
        private uint DecodeSegment(string codedData)
        {
            // encoding technique:
            // separate data into groups of 6 bits
            // 30H (48) added to convert to ASCII
            int bit, nBitEncoding = codedData.Length;
            uint val = 0;
            for (int i = 0; i < nBitEncoding; ++i)
            {
                bit = nBitEncoding - i - 1;
                val += (uint)Math.Pow(2, 6 * i) *
                    (Convert.ToUInt32(Convert.ToChar(codedData.Substring(bit, 1))) - 48);
            }
            return val;
        }
        #endregion

        #region command
        //make this a class with polymorphism

        public void StartLIDAR()
        {
            serialPort_sendCommand("BM\n", false);
        }

        public void StopLIDAR()
        {
            serialPort_sendCommand("QT\n", false);
        }

        public string GenerateCommand(int startingStep, int endStep, int clusterCount)
        {
            //lookup step angles for front, and max min detection range.
            return "GD" + PadString(startingStep.ToString(), "0", 4)
                        + PadString(endStep.ToString(), "0", 4)
                        + PadString(clusterCount.ToString(), "0", 2);
        }

        // TODO: Look at using PadLeft and PadRight functions
        private string PadString(string baseString, string padChar, int requiredLength)
        {
            string newString = baseString;
            while (newString.Length < requiredLength)
            {
                newString = padChar + newString;
            }
            return newString;
        }
        #endregion
    }
}
```

PathDecoding.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LynxSLAM
{
    //data structures to hold the path corrdinates and pose of the robot
    struct PathData
    {
        public int x;
        public int y;
        public PathData(int x, int y)
        {
            this.x = x;
            this.y = y;
        }
    }

    struct Pose
    {
        public int x;
        public int y;
        public double angle;
        public Pose(int x, int y, double angle)
        {
            this.x = x;
            this.y = y;
            this.angle = angle;
        }
    }

    public class PathDecoding
    {
        Pose currentPose = new Pose();

        PathData[] pathData = new PathData[0];
        int pathCount = 0;

        public bool stopPlannnig = false;
        //constructor
        public PathDecoding() { }

        #region Path_in
        //this function takes the data returning from the base PC and stores it in the coresponding veriables
to be acessed by other functions
        //and then returns a string containing the current pose of the robot
        public string DecodePath(string path)
        {
            int pathLength;
            bool newPath;

            Console.WriteLine(path);
            if (path != "error")
            {
                char[] delimeter = new char[1];
                delimeter[0] = ',';
                string[] components = path.Split(delimeter);

                if (components[0] != "nodata")
                {
                    currentPose.x = Convert.ToInt32(components[0]);
                    currentPose.y = Convert.ToInt32(components[1]);
                    currentPose.angle = (Convert.ToDouble(components[2]) / 1000) * (180 / Math.PI);

                    if (Convert.ToInt32(components[3]) > 0)
                    {
                        stopPlannnig = false;
                        pathCount = 0;
                        currentPose.x = Convert.ToInt32(components[0]);
                        currentPose.y = Convert.ToInt32(components[1]);
                        currentPose.angle = (Convert.ToDouble(components[2])/1000)*(180/Math.PI);
                        pathLength = Convert.ToInt32(components[3]);
                        pathData = new PathData[pathLength];
```

```
                        for (int i = 0; i < pathLength; i++)
                        {
                            pathData[i].x = Convert.ToInt32(components[(i * 2) + 4]);
                            pathData[i].y = Convert.ToInt32(components[(i * 2) + 5]);
                        }
                    }
                }
            }
            return "Robot at" + currentPose.x.ToString() + "," + currentPose.y.ToString() + " at " +
currentPose.angle.ToString() + " degrees";
        }
        #endregion
        #region Path_to_drive_instructions
        bool drivingForward = false;
        //this function uses the stored data about the path and robots pose and translates that its drive
instructions to be sent to the PIC
        public void PathDecode(RobotProtocol robot)
        {
            /// pose is the current belief pose of the robot from the mapping
            /// x_y[] is an array of x,y corrdinates of the calculated path which made in the Path_in region
            if (pathCount < pathData.Length)// size())
            {
                double adjustedAngle = currentPose.angle + 180;
                int requiredAngle;
                int adjacent = Math.Abs(currentPose.x - pathData[pathCount].x);
                if (adjacent == 0)
                {
                    adjacent += 1;
                }
                int opposite = Math.Abs(currentPose.y - pathData[pathCount].y);

                requiredAngle = Convert.ToInt16(Math.Atan(opposite / adjacent));

                //get quadrant.
                if (pathData[pathCount].x < currentPose.x)
                {
                    //left side
                    if (pathData[pathCount].y < currentPose.y)
                    {
                        //left top
                        requiredAngle = (int)(360 - requiredAngle);
                    }
                    else
                    {
                        //left bottom
                        Console.WriteLine("Test");
                    }
                }
                else
                {
                    //right side
                    if (pathData[pathCount].y < currentPose.y)
                    {
                        //right top
                        requiredAngle += 180;
                    }
                    else
                    {
                        //right bottom
                        requiredAngle = 180 - requiredAngle;
                    }
                }
                Console.WriteLine(requiredAngle + " " + adjustedAngle);

                short distance = Convert.ToInt16(Math.Sqrt(Math.Pow(adjacent, 2.0) + Math.Pow(opposite,
2.0)));
                int angleDifferance = (int)(requiredAngle - adjustedAngle);

                if (Math.Abs(angleDifferance) < Constants.deltaAngleThreshold)
                {
                    if ((Math.Abs(distance) < Constants.distanceThreshold))
                    {
                        RobotControl.Stop(robot);

                        if (drivingForward)
                        {
                            pathCount = pathCount + 1;
                            drivingForward = false;
```

```
                        }
                    }
                    else
                    {
                        short leftSpeed = Constants.driveForwardSpeed;
                        short rightSpeed = Constants.driveForwardSpeed;

                        if (angleDifferance < -3)
                        {
                            rightSpeed -= 5;

                        }
                        else if (angleDifferance > 3)
                        {
                            leftSpeed -= 5;
                        }
                        RobotControl.DriveLeftForward(robot, leftSpeed);
                        RobotControl.DriveRightForward(robot, rightSpeed);
                        drivingForward = true;
                    }
                }
                else
                {
                    if (angleDifferance > 0)
                    {
                        RobotControl.TurnLeft(robot, Constants.turnSpeed);
                    }
                    else
                    {
                        RobotControl.TurnRight(robot, Constants.turnSpeed);
                    }
                }
                Console.WriteLine(angleDifferance + "hiall");
            }
            else
            {
                pathData = new PathData[0];
            }
        }
        #endregion

        //This function was a test function to test the robot using the encoder count values to automatically
adjust its speed as it nears its destination when driving in a straight line
         public void DriveDistance(RobotProtocol protocol, int encoder, short maxSpeed)
         {
            int initial = protocol.leftEncoderCount;
            int current = initial;
            int goal = initial + encoder;
            short scaleSpeed;
            float scale = 1;
            int scaleDistance = goal - 5000;
            do
            {
