Autonomous SLAM Robot

Mechatronic Design and Integration

Team Members:

Matt Evans
6833484

Reuben Finch
2499992

Michael McGurk
5497685

Jonathan Chaplow
6596001

Authored:
6/7/2015
Executive Summary

An autonomous SLAM robot is designed and built for a room cleaning application. The robot is based on a VEX robotics chassis powered by four individual servo motors and mecanum wheels. Infrared sensors, a 9 degree of freedom inertial measurement unit and a servo motor are integrated with an Arduino Mega 2560. The robot must be able to map its surroundings while also knowing its location within the map. To achieve the target performance outputs, the robot must navigate autonomously through the environment, covering the entirety of a defined room area in the shortest time possible while avoiding any obstacles encountered. The robot communicates with a computer via Bluetooth to send localisation and mapping data via Bluetooth for display.

The VEX robotics chassis was supplied complete and the position of sensors and actuators selected in order to design an efficient and practical SLAM implementation. Concurrent design principles were used to select sensor arrangements and overall programming structure simultaneously. Testing of the capabilities and limitations of the provided equipment drove an evolutional design process from which the end result was derived. The final solution incorporated a motion processing unit for bearing control, two side mounted infrared sensors for wall tracking and mapping and two front facing rotatable infrared sensors for mapping and object detection. A series of finite state machines controlled the execution sequence of the robots motion.

The robot was demonstrated to meet the design criteria on evaluation. An accurate map was produced of the test area showing the location of the walls, obstacles and the path the robot had taken. Two minor errors occurred during demonstration, a small section of the map was not covered by the robot and an obstacle was lightly hit by a mecanum wheel. Despite these two issues, the overall performance of the robot was satisfactory. Suggestions for future improvements are a more flexible method of control and more accurate tools for odometry.
Contents

Executive Summary........................................................................................................................................ii

1.0 Introduction........................................................................................................................................... 1

2.0 Problem Description........................................................................................................................... 2
  2.1 Project Aims ......................................................................................................................................... 2
  2.2 SLAM.................................................................................................................................................. 3

3.0 Project Specifications .......................................................................................................................... 5
  3.1 The Mecanum Wheeled Robot .......................................................................................................... 5
  3.2 Sensors and Actuators ..................................................................................................................... 6
  3.3 Project Requirements ...................................................................................................................... 6

4.0 Systems Design and Integration ...................................................................................................... 7
  4.1.0 Data Acquisition .......................................................................................................................... 10
    4.1.1 IR Sensors ................................................................................................................................... 10
    4.1.2 Servo Motor ............................................................................................................................ 13
    4.1.3 MPU .......................................................................................................................................... 14
    4.1.4 Mecanum Wheels ..................................................................................................................... 18
  4.2.0 Motion Control ............................................................................................................................. 20
    4.2.1 Forward Motion .......................................................................................................................... 21
    4.2.2 Strafing ....................................................................................................................................... 22
    4.2.3 Turning ....................................................................................................................................... 22
  4.3.0 Sequence Control ......................................................................................................................... 25
    4.3.1 Align to Wall & Initialise .......................................................................................................... 26
    4.3.2 Map Walls................................................................................................................................... 27
    4.3.3 Cover Area .................................................................................................................................. 27
    4.3.3 Avoid Obstacle .......................................................................................................................... 28
  4.4.0 Localisation and Mapping ............................................................................................................ 30
    4.4.1 Localisation .............................................................................................................................. 30
    4.4.2 Mapping ..................................................................................................................................... 30

5.0 Testing ................................................................................................................................................ 35
  5.1 Infrared Sensors ............................................................................................................................... 35
  5.2 Sonar Sensor ..................................................................................................................................... 36
  5.3 MPU ................................................................................................................................................ 36
  5.4 Bluetooth ......................................................................................................................................... 37
  5.5 Mapping program ............................................................................................................................. 37
  5.6 Servo Motors and Mecanum Wheels ............................................................................................... 38
List of Figures and Tables
Figure 1: The SLAM robot ................................................................. 1
Figure 2: The SLAM arena ................................................................. 2
Figure 3: SLAM Venn diagram ......................................................... 2
Figure 4: Graphical model of the SLAM problem .............................. 3
Figure 5: SLAM mapping using particle filters [1] .............................. 4
Figure 6: Mecanum wheel layout ...................................................... 5
Figure 7: SLAM robot schematic ...................................................... 9
Figure 8: Short range IR sensor calibration ....................................... 11
Figure 9: Long range IR sensor calibration ....................................... 11
Figure 10: Side IR sensors .............................................................. 12
Figure 11: Front IR sensors ............................................................ 12
Figure 12: Level shifter .................................................................. 14
Figure 13: Magnetometer bias set .................................................. 15
Figure 14: MPU mounting position and level shifter .......................... 16
Figure 15: Magnetometer calibration .............................................. 17
Figure 16: Bot speed vs. motor speed - forward motion .................... 18
Figure 17: Mecanum wheel odometry code ..................................... 19
Figure 18: Finite state machine architecture ..................................... 20
Figure 19: Wall spike handler Arduino code ..................................... 21
Figure 20: Multiple PID controllers ............................................... 22
Figure 21: Turn to near wall Arduino code ...................................... 23
Figure 22: Deadband compensation code ...................................... 24
Figure 23: Robot motion during wall finding ................................... 25
Figure 24: Robot motion during wall mapping .................................. 25
Figure 25: Robot motion during map covering ................................................................. 25
Figure 26: Robot motion during obstacle avoidance ........................................................ 25
Figure 27: Align & initialise FSM ...................................................................................... 26
Figure 28: Time delay and stop condition of wall alignment code ...................................... 26
Figure 29: Map walls FSM ................................................................................................. 27
Figure 30: Cover map FSM ............................................................................................... 27
Figure 31: Obstacle avoidance sequence diagram .............................................................. 28
Figure 32: Section of isWall() code .................................................................................... 29
Figure 33: Robot map in progress ...................................................................................... 31
Figure 34: Serial tab ........................................................................................................... 32
Figure 35: Map tab ............................................................................................................. 33
Figure 36: Main terminal .................................................................................................. 34
Figure 37: IR shield ........................................................................................................... 36
Figure 38: SLAM robot components ................................................................................... 38
Figure 39: Map produced during evaluation run ............................................................... 40
Figure 40: The path taken as recorded by the evaluation camera ....................................... 41
Figure 41: Bot speed vs. motor speed - forward motion .................................................... 46
Figure 42: Bot speed vs. motor speed - reverse motion ..................................................... 46
Figure 43: Bot speed vs. motor speed - left strafe .............................................................. 47
Figure 44: Bot speed vs. motor speed - right strafe ........................................................... 47
Figure 45: Front IR housing mechanical drawing .............................................................. 48
Figure 46: Side IR single housing mechanical drawing ....................................................... 49
Figure 47: MPU stand mechanical drawing ..................................................................... 50
Figure 48: Level shifter schematic .................................................................................... 51
Figure 49: IMU block diagram .......................................................................................... 51
1.0 Introduction
The objective of this project is to complete an autonomous room sweeping robot which progressively tracks back and forth ultimately traversing the entire floor area. The design will incorporate simultaneous localisation and mapping (SLAM) in order to determine its position and build a map of the environment. The application for the design is a vacuum cleaning robot. Similar to commercial vacuum cleaning robots, the design will have the capability to avoid obstacles and trace a path through its own decision making. To complete this task a VEX robot has been provided with omnidirectional wheels as well as a selection of sensors and actuators. Placement of the sensors and the structure of the robot software are left undefined, providing an entirely open ended problem.

This report details the design process, successes, failures and features of the autonomous SLAM robot. A concurrent design methodology was applied to selecting the form and operation of the robot. The position and arrangement of the supplied sensors and actuators were considered alongside the operating principle of the autonomous robot. This process also simultaneously directed the structure of the control software. Integration and calibration of the sensors and actuators is detailed. Limitations of the hardware provide evidence of the adoption of an iterative design process, as the strengths and weaknesses of the proposed systems are discovered through testing. Selected methods of control are discussed along with the structure and operation of the robot software. Data acquisition and display techniques are described. The design process is detailed in which individually developed modules are integrated into a fully functioning SLAM algorithm. A critical discussion of the robot’s performance was undertaken, discussing the performance, strengths, weaknesses and limitations of both the supplied parts and the implemented system.
2.0 Problem Description

The SLAM robot must be completely autonomous. The robot will be placed in the testing area in a random location and perform without any human interference (Figure 2). Once within the arena the robot must decide the best way to map the entire room covering the surface area presented in the least amount of time possible. While navigating and mapping the room, the robot must also identify and avoid obstacles. To this end, the machine will decide on the best path to optimise both time and area covered. The robot outputs a map as it covers the room, showing the walls, the area covered and the obstacles discovered.

![Figure 2: The SLAM arena](image)

2.1 Project Aims

The project aims to combine knowledge attained in mechanical design, electronic hardware and software development into the design of an intelligent machine. Integrating sensing, signal processing, motion actuation and programming, the project aims to provide a synthesis of engineering ability, problem solving, system design and teamwork. Development of an autonomous SLAM robot provides the ideal platform to incorporate the multidisciplinary approach of the Mechatronics program and test the practical application of theoretical approaches (Figure 3).