                List<byte> Pro_cmd = new List<byte>();
                Pro_cmd.Add(ProtocolConstants.encoderLeftRear);
                protocol.Protocol(Pro_cmd);
                Pro_cmd.Clear();
                current = protocol.leftEncoderCount;
                if (current > scaleDistance)
                {
                    scale = 1 - ((float)(current - scaleDistance)/5000);
                    scaleSpeed = (short)(maxSpeed * scale);
                    if (scaleSpeed < 18)
                    {
                        scaleSpeed = 18;
                    }
                }
                else
                {
                    scaleSpeed = maxSpeed;
                }
                RobotControl.DriveForward(protocol, scaleSpeed);
                Console.WriteLine(scaleSpeed);
            } while (protocol.leftEncoderCount < initial + encoder);
            RobotControl.Stop(protocol);
        }
    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace LynxSLAM
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

RobotControl.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LynxSLAM
{
    //this class contains high level functions that send the corresponding drive
commands to the PIC with the specified speeds, angles and distances
    static class RobotControl
    {
        public static bool eStopActive = false;
        public static void PointTurnLeft(RobotProtocol lynxRobot, short s, int angle)
        {
            if (!eStopActive)
            {
                List<byte> cmd = new List<byte>();
                byte angle_low = (byte)(angle & 0x00FF);
                byte angle_high = (byte)((angle & 0xFF00) >> 8);
                byte speed = (byte)s;
                cmd.Add(ProtocolCommands.pointTurnLeft);
                cmd.Add(speed);
                cmd.Add(angle_high);
                cmd.Add(angle_low);
                lynxRobot.Protocol(cmd);
                lynxRobot.driveCmdComplete = false;
            }
        }

        public static void PointTurnRight(RobotProtocol lynxRobot, short s, int angle)
        {
            if (!eStopActive)
```

```
            {
                 List<byte> cmd = new List<byte>();
                byte angle_low = (byte)(angle & 0x00FF);
                byte angle_high = (byte)((angle & 0xFF00) >> 8);
                byte speed = (byte)s;
                cmd.Add(ProtocolCommands.pointTurnRight);
                cmd.Add(speed);
                cmd.Add(angle_high);
                cmd.Add(angle_low);
                lynxRobot.Protocol(cmd);
                lynxRobot.driveCmdComplete = false;
            }
        }

        public static void TurnLeft(RobotProtocol lynxRobot, short s)
        {
            if (!eStopActive)
            {
                List<byte> cmd = new List<byte>();
                byte speed = (byte)s;
                cmd.Add(ProtocolCommands.turnLeft);
                cmd.Add(speed);
                cmd.Add(0);
                cmd.Add(0);
                lynxRobot.Protocol(cmd);
                lynxRobot.driveCmdComplete = false;
            }
        }

        public static void TurnRight(RobotProtocol lynxRobot, short s)
        {
            if (!eStopActive)
            {
                List<byte> cmd = new List<byte>();
                byte speed = (byte)s;
                cmd.Add(ProtocolCommands.turnRight);
                cmd.Add(speed);
                cmd.Add(0);
                cmd.Add(0);
                lynxRobot.Protocol(cmd);
                lynxRobot.driveCmdComplete = false;
            }
        }

        public static void Stop(RobotProtocol lynxRobot)
        {
            List<byte> cmd = new List<byte>();
            cmd.Add(ProtocolCommands.stop);
            cmd.Add(0);
            cmd.Add(0);
            cmd.Add(0);
            lynxRobot.Protocol(cmd);
            lynxRobot.driveCmdComplete = false;
        }

        public static void DriveForward(RobotProtocol lynxRobot, short s)
        {
            if (!eStopActive)
            {
                List<byte> cmd = new List<byte>();
```

```
                byte speed = (byte)s;
                cmd.Add(ProtocolCommands.driveForward);
                cmd.Add(speed);
                cmd.Add(0);
                cmd.Add(0);
                lynxRobot.Protocol(cmd);
                lynxRobot.driveCmdComplete = false;
            }

        }

        public static void DriveReverse(RobotProtocol lynxRobot, short s)
        {
            if (!eStopActive)
            {
                List<byte> cmd = new List<byte>();
                byte speed = (byte)s;
                cmd.Add(ProtocolCommands.driveReverse);
                cmd.Add(speed);
                cmd.Add(0);
                cmd.Add(0);
                lynxRobot.Protocol(cmd);
                lynxRobot.driveCmdComplete = false;
            }
        }

        public static void DriveForwardEnocder(RobotProtocol lynxRobot, short s, short
distance)
        {
            if (!eStopActive)
            {
                List<byte> cmd = new List<byte>();
                byte distance_low = (byte)(distance & 0x00FF);
                byte distance_high = (byte)((distance & 0xFF00) >> 8);
                byte speed = (byte)s;
                cmd.Add(ProtocolCommands.driveForwardEncoder);
                cmd.Add(speed);
                cmd.Add(distance_high);
                cmd.Add(distance_low);
                lynxRobot.Protocol(cmd);
                lynxRobot.driveCmdComplete = false;
            }
        }

        public static void DriveReverseEnocder(RobotProtocol lynxRobot, short s, short
distance)
        {
            if (!eStopActive)
            {
               List<byte> cmd = new List<byte>();
                byte distance_low = (byte)(distance & 0x00FF);
                byte distance_high = (byte)((distance & 0xFF00) >> 8);
                byte speed = (byte)s;
                cmd.Add(ProtocolCommands.driveReverseEncoder);
                cmd.Add(speed);
                cmd.Add(distance_high);
                cmd.Add(distance_low);
                lynxRobot.Protocol(cmd);
                lynxRobot.driveCmdComplete = false;
            }
        }
```

```
        public static void DriveLeftForward(RobotProtocol lynxRobot, short s)
        {
            if (!eStopActive)
            {
                List<byte> cmd = new List<byte>();
                byte speed = (byte)s;
                cmd.Add(ProtocolCommands.driveLeftForward);
                cmd.Add(speed);
                cmd.Add(0);
                cmd.Add(0);
                lynxRobot.Protocol(cmd);
                lynxRobot.driveCmdComplete = false;
            }
        }

        public static void DriveRightForward(RobotProtocol lynxRobot, short s)
        {
            if (!eStopActive)
            {
                List<byte> cmd = new List<byte>();
                byte speed = (byte)s;
                cmd.Add(ProtocolCommands.driveRightForward);
                cmd.Add(speed);
                cmd.Add(0);
                cmd.Add(0);
                lynxRobot.Protocol(cmd);
                lynxRobot.driveCmdComplete = false;
            }
        }

        public static void DriveLeftReverse(RobotProtocol lynxRobot, short s)
        {
            if (!eStopActive)
            {
                List<byte> cmd = new List<byte>();
                byte speed = (byte)s;
                cmd.Add(ProtocolCommands.driveLeftReverse);
                cmd.Add(speed);
                cmd.Add(0);
                cmd.Add(0);
                lynxRobot.Protocol(cmd);
                lynxRobot.driveCmdComplete = false;
            }
        }

        public static void DriveRightReverse(RobotProtocol lynxRobot, short s)
        {
            if (!eStopActive)
            {
                List<byte> cmd = new List<byte>();
                byte speed = (byte)s;
                cmd.Add(ProtocolCommands.driveRightReverse);
                cmd.Add(speed);
                cmd.Add(0);
                cmd.Add(0);
                lynxRobot.Protocol(cmd);
                lynxRobot.driveCmdComplete = false;
            }
        }
    }
}
```

RobotProtocol.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.Ports;
using System.Timers;

namespace LynxSLAM
{
    //this class contains the robot drive command values
    public static class ProtocolCommands
    {
        public const byte stop = 0x01;
        public const byte driveForward = 0x02;
        public const byte driveForwardEncoder = 0x03;
        public const byte driveReverse = 0x04;
        public const byte driveReverseEncoder = 0x05;
        public const byte pointTurnLeft = 0x06;
        public const byte pointTurnRight = 0x07;
        public const byte driveLeftForward = 0x08;
        public const byte driveRightForward = 0x09;
        public const byte driveLeftReverse = 0x0A;
        public const byte driveRightReverse = 0x0B;
        public const byte turnLeft = 0x0C;
        public const byte turnRight = 0x0D;
    }