![Figure 3: SLAM Venn diagram](image)
2.2 SLAM

SLAM is simultaneous localisation and mapping. This is the problem of constructing a map of an unknown environment while keeping track of the robot's position within the environment. While SLAM appears to be a chicken and egg problem, there are a number of approaches to solving it. The majority of SLAM algorithms use probabilistic models to estimate position (Figure 4). As such, SLAM is generally accurate for environments that are fixed, structured and limited in size. Attempting to perform SLAM in large environments with dynamic behaviour is much more difficult due to unpredictable events generating additional uncertainty in SLAM algorithms.

![Figure 4: Graphical model of the SLAM problem](image)

SLAM can be split into a number of graphical and computational techniques and have been established as the taxonomy of the SLAM problem. The sampling can be volumetric or feature-based. This means that either the entire map is sampled at a rate that is approaching photorealism, or just mapping the distinguishable features. This depends on if the output required is for human readability or the efficiency for the system.

The use of a topological approach is also considered an alternative to metric information being set to the location of landmarks/places of interest on the map. This is not commonly used despite humans tendency to use such techniques ourselves i.e. Location A is west of Location B. as opposed to the metric approach of Location A is 20 metres west of B. The recognition of objects is also very important in SLAM and can reference to a known object/landmark or store and compare unknown landmarks. This is a very difficult problem in slam, as the observation of landmarks via sensors changes how it is observed, but still needs to match it with a previous or stored observation. The addition of movement into the mapping space changes the requirements of the sensors dramatically compared to a static
system. As members of the environment need to be analysed for velocity and acceleration on top of location and orientation. The certainty of a SLAM systems’ location can be dramatically adjusted through the use of a more absolute positioning system i.e. GPS. This allows adjustment to drifting or the accrualment of errors in a run. The SLAM robot can also be active or passive, or that its sensor data and observations are on board the robot, or watching the system from outside. This use of either technique depends on the expected location of the robot to run in, as mapping an unexplored tunnel system can’t be observed easily, but vacuuming a house can. A final aspect is the use of multiple robots for a single system, as this allows robots to reference their locations based on surrounding ones. Simultaneously controlling multiple robots adds difficulty that a single unit does not.

Beyond this SLAM can be broken down into three main paradigms, Kalman filter, particle filter, and graph based SLAM. The Extended Kalman Filter uses metric, feature-based techniques to display its surroundings, making calculations as it goes. But has a slowly increasing error as feature location and pose location error goes up. The graph based technique makes much simpler samples of pose and environment, as a nodular or gridded map. The edges of the features are used as reference when processing the map data to overlay the many samples taken in relation to the robot pose. This method however requires a lot of data space preallocated and high computational power. It can be used on a much more accurately than Kalman however. Particle method uses probability to determine the most likely and useful particle of a feature to scan for as a higher chance of change is expected from previous changes. Kinematics can be used with this resampled data to determine the velocities and acceleration of features and therefore locational information can be derived. This is a very easy technique to implement despite being a very complex idea.

Figure 5: SLAM mapping using particle filters [1]
3.0 Project Specifications
A robot chassis has been provided on which the basis of the project is formed. The chassis provided is based on VEX Robotic parts and comes complete with a set of mecanum wheels, battery, drivers and an Arduino 2560. Basic interfacing software such as motor control, IR sensor reading and battery level checks are also provided. A kit consisting of multiple IR sensors, a sonar sensor, motion processor unit (MPU) and servo motor are also provided, with their configuration chosen during design. A 2x1.5m rectangular arena and obstacles are provided for testing.

3.1 The Mecanum Wheeled Robot
Motion of the robot chassis is provided by four mecanum wheels. These wheels have a central hub that host free spinning rollers at a 45° angle about the hubs circumference. The mecanum wheels allow the platform to be driven in any direction, without the need for changing the orientation of the wheels. The wheels also allow for on the spot rotational motion. The wheels give the advantage of providing unrestricted directional motion. Figure 6 shows the orientation of the mecanum wheels used to achieve omnidirectional motion. The main disadvantage when using mecanum wheels for SLAM purposes is the slippage that can occur between the rollers and contact surface. This limits the accuracy of odometry calculated purely by the rotation of the mecanum wheels. The controller for the robot chassis is the Arduino Mega 2560. A shield is provided for the board which handles the IO between the various motors and sensors. This board is hosted on a base constructed of VEX Robotics components, which houses a battery pack and four Vex 2-wire 393 servos. The VEX framework provides a simple method of extending the chassis to mount additional hardware and supporting circuitry.

![Figure 6: Mecanum wheel layout](image)
3.2 Sensors and Actuators
The kit of sensors and actuators supplied included four IR sensors, a sonar sensor, a small servo and a Motion Processor Unit (MPU). The use of additional hardware was not permitted, however passive components were allowed for additional interfacing and signal processing and conditioning circuitry.