    //this class contains the robot read command values
    public static class ProtocolConstants
    {
        public const byte encoderLeftRear = 0x83;
        public const byte encoderRightRear = 0x84;
        public const byte infraRedFront = 0x85;
        public const byte infraRedFrontLeft = 0x86;
        public const byte infraRedFrontRight = 0x87;
        public const byte infraRedRear = 0x88;
        public const byte infraRedRearLeft = 0x89;
        public const byte infraRedRearRight = 0x8A;
    }

    public class RobotProtocol : SerialConnection
    {
        static bool _connected = false;
        public int leftEncoderCount, leftEncoderOverflow, rightEncoderCount, rightEncoderOverflow,
IR_distance;
        public bool IR_unsafe, IR_F_unsafe, IR_FL_unsafe, IR_FR_unsafe, IR_R_unsafe, IR_RL_unsafe,
IR_RR_unsafe, driveCmdComplete, breakWaiting, turning;
        static private bool waiting, resend;

        private Timer waitLoop;

        public RobotProtocol()
        {
            InitComms("COM3", 115200);
            IR_unsafe = false;
            IR_F_unsafe = false;
            IR_FL_unsafe = false;
            IR_FR_unsafe = false;
            IR_R_unsafe = false;
            IR_RL_unsafe = false;
            IR_RR_unsafe = false;
            driveCmdComplete = true;
            breakWaiting = false;
            turning = false;
        }

        private void InitialiseRobotTimer()
        {
            waitLoop = new Timer(100); // inialises with an interval of 100 ms
           // waitLoop.Interval = ;
            waitLoop.Elapsed += new ElapsedEventHandler(OnTimedEvent);
           // waitLoop.Tick += new EventHandler(programLoop_Tick);
        }

        private static void OnTimedEvent(object source, ElapsedEventArgs e)
```

```
        {
            waiting = false;
        }

        public override void serialPort_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {
            SerialPort sp = (SerialPort)sender;

            byte checksum = 0;
            int count;
            int byte_0 = 0;

            byte_0 = sp.ReadByte();

            if ((131 == byte_0) || (byte_0 == 132))
            {
                count = 5;

                while (sp.BytesToRead < 5) { }
                byte[] bytes_received = new byte[count + 1];
                bytes_received[0] = (byte)byte_0;
                for (int i = 1; i < count + 1; i++)
                {
                    bytes_received[i] = (byte)sp.ReadByte();
                }

                for (int i = 0; i < count + 1; i++)
                {
                    checksum = (byte)(bytes_received[i] ^ checksum);
                }

                // receives and stores the encoder values forthe left and right encoders when the
coresponding command in sent to the PIC
                if (checksum == 0)
                {
                    if (byte_0 == 132)
                    {
                        rightEncoderCount = ((int)bytes_received[1] << 8) | (int)bytes_received[2];
                        rightEncoderOverflow = ((int)bytes_received[3] << 8) | (int)bytes_received[4];

                        resend = false;
                        waiting = false;
                        return;
                    }
                    else
                    {
                        leftEncoderCount = ((int)bytes_received[1] << 8) | (int)bytes_received[2];
                        leftEncoderOverflow = ((int)bytes_received[3] << 8) | (int)bytes_received[4];

                        resend = false;
                        waiting = false;
                        return;
                    }
                }
                else
                {
                    resend = true;
                    waiting = true;

                    return;
                }
            }
            //receives and stores information about the IR distance reading when requested from the FitPC
            else if ((133 <= byte_0) & (byte_0 <= 138))
            {
                count = 2;

                while (sp.BytesToRead < 2) { }
                byte[] bytes_received = new byte[count + 1];
                bytes_received[0] = (byte)byte_0;
                for (int i = 1; i < count + 1; i++)
                {
                    bytes_received[i] = (byte)sp.ReadByte();
                }

                for (int i = 0; i < count + 1; i++)
                {
                    checksum = (byte)(bytes_received[i] ^ checksum);
```

```
            }

            if (checksum == 0)
            {
                IR_distance = ((int)bytes_received[1] << 8) | (int)bytes_received[2];

                resend = false;
                waiting = false;
                return;
            }
            else
            {
                resend = true;
                waiting = true;
                return;
            }
        }
        else
        {
            switch (byte_0)
            {
                case 1:         //checksum passed for write/drive commades
                    resend = false;
                    waiting = false;
                    return;

                case 2:
                    resend = true;
                    waiting = false;
                    return;

                case 3:             //last drive comand sent to the PIC has completed, send next one
                    resend = false;
                    waiting = false;
                    driveCmdComplete = true;
                    return;

                case 4:            //receive_error on the PIC
                    resend = true;
                    waiting = true;
                    return;

                case 7:             //robot is turning
                    turning = true;
                    return;

                case 8:             //robot is not turning
                    turning = false;
                    return;

                case 249:           //nothing is too close to the robot
                    IR_unsafe = false;
                    IR_unsafe = false;
                    IR_F_unsafe = false;
                    IR_FL_unsafe = false;
                    IR_FR_unsafe = false;
                    IR_R_unsafe = false;
                    IR_RL_unsafe = false;
                    IR_RR_unsafe = false;

                    return;

                case 250:           //the rear right IR sensor detects something too close to the robot
                    IR_RR_unsafe = true;
                    IR_unsafe = true;

                    return;

                case 251:           //the rear left IR sensor detects something too close to the robot
                    IR_RL_unsafe = true;
                    IR_unsafe = true;

                    return;

                case 252:           //the rear IR sensor detects something too close to the robot
                    IR_R_unsafe = true;
                    IR_unsafe = true;
```

```
                        return;

                    case 253:           //the front right IR sensor detects something too close to the robot
                        IR_FR_unsafe = true;
                        IR_unsafe = true;

                        return;

                    case 254:           //the front left IR sensor detects something too close to the robot
                        IR_FL_unsafe = true;
                        IR_unsafe = true;

                        return;

                    case 255:           //the front IR sensor detects something too close to the robot
                        IR_F_unsafe = true;
                        IR_unsafe = true;

                        return;

                    default:
                        resend = true;
                        waiting = true;

                        return;
                }
            }
        }

        //this function sends data to the PIC to be processed, either drive commands or request for sensor
data, and has a timeout loop in the PIC does not respond in time
        public void serialPort_sendBytes(byte[] data, int timeOut = 0)
        {
            waiting = true;
            int counter = 1;
            for (int i = 0; i < data.Length; i++)
            {
                serialPort_sendByte(data[i]);
            }
            // waits for a responce from the PIC
            while (waiting)
            {
                counter++;
                if (counter == timeOut) //set 0 for no timeout
                {
                    breakWaiting = true;
                    break;
                }
            }
        }

        //calculates the checksum and sends the commands to the PIC
        public void Protocol(List<byte> cmd)
        {
            if(true)
            {
                List<byte> cmd_line = new List<byte>();
                byte checksum = 0;
                resend = false;
                cmd_line = cmd;

                for (int i = 0; i < cmd_line.Count; i++)
                {
                    checksum = (byte)(cmd_line[i] ^ checksum);
                }
                cmd_line.Add(checksum);
                byte[] cmd_array = cmd_line.ToArray();