The IR sensors provided were the SHARP GP2Y0A21YK and the GP2D120XJ00F. Two of each model were provided, providing a short and long range of object detection. The long range infrared sensors have a range of 200mm to 1500mm while the short range sensors have a range of 40mm to 300mm. The sensors have data acquisition circuitry embedded and are able to be polled for new distance information at speeds up to 6ms. Code to handle sample averaging and convert the raw analogue signal from the sensors to usable data was provided. The sonar sensor provided was an MB1000 LV-MaxSonar-EZ0 from MaxBotix. The sonar provided a 0.12 to 6.45m range with a 2.54cm (1") resolution. The sonar sensor provides a broader cone of detection as opposed the straighter line detection of the IR sensors. The sonar is able to be polled at speeds of up to 10ms. A hobby grade servo motor was also supplied on which to mount sensors with a usable range of around 0 to 140°.

The MPU is a 9 degree of freedom (DOF) smart sensor made up of nine individual sensors. Three accelerometers, gyroscopes and magnetometers are incorporated in the device, one of each for each axis (Figure 49). All the sensors are MEMS devices constructed within a single integrated circuit. The final element of the MPU is the processor itself. A proprietary ultra-microprocessor called the Digital Motion Processor (DMP) which can be programmed to read the sensors internally and produce a data-fused output. The DMP is a hardware accelerator, designed to perform complicated mathematical fusing operations quickly and without using excessive power.

3.3 Project Requirements
The robot will be placed randomly in a rectangular playing field 1.5 metres wide and 2 metres long. The robot must navigate the area while avoiding contact with any walls or obstacles. Complete coverage of the area must be performed by the robot. A map of the area must be constructed showing the location of wall, the position of any obstacles and the area that has been covered by the robot. No human interaction is allowed while the robot is in motion. The performance of the robot will be evaluated on coverage of the area, avoidance of walls and obstacles and time taken to complete coverage. The robot must be constructed using the equipment provided and will perform all processing using the onboard microcontroller.
4.0 Systems Design and Integration

In designing the form and operation of the robot, concurrent design methodology was employed. The method in which the robot acquires data and navigates is tightly coupled with the physical arrangement of the sensors and actuators. It was therefore identified that the choice of positioning and the roles assigned to each sensor would have a major impact on the structure of the software and performance of the robot. A variety of arrangements were drafted and experimented with before the final layout was reached. It was found through testing that while many of the designs functioned well in theory; limitations of the hardware prevented some designs from being practical.

An early version of the SLAM robot involved mounting a short range IR, long range IR and sonar on a spur gear in the forward facing position. The assembly would be driven by a second gear mounted on the limited range servo, providing a full 360° of rotation from the limited motion of the servo. It was theorised that a single cluster of sensors could accurately provide sufficient data for mapping, localisation and obstacle avoidance. Upon testing, two observations were made that made this design impractical. The data from the sensors is not always accurate. Multiple readings need to be taken and averaged to obtain useful data. This process takes time and as such it was not practical to perform such a multitude of tasks with a single cluster of sensors, particularly when both the sensors and the platform are in motion. The second observation was the apparent drift of the MPU over time. While the drift was not critical, it was apparent that a second means of position tracking was required in order to keep the bot tracking true.

After the first concept was scrapped, it became clear that an iterative process was required in order to produce a device capable of sophisticated localisation and mapping. In order to facilitate this, a number of additional concepts were outlined. The ability of the hardware to perform to the required standard was then tested, with the team reporting back with their conclusions on the feasibility of each idea.

It was suggested that the sonar could be used for obstacle detection and also to perform localisation and mapping data acquisition. After thoroughly testing the device, it was found that the device was not accurate enough for mapping or localisation. While the sonar provided a board detection beam for obstacle avoidance, it was limited by the minimum read time. When combined with additional time taken to store and average multiple reads, it was concluded that the performance was not ideal for our purposes. The provided device was also found to be affected by noise and prone to false readings. Use of the sonar was abandoned entirely.
Various techniques were experimented with in order to achieve accurate localisation. The three main techniques attempted were odometry from the output to the mecanum wheel servos, position estimation using periodic IR readings and distance approximation by double integrating MPU acceleration data. Double integrating the MPU data was found to be infeasible. The error involved in this process was extreme. This was primarily due to the effect of the gravity vector on the X and Y axis. Any minute deviation causes the gravity vector to project onto the XY plane, sending the distance approximation far beyond any realistic value. Research into the implementation of the MPU supported our results and suggested that it is infeasible to use the device in this fashion. Odometry via the servo outputs was found to be adequate, provided the robot was driven in a manner that fit our derived model. Actions such as driving in arcs, braking time and wheel slippage caused some inaccuracies in the localisation data. Using periodic IR readings to estimate position was found to be reasonable. The limited range of the IR sensors and changes in sensor position had a significant impact on the effectiveness of localisation. Ultimately a combination of IR and servo odometry was implemented.