                // sends cmd_array to the pic with a timeout of 60000 counts and if the PIC sends back that
the checksum did pass then cmd_array is resent
                do
                {
                    serialPort_sendBytes(cmd_array, 60000);
                } while (resend);
            }
        }
    }
}
```

SerialConnection.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.Ports;

namespace LynxSLAM
{
    public class SerialConnection
    {
        private SerialPort serialPort;
        private bool polling;
        public List<string> lastData;

        #region Initialisation

        public bool InitComms(string port, int baudRate)
        {
            bool success = false;
            serialPort = new SerialPort(port);
            serialPort.BaudRate = baudRate;
            serialPort.DataReceived += new SerialDataReceivedEventHandler(serialPort_DataReceived);
            serialPort.DataBits = 8;
            serialPort.Parity = Parity.None;
            serialPort.StopBits = StopBits.One;

            try
            {
                serialPort.Open();
                success = true;
            }
            catch (Exception e)
            {
            }
            return success;
        }
        #endregion

        #region Status

        public string serialPort_GetName()
        {
            string port = "";
            try
            {
                port = serialPort.PortName;
            }
            catch (Exception e)
            {
                port = e.GetType().ToString();
            }
            return port;
        }
        public int serialPort_GetBaud()
        {
            int baud;
            try
            {
                baud = serialPort.BaudRate;
            }
            catch (Exception e)
            {
                baud = 0;
            }
            return baud;
        }
        public bool serialPort_IsOpen()
        {
            bool open;
            try
            {
                open = serialPort.IsOpen;
            }
            catch (Exception)
```

```
            {
                open = false;
            }
            return open;
        }

        public bool serialPort_IsPolling()
        {
            if (polling)
                return true;
            else
                return false;
        }

        #endregion

        #region Send Data
        public void serialPort_sendByte(uint send)
        {
            byte[] data = new byte[1];
            data[0] = (byte)send;
            try
            {
                serialPort.Write(data, 0, 1);
            }
            catch (Exception) { }
        }

        public void serialPort_sendByte(byte send)
        {
            if (serialPort_IsOpen())
            {
                byte[] data = new byte[1];
                data[0] = (byte)send;
                try
                {
                    serialPort.Write(data, 0, 1);
                }
                catch (Exception) { }
            }
        }

        public void serialPort_sendString(string s)
        {
            try
            {
                serialPort.WriteLine(s);
            }
            catch (Exception) { }
        }

        #endregion

        #region Send Data and Poll for Response

        public virtual List<string> serialPort_SendReceive(string toSend)
        {
            polling = true;
            serialPort.WriteLine(toSend);
            while (polling) { } // data received will set polling to false after updating lastData arraylist
i.v.
            return lastData;
        }
        public virtual List<string> serialPort_SendReceive(uint toSend, int timeOut = 0)
        {
            int counter = 1;
            if (serialPort_IsOpen())
            {
                polling = true;
                byte[] data = new byte[1];
                data[0] = (byte)toSend;
                serialPort.Write(data, 0, 1);
                while (polling) // data received will set polling to false after updating lastData arraylist
i.v.
                {
                    counter++;
                    if (counter == timeOut) //set 0 for no timeout
                        break;
```

```
                }
                return lastData;
            }
            else
            {
                //TODO: Create error to return
                List<string> error = new List<string>();
                return error;
            }
        }
        #endregion

        #region Receive Data
        public virtual void serialPort_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {
            SerialPort sp = (SerialPort)sender;
            try
            {
                List<string> lines = new List<string>();
                while (sp.BytesToRead > 0)                  // while data is in the RX buffer
                {
                    lines.Add(sp.ReadTo("\n"));             // get one line at a time
                }
                lastData = lines;
                polling = false;
            }
            catch (Exception) { }
        }
        #endregion
    }
}
```

TCPClient.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Net;
using System.Net.Sockets;

namespace LynxSLAM
{
    class TCP
    {
        private NetworkStream stream;
        private TcpClient client;

        public TCP()
        {

        }

        public bool IsConnected()
        {
            try
            {
                return client.Connected;
            }
            catch (Exception)
            {
                return false;
            }
        }

        public void Connect(String server)
        {
            try
            {
                // Create a TcpClient.
                // Note, for this client to work you need to have a TcpServer
                // connected to the same address as specified by the server, port
                // combination.
                Int32 port = 13000;
```

```
                client = new TcpClient(server, port);

                // Get a client stream for reading and writing.
                //  Stream stream = client.GetStream();
                stream = client.GetStream();
            }
            catch (ArgumentNullException e)
            {
                Console.WriteLine("ArgumentNullException: {0}", e);
            }
            catch (SocketException e)
            {
                Console.WriteLine("SocketException: {0}", e);
            }
        }


        public string SendData(string message)
        {
            string toReturn = "";
            try
            {
                // Translate the passed message into ASCII and store it as a Byte array.
                Byte[] data = System.Text.Encoding.ASCII.GetBytes(message);

                // Send the message to the connected TcpServer.
                stream.Write(data, 0, data.Length);

                ////Console.WriteLine("Sent: {0}", message);

                // Receive the TcpServer.response.

                // Buffer to store the response bytes.
                data = new Byte[102400];//TODO: CHECK LENGTH

                // String to store the response ASCII representation.
                String responseData = String.Empty;

                // Read the first batch of the TcpServer response bytes.
                Int32 bytes = stream.Read(data, 0, data.Length);
                responseData = System.Text.Encoding.ASCII.GetString(data, 0, bytes);
                toReturn = responseData;
            }
            catch (Exception)
            {
                toReturn = "error";
            }
            return toReturn;
        }
    }
}
```

UDPControl.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Net;
using System.Net.Sockets;

using System.Windows.Forms;

// So that listener doesn't keep UI from updating
using System.Threading;

namespace LynxSLAM
{
    public class UDPControl
    {
        public int leftSpeed = 0;
        public int rightSpeed = 0;

        public int noUDP = 0;
```

```
        public UDPControl()
        {
        }

        public void InitUDP()
        {
            ThreadStart listen = new ThreadStart(StartListener);
            Thread thread = new Thread(listen);
            thread.Start();
        }

        private void StartListener()
        {
            bool done = false;
            string[] speeds;
            int listenPort = Convert.ToInt16("15000");
            UdpClient listener = new UdpClient(listenPort);
            IPEndPoint groupEP = new IPEndPoint(IPAddress.Any, listenPort);

            try
            {
                while (!done)
                {
                    byte[] bytes = listener.Receive(ref groupEP);

                    string received = Encoding.ASCII.GetString(bytes, 0, bytes.Length);
                    noUDP = 0;

                    // Format will be: left speed, right speed
                    char[] delimeter = new char[1];
                    delimeter[0] = ',';
                    speeds = received.Split(delimeter);

                    leftSpeed = Convert.ToInt32(speeds[0]);
                    rightSpeed = Convert.ToInt32(speeds[1]);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.ToString());
            }
            finally
            {
                listener.Close();
            }
        }
    }
}
```

Appendix C – dsPIC Code

ASCII_Protocol.c

```
#include <p33FJ128MC804.h>
#include <libpic30.h>
#include <stdio.h>
#include <stdlib.h>
#include "functions.h"

#if 0
// Set the configuration bits
_FBS( RBS_NO_RAM &  BSS_NO_BOOT_CODE & BWRP_WRPROTECT_OFF ); // No Boot Ram, No Boot Segment Program Memory,
Write protect disabled
_FSS( RSS_NO_RAM & SSS_NO_FLASH & SWRP_WRPROTECT_OFF );                        // No Secure Ram, No Secure
Segment, Write protect disabled
_FGS(GSS_OFF & GCP_OFF & GWRP_OFF );                                                                    // Code
protect off,code protect disabled, Write protect disabled
_FOSCSEL( FNOSC_FRC & IESO_OFF  );                                                                      // Start up
with Fast RC we switch to PRIPLL in code later, Two speed start up disabled
_FOSC( FCKSM_CSECMD  & IOL1WAY_OFF & OSCIOFNC_ON & POSCMD_NONE  );    // Only clock switching enabled, Single
configuration for remappable I/O IOL1WAY_OFF Disabled, OSC2 Pin function :OSC2 is clock O/P, Oscillator
Selection:Primary disabled
_FWDT(FWDTEN_OFF & WINDIS_OFF);                                                                                 //
Disable watch dog timer
_FPOR(ALTI2C_OFF & FPWRT_PWR1 );                                                                        // I2C mapped
to SDA1/SCL1,Power on reset disabled
_FICD(JTAGEN_OFF & ICS_PGD3 );
#endif

#define byte        unsigned char

#define sIR               0.4     // IR min distance
#define prox_error_F      255
#define prox_error_FL     254
#define prox_error_FR     253
#define prox_error_R      252
#define prox_error_RL     251
#define prox_error_RR     250
#define prox_safe         249
#define chksum_pass       1
#define chksum_error      2
#define drive_cmd_fin     3
#define receive_error     4
#define processing_cmd    5
#define cmd_finished      6
#define rotating          7
#define not_rotating      8

int IR_error_code;

//encoder veriables
unsigned int en_r_l, en_r_r, en_f_l, en_f_r;
int      overflow_r_l, overflow_r_r, overflow_f_l, overflow_f_r, turning, encoder_reset, driving;

//////////modidyed libary fucntions
#define ENCODER_COUNTS_PER_DEGREE 18;
#define COUNTS_PER_MM 4.22;

void point_turn_left_1(unsigned char speed, unsigned int angle)
{
    int encoder_counts = angle * ENCODER_COUNTS_PER_DEGREE;
    spi_command(MTR_LEFT, WRITE_REVERSE_ENC_CNT, speed, encoder_counts);
    spi_command(MTR_RIGHT, WRITE_FORWARD_ENC_CNT, speed, encoder_counts);
}

void point_turn_right_1(unsigned char speed, unsigned int angle)
{
    int encoder_counts = angle * ENCODER_COUNTS_PER_DEGREE;
    spi_command(MTR_RIGHT, WRITE_REVERSE_ENC_CNT, speed, encoder_counts);
    spi_command(MTR_LEFT, WRITE_FORWARD_ENC_CNT, speed, encoder_counts);
}
```

```
void turn_left_1(unsigned char speed)
{
    spi_command(MTR_LEFT, WRITE_REVERSE_SPEED, speed, 0);
    spi_command(MTR_RIGHT, WRITE_FORWARD_SPEED, speed, 0);
}

void turn_right_1(unsigned char speed)
{
    spi_command(MTR_RIGHT, WRITE_REVERSE_SPEED, speed, 0);
    spi_command(MTR_LEFT, WRITE_FORWARD_SPEED, speed, 0);
}


void drive_forward_encoder_1(unsigned char speed, unsigned int distance)
{
    unsigned int encoder_counts = distance * COUNTS_PER_MM;
    spi_command(MTR_LEFT,  WRITE_FORWARD_ENC_CNT, speed, encoder_counts);
    spi_command(MTR_RIGHT, WRITE_FORWARD_ENC_CNT, speed, encoder_counts);
}
//////////////////////

//this function retrieves and returns one byte from the received register
byte getSerial(void)
{
        U2RXREG = 0;
        while(!U2STAbits.URXDA){}; // Wait for character
        return U2RXREG;
}

//this function sneds one byte to the transmit register
void sendSerial(byte send)
{
        while(U2STAbits.UTXBF){};
         U2TXREG = send;
}

//this function sets the veriable IR_error_code where any value > 0 means atleast one IR reading is within
the minimum IR reading and therefore the robot is unsafe
void check_IR()
{
        float pF,pFL,pRL,pFR,pRR,pR;

        pF=prox_get_distance(PROX_FRONT);
        pFL=prox_get_distance(PROX_FRONT_LEFT);
        pFR=prox_get_distance(PROX_FRONT_RIGHT);
        pR=prox_get_distance(PROX_REAR);
        pRL=prox_get_distance(PROX_REAR_LEFT);
        pRR=prox_get_distance(PROX_REAR_RIGHT);

        if (pF<sIR)
        {
                sendSerial(prox_error_F);
                IR_error_code = 1;
                return;
        }
        if (pFL>sIR)
        {
                sendSerial(prox_error_FL);
                IR_error_code = 2;
                return;
        }
        if (pFR>sIR)
        {
                sendSerial(prox_error_FR);
                IR_error_code = 3;
                return;
        }
        if (pR>sIR)
        {
                sendSerial(prox_error_R);
                IR_error_code = 4;
                return;
        }
        if (pRL>sIR)
        {
                sendSerial(prox_error_RL);
                IR_error_code = 5;
                return;
```

```
        }
        if (pRR>sIR)
        {
                sendSerial(prox_error_RR);
                IR_error_code = 6;
                return;
        }
        if (IR_error_code != 0)
        {
                sendSerial(prox_safe);
                IR_error_code = 0;
        }
        return;
}

void protocol(void)
{
        byte cmd;
        byte checksum;
        int distance;
        byte distance_high;
        byte distance_low;
        int angle;
        byte en_low, en_high;
        byte overflow_low, overflow_high;
        int i;
        checksum = 0;

        //retrieves the initial command byte sent from the FitPC
        cmd = getSerial();

        if((cmd & 0x40)==0x40)    // bit 6 of cmd byte - if 1, byte 2 is number of bytes being sent (aka
veriable length)
                                                                // if 0, number of bytes being sent is 6
        {
                //left here as infrastructure if someone wants to implement receiving a variable length
command if needed

        }
        else
        {
                byte recieved_bytes[5];
                recieved_bytes[0] = cmd;
                i = 0;

                cmd = recieved_bytes[0] & 0x7F;
                if ((recieved_bytes[0] & 0x80)==0x00)//Write command
                {
                        for(i = 1; i < 5; i++)
                        {
                                recieved_bytes[i] = getSerial();
                        }
                        i = 0;
                        for(i=0;i<5;i++)
                        {
                                checksum = recieved_bytes[i] ^ checksum;
                        }

                        if (checksum == 0)
                        {
                                sendSerial(chksum_pass);
                                //drive commands that have been sent down from the FitPC
                                switch (cmd)
                                {
                                        case 0x01:       //sends stop command to the motors from the
FitPC
                                                drive_forward(0);
                                                turning = 0;
                                                if (driving == 0)
                                                {
                                                        encoder_reset = 0;
                                                }
                                                driving = 1;
                                                break;

                                        case 0x02:       //sends forward command to the motors from the
FitPC
                                                drive_forward(recieved_bytes[1]);
```

```
                                        turning = 0;
                                        if (driving == 0)
                                        {
                                                encoder_reset = 0;
                                        }
                                        driving = 1;
                                        break;

                                case 0x03:      //sends forward encoder command to the motors
from the FitPC
                                        distance = ((int)recieved_bytes[2] << 8) |
(int)recieved_bytes[3];
                                        drive_forward_encoder_1(recieved_bytes[1],distance);
                                        turning = 0;
                                        if (driving == 0)
                                        {
                                                encoder_reset = 0;
                                        }
                                        driving = 1;
                                        break;

                                case 0x04:      //sends reverse command to the motors from the
FitPC
                                        drive_reverse(recieved_bytes[1]);

                                        turning = 0;
                                        if (driving == 0)
                                        {
                                                encoder_reset = 0;
                                        }
                                        driving = 1;
                                        break;

                                case 0x05:      //sends reverse encoder command to the motors
from the FitPC
                                        distance = ((int)recieved_bytes[2] << 8) |
(int)recieved_bytes[3];
                                        drive_reverse_encoder(recieved_bytes[1],distance);
                                        turning = 0;
                                        if (driving == 0)
                                        {
                                                encoder_reset = 0;
                                        }
                                        driving = 1;
                                        break;

                                case 0x06:      //sends point turn left command to the motors
from the FitPC
                                        angle = ((int)recieved_bytes[2] << 8) |
(int)recieved_bytes[3];
                                        point_turn_left_1(recieved_bytes[1],angle);
                                        turning = 1;
                                        encoder_reset = 1;
                                        driving = 0;

                                        sendSerial(rotating);
                                        break;

                                case 0x07:      //sends point turn right command to the motors
from the FitPC
                                        angle = ((int)recieved_bytes[2] << 8) |
(int)recieved_bytes[3];
                                        point_turn_right_1(recieved_bytes[1],angle);
                                        turning = 1;
                                        encoder_reset = 1;
                                        driving = 0;

                                        sendSerial(rotating);
                                        break;

                                case 0x08:      //sends left forward command to the motors from
the FitPC
                                        drive_leftmotor_forward(recieved_bytes[1]);

                                        break;

                                case 0x09:      //sends right forward command to the motors from
```

```
the FitPC
                                                drive_rightmotor_forward(recieved_bytes[1]);

                                                break;

                                        case 0x0A:      //sends left reverse command to the motors from