Through testing of various filter and data fusion techniques, the MPU was tuned to produce a reliable bearing output. It was noted however that a drift of approximately 1° per minute was observed. It was decided that the MPU bearing alone was not enough to guarantee straight line motion. Side facing IR sensors were implemented to allow the robot to track its position against a wall and maintain straight line motion. Provision was allowed for any irregularities in the wall position, such as an obstacle passing between the wall and the sensor.

The planned path of the robot and order of operations was also considered in depth. Factors considered were the length of time taken to complete a run, the extent of coverage of the area and the difficulty level of programming the control algorithms with respect to feasible sensor arrangements. Pseudo-code algorithms for spiral type manoeuvres were written to gauge the impact on performance. Although such movements are common in commercially available vacuum robots, it was observed that accurately programming obstacle avoidance and coverage into the SLAM robot would not be as straightforward as other methods. The increasing number of turns and distance from the walls suggested the algorithm would be error prone. In order to achieve the highest map accuracy in the minimum time, a two stage coverage operation was designed. The robot would perform an initial mapping run in which the outer walls were covered. The inner area would then be covered by the robot zigzagging over the uncovered section. This allowed for greater accuracy in mapping as the wall data could be read once, with obstacle data being added as new objects were discovered. This reduced the risk of inaccuracies that were involved in using the IR sensors at a range and removed that risk of conflicting wall reads being apparent in the map.
After extensive testing, the final form of the robot was designed, consisting of a side facing short and long range IR and a forward facing servo controlled short and long range IR. The IRs were calibrated, aligned and fixed using rigidly bolted metal VEX parts. The MPU was cushioned from vibration using a rubber pad and mounted away from metal parts and motors generating electrical noise. Covers were added to shield IR receivers from being triggered by stray light sources. Spiral wrap was added to tidy cables, allowing for fewer tangles, easy identification and reduced risk of damage. A capacitor was added to the 5V rail on the Arduino board as a precautionary measure intended to reduce the impact of noise and voltage drop due to periodic sensor readings from multiple devices and the high current draw from the wheel servos when accelerating. Figure 7 depicts the design of the autonomous robot system.
4.1.0 Data Acquisition

Data was taken in from several sensors to allow for SLAM, the combination of these inputs gave the robot an effective “view” of where it is according to obstacles and walls around it. The IR sensors are placed on the front and left side of the robot while the MPU sits at the rear constantly reading in position. The sonar sensor was not used as it was deemed too inaccurate for use in obstacle detection.

4.1.1 IR Sensors

There are two types of IR sensors being used within the project; these are the long range GP2Y0A02 and the short range GPD2120XJ00F. The long range IR sensor has a minimum range of 200mm and a maximum range of 1500mm while the short range has a minimum range of 40mm to a maximum of 300mm. Each of these sensors send out an infrared light. The time taken for the light to return is calculated by circuitry on board the IR sensor and converted to a voltage output. This voltage is then sent to an analogue pin on the Arduino which is read periodically and converted to distance using code on board the Arduino. The readings update every 6ms. As the values are read by the Arduino, they are summed and averaged over a defined sample. This sampling is necessary as the signals are not 100% accurate and are affected by ambient IR light from the environment. A range of sampling averages were tested and different amounts used based on the application. Accuracy and speed are direct trade-offs when integrating the IR sensors. For high accuracy map data, an average of 30 reads was preferred. For obstacle detection, an average of 5 reads was sufficient. During testing it was noticed that the IR sensors could be prone to error when placed side by side or used under fluorescent lighting. To reduce this effect, each of the sensors were placed within an additively manufactured cover to reduce interference from other sensors. It was also noted during testing that the output of the IR sensors was affected by the material type and colour it was directed to. To account for this, the sensors were tested and calibrated as close to the evaluation environment as possible, by placing the robot in the testing arena while plenty of daylight was present.