the FitPC
                                                drive_leftmotor_reverse(recieved_bytes[1]);

                                                break;

                                        case 0x0B:      //sends right reverse command to the motors from
the FitPC
                                                drive_rightmotor_reverse(recieved_bytes[1]);

                                                break;

                                        case 0x0C:      //sends turn left command to the motors from the
FitPC
                                                turn_left_1(recieved_bytes[1]);
                                                turning = 1;
                                                encoder_reset = 1;
                                                driving = 0;

                                                sendSerial(rotating);
                                                break;

                                        case 0x0D:      //sends turn right command to the motors from
the FitPC
                                                turn_right_1(recieved_bytes[1]);
                                                turning = 1;
                                                encoder_reset = 1;
                                                driving = 0;

                                                sendSerial(rotating);
                                                break;
                                        default:
                                                break;
                                }
                        }
                        else
                        {
                                sendSerial(chksum_error);
                        }
                }
                else    //Read commands
                {
                        //finishes retrieving the data sent from the FitPC and calculates the checksum
                        for(i = 1; i < 2; i++)
                        {
                                recieved_bytes[i] = getSerial();
                        }
                        i = 0;
                        for(i=0;i<2;i++)
                        {
                                checksum = recieved_bytes[i] ^ checksum;
                        }
                        if (checksum == 0)
                        {
                                switch (cmd)
                                {
                                        case 0x03:      //sends the Left Encoder values to the FitPC

                                                en_low = (byte)(en_r_l & 0x00FF);
                                                en_high = (byte)((en_r_l & 0xFF00) >> 8);
                                                overflow_low = (byte)(overflow_r_l & 0x00FF);
                                                overflow_high = (byte)((overflow_r_l & 0xFF00) >> 8);

                                                checksum = recieved_bytes[0] ^ en_low ^ en_high ^
overflow_low ^ overflow_high;

                                                sendSerial(recieved_bytes[0]);
                                                sendSerial(en_high);
                                                sendSerial(en_low);
                                                sendSerial(overflow_high);
                                                sendSerial(overflow_low);
                                                sendSerial(checksum);
                                                break;
```

```
                                    case 0x04:        //sends the Right Encoder values to the FitPC

                                            en_low = (byte)(en_r_r & 0x00FF);
                                            en_high = (byte)((en_r_r & 0xFF00) >> 8);
                                            overflow_low = (byte)(overflow_r_r & 0x00FF);
                                            overflow_high = (byte)((overflow_r_r & 0xFF00) >> 8);

                                            checksum = recieved_bytes[0] ^ en_low ^ en_high ^
overflow_low ^ overflow_high;

                                            sendSerial(recieved_bytes[0]);
                                            sendSerial(en_high);
                                            sendSerial(en_low);
                                            sendSerial(overflow_high);
                                            sendSerial(overflow_low);
                                            sendSerial(checksum);
                                            break;

                                    case 0x05:       //sends the IR Front value to the FitPC
                                            distance_low = (byte)(distance & 0x00FF);
                                            distance_high = (byte)((distance & 0xFF00) >> 8);

                                            checksum = recieved_bytes[0] ^ distance_low ^
distance_high;

                                            sendSerial(recieved_bytes[0]);
                                            sendSerial(distance_high);
                                            sendSerial(distance_low);
                                            sendSerial(checksum);
                                            break;

                                    case 0x06:       //sends the IR Front Left value to the FitPC
                                            distance_low = (byte)(distance & 0x00FF);
                                            distance_high = (byte)((distance & 0xFF00) >> 8);

                                            checksum = recieved_bytes[0] ^ distance_low ^
distance_high;

                                            sendSerial(recieved_bytes[0]);
                                            sendSerial(distance_high);
                                            sendSerial(distance_low);
                                            sendSerial(checksum);
                                            break;

                                    case 0x07:       //sends the IR Front Right value to the FitPC
                                            distance_low = (byte)(distance & 0x00FF);
                                            distance_high = (byte)((distance & 0xFF00) >> 8);

                                            checksum = recieved_bytes[0] ^ distance_low ^
distance_high;

                                            sendSerial(recieved_bytes[0]);
                                            sendSerial(distance_high);
                                            sendSerial(distance_low);
                                            sendSerial(checksum);
                                            break;

                                    case 0x08:       //sends the IR Rear value to the FitPC
                                            distance_low = (byte)(distance & 0x00FF);
                                            distance_high = (byte)((distance & 0xFF00) >> 8);

                                            checksum = recieved_bytes[0] ^ distance_low ^
distance_high;

                                            sendSerial(recieved_bytes[0]);
                                            sendSerial(distance_high);
                                            sendSerial(distance_low);
                                            sendSerial(checksum);
                                            break;

                                    case 0x09:       //sends the IR Rear Left value to the FitPC
                                            distance_low = (byte)(distance & 0x00FF);
                                            distance_high = (byte)((distance & 0xFF00) >> 8);

                                            checksum = recieved_bytes[0] ^ distance_low ^
distance_high;
```

```
                                                sendSerial(recieved_bytes[0]);
                                                sendSerial(distance_high);
                                                sendSerial(distance_low);
                                                sendSerial(checksum);
                                                break;

                                        case 0x0A:       //sends the IR Rear Right value to the FitPC
                                                distance_low = (byte)(distance & 0x00FF);
                                                distance_high = (byte)((distance & 0xFF00) >> 8);
                                                checksum = recieved_bytes[0] ^ distance_low ^
distance_high;

                                                sendSerial(recieved_bytes[0]);
                                                sendSerial(distance_high);
                                                sendSerial(distance_low);
                                                sendSerial(checksum);
                                                break;
                                }
                        }
                        else
                        {
                                sendSerial(chksum_error);
                        }
                }
        }
}

//This function retrieves the encoder count value and keeps track of any overflows and stores them in
variables which can be read whenever the FitPC requests them
void get_encoder_data()
{

        int left_motor_status = motor_get_status (MTR_LEFT,0);
        int right_motor_status = motor_get_status (MTR_RIGHT,0);

        en_r_l = encoder_rear(MTR_LEFT);
        if (left_motor_status & ERROR_ENCODER_OVERFLOW_REAR)
        {
                motor_reset_status(MTR_LEFT, ERROR_ENCODER_OVERFLOW_REAR);
                if (en_r_l < 32000)
                {
                        overflow_r_l = overflow_r_l + 1;
                }
                else
                {
                        overflow_r_l = overflow_r_l - 1;
                }
        }

        en_r_r = encoder_rear(MTR_RIGHT);
        if (right_motor_status & ERROR_ENCODER_OVERFLOW_REAR)
        {
                motor_reset_status(MTR_RIGHT, ERROR_ENCODER_OVERFLOW_REAR);
                if (en_r_r < 32000)
                {
                        overflow_r_r = overflow_r_r + 1;
                }
                else
                {
                        overflow_r_r = overflow_r_r - 1;
                }
        }

}

int main(void)
{

    initialise();
        turning = 0;
        encoder_reset = 0;
        char status;

        while(1)
        {
                check_IR();
                get_encoder_data();
```

```
                //polled to see if anything had be recieved from the FitPC
                if (U2STAbits.URXDA)
                {
                        protocol();
                }

                //When a drive commade is sent other then turning the right and left encoder values reset
to 0
                if ((turning == 0) & (encoder_reset == 0))
                {
                        status = spi_command(MTR_RIGHT, READ_RESET_ENCODER_REAR, 0, 0);
                        status = spi_command(MTR_LEFT, READ_RESET_ENCODER_REAR, 0, 0);
                        encoder_reset = 1;

                        sendSerial(not_rotating);
                }

                //If the receiver buffer overflows, clear flag to enable receiving again
                if(U2STAbits.OERR == 1)
                {
                        U2STAbits.OERR = 0;
                }
        }
}

// ISR for Analog Sensors
void __attribute__((interrupt, no_auto_psv)) _DMA0Interrupt(void)
{
    process_DMA0_interrupt();
}

// ISR for IMU
void __attribute__ ((interrupt, no_auto_psv)) _U1RXInterrupt(void)
{
    process_U1RX_interrupt();
}
```