The two types of sensors are used in pairs on the SLAM robot, with one pair facing the left of the robot while the other pair faces forward. The forward facing pair are mounted on the servo motor, allowing the forward IR’s to sweep the area ahead of the robot. The placement of short and long range IRs in pairs allowed the robot to have a sidewall in range during all parts of its motion. This was key in implementing accurate motion control, allowing the robot to maintain a set distance from a wall at any given position, ensuring the robot moved in straight lines at all times. This compensated for gyro drift and allowed for coverage of area in the shortest time. The short and the long range IR have an overlap in their ranges. In order to ensure consistent values, the average of the long and short ranges was taken when the distance was in range of both sensors. The sensors were tested to determine their accuracy. It was noted that the sensors tended to be less accurate at the limits of their rated range. To compensate for this, experimental readings were taken from the sensors at a variety of ranges and a calibration curve fitted the resulting data (Figure 8, Figure 9). The equation of the curve was then used to modify the raw sensor readings.
The forward facing IR sensors perform a multitude of tasks, detecting whether an object is ahead of the robot, determining whether an object is a wall or an obstacle, mapping any objects discovered and aiding in localisation. When the robot is run in forward motion, the sensors sweep from side to side, searching for obstacles. During testing it was noted that a trade-off between the amount of reads and object detection was required. As the robot is in motion, it must be able to respond to an event quickly. This proved to be problematic when the IR sensors attempts to detect an obstacle in front of the robot, as the IR beam is narrow and must be swept across the front of the bot to cover the entire area. This process takes time for the servo to move and additional time for multiple readings to be taken and averaged. Additional problems occurred if too wide of a sweep was conducted, increasing...
the risk of the robot detecting a side wall as an obstacle ahead. Before the forward sweep was implemented, the robot was running at relatively high speed. When implementing obstacle detection using the IRs, the speed had to be halved in order to meet the requirement of detecting obstacles. Using IRs for obstacle detection had the biggest negative impact on the speed of the robot. The initial program would sweep the sensor from left to right, taking an average of 10 readings per degree of the servo. This proved much too slow. After many trial and error exercises, an ideal balance was reached consisting of the average of five readings across four positions of the servo. This was enough to ensure maximum coverage in the minimum amount of time. It is important to note that when looking for an obstacle or a wall, the accuracy of the reading is less important. The cycle of the servo was also altered to optimise detection. While the original swept from left to right, it was found that a right to left recurring sweep was more efficient. While this introduced an additional delay in travel time, it resulted in the area ahead of the bot being covered by the IR sensors more evenly. It was also realised that the delay in motion of the servo could impact the accuracy of readings. As far as the controller is concerned, the servo is in position the instant a new voltage is updated at the output. In reality, a finite amount of time is required for the servo to move. In order to compensate for this, when the servo was performing high accuracy operations such as mapping the rate of change of the servo was dramatically reduced.

Figure 10: Side IR sensors

Figure 11: Front IR sensors
Both the forward facing and side facing pair of IR sensors are used for mapping. During a run, the robot circles the perimeter of the arena, reading in data from the side sensors and sending it to a PC via the Bluetooth module. Turns during the mapping run are kept at a low speed to allow maximum data gathering. The forward facing sensors are swept across any detected obstacles to add their absolute position to the map (Figure 10, Figure 11). The position to the object is calculated using the current robot position, bearing and the bearing of the top mounted servo.

4.1.2 Servo Motor

The servo motor is located on the front of the robot. It has the ability to rotate 160° from its origin and takes in a voltage either positive or negative to move it. The servo was positioned so that the midpoint of its motion sits at the North bearing relative to the bot frame, providing equal swing left and right of the centre. This was determined to be at 78° after the sensor mount was attached using the supplied splined coupling. Initially the sonar sensor was placed on top of the servo, the sonar was deemed to be inaccurate for the purpose so was not used. The pair of IR sensors were tested and approved as appropriate for the job and were mounted within their casings on top of the servo motor.

In order to ensure that the positions of the servo were valid, calibration was performed. A long rod was attached to the top of the IR housing on the servo and a blank piece of paper placed under it. The servo was then stepped through 1° increments through the full range of its motion, and a line of the rod position traced. It was discovered that the servo produced repeatable increments between 15 and 130°. While additional motion was possible, it did not fit the rest of the data and was not used. The range of the motion was measured and the actual average rotation per degree sent to the servo was taken. It was found that the servo only moved 0.8727° per one degree sent over the defined range. This was used in calculations when determining the distance from the bot to a wall or object and also when collecting data for mapping and proved to be accurate.

There was an issue discovered when sweeping the sensors. If the servo sweeps too fast the data it collects at each step is inaccurate. This is because it does not have enough data to average, resulting in a poor reading. Yet if it sweeps too slowly the robot will collide with obstacles or the wall as it will not have detected them fast enough. Therefore the sweeping cycle had to be optimized to sweep fast enough to detect obstacles yet slowly enough to make sure the readings were accurate.
4.1.3 MPU
The Invensense MPU 9150 was at the time of its release the world’s first nine degree of freedom inertial measurement sensor. The breakout board we were supplied with enabled simple two-way communication with the MPU using I2C. According to the manufacturer’s data sheet the MPU must not be supplied with a voltage higher than 3.465V. This was achieved using the Bluetooth power rail which runs at 3.3V. During early testing frequent I2C crashes were experienced. After significant consternation it was found that making use of level-shifting to maintain the Clock and Data lines at the maximum voltage of 3.3V alleviated the vast majority of the crashes (Figure 12). The proprietary nature of the Digital Motion Processor (DMP) is made understanding what method exactly was being used to fuse the accelerometer and gyroscope output.

Dead reckoning is an incredibly difficult task to perform at 100% accuracy. Initially we investigated making use of the accelerometer output to calculate distance by double integration. However the difficulty with integration noise made the accuracy required impossible to achieve. Due to the influence of the gravitational acceleration mounting the IMU chip even a single degree from level would result in over 60 thousand meters of error per minute. This method was thus quickly dismissed and achieved using the IR sensors and velocity based dead reckoning using the servo motor output signals.

Initially it was hoped that the combination of magnetometer and gyros would enable a true North bearing to be calculated. This would have simplified initialisation of the bot and aligning the tracking to the wall direction. After testing it was determined that due to the electromagnetic noise within the testing area the magnetometer output was not viable. Simply moving the bot to a different area of the room introduced significant variation to the output. The final MPU code was thus only concerned with providing a stable relative bearing for the bot control. The true difficulty in the verification of the magnetometer was due to the inconsistency provided by each of the sensors. Three different methodologies were
attempted to gain a stable and accurate reading. Firstly the raw sensor output for the X and Y dimensions were entered into an inverse tangent function and then mapped to a measurement in degrees. This result had significant noise in the output and due to unequal output values for each dimension the rotation of the IMU did not produce a consistent relative output. This meant that for 90 degrees of actual IMU rotation, the output may only change by 60 degrees, a further 90 degrees (for a total of 180 degrees) was only outputting 120 degrees and then the final 180 degrees to the initial heading would swing the remaining 240 degrees.

A function was written that took the maximum and minimum values returned for each of the X and Y dimension magnetometers and used those data points to scale the current reading within a -PI/2 to PI/2 value to input into the arctan function. This would allow the variability within the manufacture of the magnetometer sensors to be eliminated. However the function could not account for the non-linearity between each sensor. Meaning the numerical distance from the max value to the mean was not equal to the distance from the mean to the minimum value. More advanced offset calculation would be required to sufficiently calibrate the output into meaningful values.

The difficulty in calculating the gain offset for each sensor was the necessity to first establish the exact output range. However with the inconsistency of the magnetometer output it was difficult to know if the variation of the output was due to actual magnetic field changes or nearby ferromagnetic influence. It was noted during testing that even the presence of a watch could skew the output more than 20 degrees.

As detailed in the block diagram for the MPU9150 chip, the ADC which sequentially reads the magnetometer values has a register which can be set to alter the gain for the output. Setting the magnetometer offset gain was achieved by writing to this register (Figure 13). However even with these significant attempts at turning the sensors no meaningful output could be captured in the testing environment. The dependence on the Earth’s natural magnetic field, with a strength in Auckland of only 50,000nT (200 times weaker than a fridge magnet) means that almost any nearby metallic object can so significantly skew the output as to be unusable. Thus after extensive attempts the decision was made to abandon the magnetometer as a utilised sensor and rely on the optical wall tracking to align the robot.
Fusing the output of the accelerometers and gyros was attempted using a Kalman filter. However the drift was not acceptable. A Mahony and Madgwick [2] filter was also attempted but the noise level was high. The most stable and accurate solution was to disregard the magnetometer output and use the reverse engineered DMP firmware. Due to the proprietary nature of the manufacturers data fusion algorithm there is not published equations. The MotionApps file that was uploaded to the DMP as the firmware was decoded by Jeff Rowberg [3] and Noah Zerkin [4] and using I2C logic analysers. With the MotionApps DMP firmware loaded into the MPU the Arduino can simply read a packet of data which consists of an array of four Quaternions. These are then decoded using arctan2 into a heading value. The DMP takes about 10 seconds to settle to a stable value, thus we steady the bot for at least this time to enable a steady lock before moving. If the robot is disturbed before the MPU is allowed to settle, a high rate of drift occurs from the outset resulting in poor performance.

Significant attempts at using the magnetometer were made. Calibration involved quantifying the output of the sensors for every angle to be able to calculate the variability between the individual sensors. Even after calibrating for this variation the variability of the electromagnetic field within the testing area the output was not stable (Figure 15).

The remaining sensors contained in the MPU9150 namely the accelerometers and rate gyros were excellent. In order to provide the highest chance of accurate and stable values, the IMU was mounted on a 3D printed ABS standoff. Initially the sensor board was mounted on this standoff in order to provide physical separation for the contained magnetic sensors from the VEX robotics metal chassis, however after the determination that the magnetometers were not useful the placement of the remaining sensors had been undertaken and the

![Figure 14: MPU mounting position and level shifter](image)
standoff remained. In an attempt to ensure the harsh accelerations of the robot in the vertical direction by the Mecanum wheels would not cause the IMU accelerometers to overload the IMU was mounted on a section of elastomeric foam. This reduced peak accelerations and provided more stable output.

![Figure 15: Magnetometer calibration](image)

The final solution provided by the MPU9150 was to provide an accurate heading for the bot once the settling time had passed. The IMU was frequently polled to determine if the robot was on the correct heading, if not this error term was fed into a PID controller to bring the robot back on track. This control methodology was quite impressive to watch, due to the inconsistent grip and tracking rate offered by the Mecanum wheels the robot could be seen to track off path, but quickly the PID would augment the drive velocity for two of the motors and the robot would rotate and strafe mildly back to its course.