Appendix D – UTM-30LX Specifications

Date: 2009.5.18

Scanning Laser Range Finder
UTM-30LX/LN
Specification

Symbol	Amendment Details	Amendment	Date	Amended by	Number
④× 2	Correction on synchronization output	2.4	2009.5.18	Takai	PR-5647
③× 2	Changes in laser(λ :870n→ 905nm)	2.3	2009.4.14	Kamon	PR-5635
②× 1	Correction	4	2008.8.18	Kamitani	PR-5503
①×1	Cautions were added	6	2008.5.1	Kamitani	PR-5466

Approved by	Checked by	Drawn by	Designed by	Title	UTM-30LX/LN Specification	
	MORI	KAMITANI	HINO	Drawing No.	C-42-3615	1/6

1. Introduction

1.1 Operation principles

905nm △3

UTM-30LX/LN use laser source (λ = ~~870nm~~) to scan 270° semicircular field (Figure 1). It measures distance to objects in the range and co-ordinates of those point calculated using the step angle. Sensor's measurement data along with the angle are transmitted via communication channel. Laser safety class 1.

Sensor is divided into two types depending upon the type of output.

1.2 Type

1.2.1 U TM-30LX

Synchronous output signal is available. The timing chart of this signal is shown in section 6 (Figure 3). △4 This synchronous signal can be obtain at each scan. These are mainly intended for robotic applications.

1.2.2 UTM-30LN

It outputs warning signal whenever there is any object in the preset area. These are mainly intended for area protection.

2. Structure (Laser range figure)

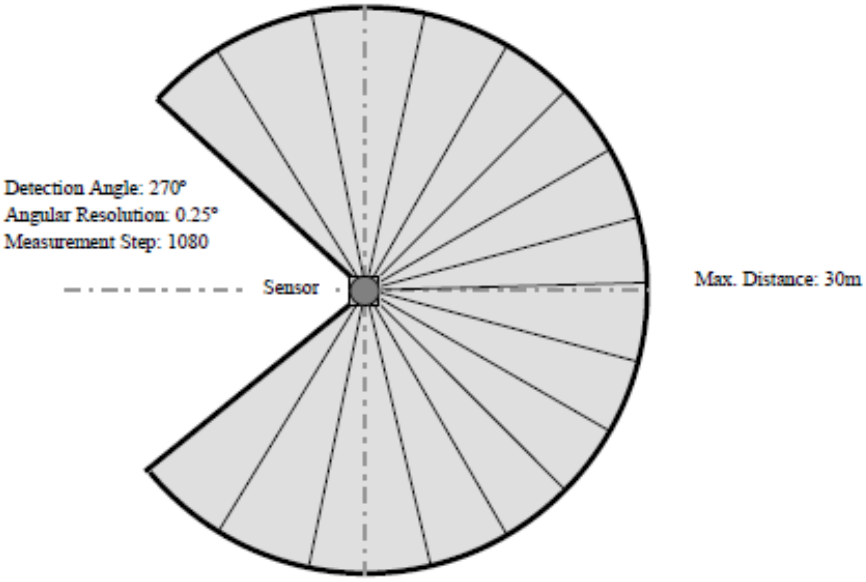


Figure 1

3. Important note

- This sensor is not a safety device/tool
- This sensor is not for use in military applications
- Read specifications carefully before use.

4. Specifications

Product Name	Scanning Laser Range Finder	
Model	**UTM-30LX**	**UTM-30LN**
Light Source	Laser Semiconductor λ = ~~870nm~~, 905nm ⚠ Laser Class 1	
Supply Voltage	12VDC ± 10%	
Supply Current	Max: 1A, Normal : 0.7A	
Power Consumption	Less than 8W	
Detection Range and Detection Object	Guaranteed Range: 0.1 ~ 30m (White Kent Sheet) Maximum Range : 0.1 ~ 60m Minimum detectable width at 10m : 130mm (Vary with distance)	
Accuracy	Under 3000lx : White Kent Sheet: ±30mm*1 (0.1m to 10m) Under 100000lx : White Kent Sheet: ±50mm*1 (0.1m to 10m)	
Measurement Resolution and Repeated Accuracy	1mm 0.1 – 10m : σ <10mm, 10 – 30m : σ <30mm (White Kent Sheet) Under 3000lx : σ = 10mm*1 (White Kent Sheet up to 10m) Under 100000lx : σ = 30mm*1 (White Kent Sheet up to 10m)	
Scan Angle	270°	
Angular Resolution	0.25° (360°/1440)	
Scan Speed	25ms (Motor speed : 2400rpm)	
Interface	USB Ver2.0 Full Speed (12Mbps)	
Output	Synchronous Output 1- Point	Warning Output 1- Point
Ambient Condition (Temperature, Humidity)	-10°C ~ +50°C Less than 85%RH (Without Dew, Frost)	
Storage Temperature	-25~75°C	
Environmental Effect	Measured distance will be shorter than the actual distance under rain, snow and direct sunlight*2.	
Vibration Resistance	10 ~ 55Hz Double amplitude 1.5mm in each X, Y, Z axis for 2hrs. 55 ~ 200Hz 98m/s² sweep of 2min in each X, Y, Z axis for 1hrs.	
Impact Resistance	196m/s² In each X, Y, Z axis 10 times.	
Protective Structure	Optics: IP64	
Insulation Resistance	10MΩ DC500V Megger	
Weight	210g (Without cable)	
Case	Polycarbonate	
External Dimension (W×D×H)	60mm×60mm×85mm MC-40-3127	

*1 Under Standard Test Condition (Accuracy can not be guaranteed under direct sunlight.)
*2 For sensor functions, please verify the in an indoor environment of 1000 lx or less. In avoiding unnecessary disturbance cause by the raindrops, perform necessary signal processing for LX type and switch OFF the delay function for LN type.

5. Quality Reference Value

Vibration resistance during operation	10~150Hz 19.6m/s² Sweep of 2min in each X,Y,Z axis for 30min
Impact resistance during operation	49m/s² X, Y,Z axis 10 times
Angular Speed	2π/s (1Hz)
Angular Acceleration	π/2rad/ s²
Life-span	5 Years (Varies with operating conditions)
Noise Level	Less than 25dB at 300 mm
Certification	FDA Approval (21 CFR part 1040.10 and 1040.11)

6. Interface

6.1 Robot Cable 4 Pin

Color	Function
Brown	+12 V
Blue	0 V
Green	Synchronous Output/ Warning Output
White	COM Output (0V: Common to Power)

Note: 0 V of the power supply and Output is not internally connected. Connect it when it is necessary ⚠

6.2 USB Connector

TYPE-A

Note:

SG for communication and GND are connected internally (Isolated with Input -VIN).
Isolate the device from any connection that generate electric noise.
This sensor is compatible with SCIP2.0 communication protocol standard.

6.3 Output circuit diagram

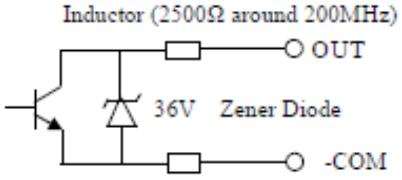


Figure 2

7. Control Signal

7.1 Synchronous Output (UTM-30LX)

1 pulse is approximately 1 ms. Output signal Synchronization timing chart is shown below. (Figure 3).

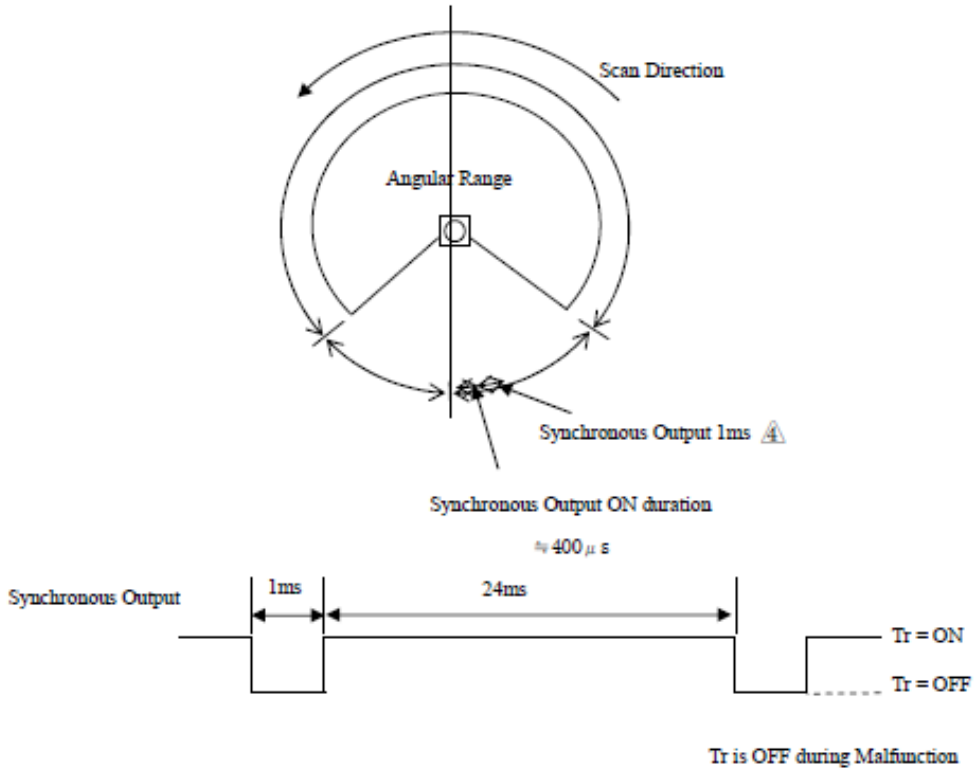


Figure 3

7.2 Warning Signal (UTM-30LN)

Warning signal can be obtain by switching ON the signal through the application software. When the output signal is set fro warning signal, output signal switch OFF when obstacles exist inside the warning area. (Output signal is ON when obstacle does not exist.

Area can be set using 3~7 co-ordinate points.

Maximum of the output delay is 128 times (3.2 sec)

Example

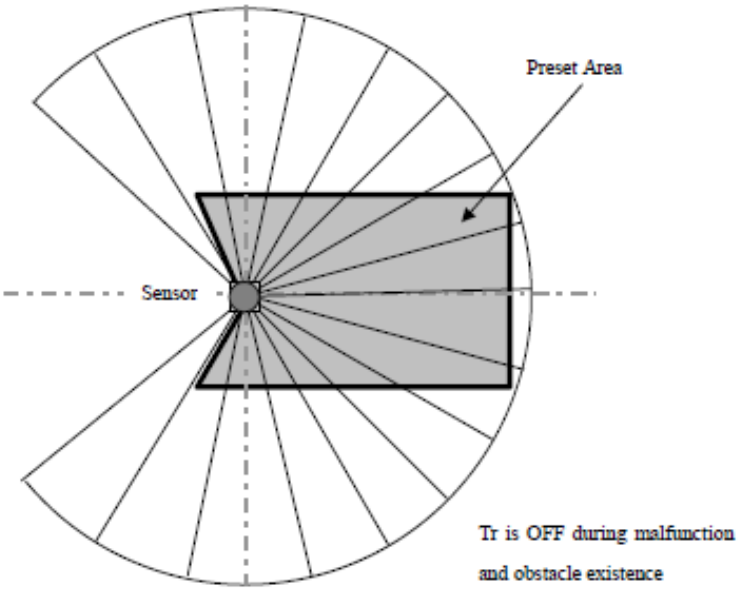


Figure 4

8. Malfunction Output:

1. Laser malfunction : When laser does not radiate or exceeds safety class 1.
2. Motor malfunction : When rotation speed is differ from the default value (> 25 m/s).

Synchronous/Warning signal will be turned OFF when these malfunctions are detected. Error details can be obtain via communication.

9. Cautions ⚠

Heat is generated as the sensor runs at a very high speed. The heat generated is concentrated at the bottom of the sensor. Please mount heatsinks or any appropriate component to release the generated heat. An aluminum plate (200 x 200 x 2) is recommended as the heatsinks.

Error could happen when 2 or more identical sensor is mounted at the same detection plane. This is because the sensor could not identify the origin of the received laser pulses. When this error occur, it will cause 1 -2 step difference, performing data filtering could overcome this problem.