When the robot reaches a wall or an obstacle a turn is made, the direction depends upon the control state machine. The IMU output provides a feedback turn to ensure that the turns are accurate and consistent. Meaning the rot turns exactly 90 degrees each time, this is significant as the robot makes over twenty turns each run on the course and the accumulation of even a few degrees of error per turn would quickly add up to a very inaccurate sense of direction.
4.1.4 Mecanum Wheels

The mecanum wheels are actuated by a Vex 2-Wire 393 servo motor. This motor is fed a voltage through the Arduino board which is given an input in milliseconds. At 1500ms the motor is still and does not move, this number is increased or decreased to determine the direction in which the robot is to move. For example, to move forward the two left motors are fed a positive value while the two right motors are fed a negative. This is because the motors are mirrored; therefore need the opposite increment from 1500 to move in the same direction as the opposing wheels.

The control of this was used to implement a dead reckoning which the robot can use to determine its X and Y coordinates combined with the use of the gyroscope. A series of data was taken by increasing the ms fed to the robot by increments of 10 from 1500 up to 1800 or down to 1200 respectively. Meaning the robot would move faster with each increment. The time taken to travel one meter was recorded and this was repeated 3 times for each increment to give an average. This method was used for forward, backwards, and both strafing motions. From this data 4 3rd degree polynomial equations were fitted to the data. The output of the odometry calibration can be seen in Figure 16 Appendix B.

![Figure 16: Bot speed vs. motor speed - forward motion](image_url)
These equations were then used within each movement function to calculate the distance the robot had travelled. The motor speeds were averaged and input to the function. The millis() function was used in the Arduino code to serve as a time approximation, providing distance when combined with the velocity approximation (Figure 17). The performance of this dead reckoning technique was proven to be very accurate when the bot was driven precisely as modelled, i.e. in direct lines forward, backward, left and right. In reality, some inaccuracies were observed on implementation. If the four motors varies in speed to greatly, the approximation was up to 50% inaccurate. This occurred commonly in two parts of the robot’s motion. While wall tracking, the robot would drive two wheels faster than the others in order to maintain its alignment. This effect was more pronounced when the MPU was not exactly aligned with the walls of the testing area. Secondly, if the bot executed a turn and did not stop precisely at the target bearing, the motion control would compensate as the bot accelerated by driving two wheels faster than the other. To reduce the impact of the two primary causes of inaccuracy, drift compensation code was added to the MPU and the accuracy of the robots turn was addressed and optimised. The drift of the MPU was found to be predictable when used in the XY plane. A positive drift in bearing was observed at the rate of approximately 1° per minute. Turning the robot to within a degree of the target bearing proved to be a difficult challenge. This was due to the large deadband present in when driving the motors. With deadband compensation code implemented and a tuned PID controller, reliable results were experienced within 3° of the target bearing. While not 100% accurate, the improvements made a visible difference to the performance of the dead reckoning code. Combined with the IR sensors, the dead reckoning was able to produce very accurate results when covering the target area.

```cpp
// Compute total distance travelled by dead reckoning
transpeed = (LFSpeed + RSspeed + LRSspeed + RRSspeed)/4 - headingPIDoutput;

if (abs(prevmillis - millis) > 1000)
{
    // compute odometry translation
    t_dist += ((transpeed * transpeed) - (0.00000003 * transpeed * transpeed) - (0.0000002 * transpeed * transpeed) + (0.0159 * transpeed) - 0.003) * dt;

    prevmillis = millis;
}
```

Figure 17: Mecanum wheel odometry code
4.2.0 Motion Control

In order to handle the control over the motion of the robot, a finite state machine (FSM) was implemented (Figure 18). This state machine called functions that had three primary modes of motion, forward, strafe and turn. Individual parameters were passed to the functions, allowing for different varieties of each motion. Forward motion allowed the robot to wall track in the North, South, East and West direction relative to the testing walls. The robot was able to travel a pre-defined distance or run until a wall was found. Strafe motion allowed for left and right strafing relative the bots current bearing. Wall tracking was also provided in the mode using the forward facing sensors and bearing control implemented. Turning of the robot allowed relative left and right 90° turns, 180° turns and wall finding. In order to accurately track the bearing of the robot, global variables were used to determine the target bearing. This reduced any cumulative error that may have occurred if bearing values were calculated relatively.

Figure 18: Finite state machine architecture
4.2.1 Forward Motion

To control the forward motion of the robot, the Arduino sent a preset speed value to the wheel servos causing the robot to begin motion in the forward direction. In order to track true to the walls and to the target bearing, two PID controllers were implemented and their outputs added to the set speed value.

The wall tracking PID read the distance to the side wall at the start of each forward motion. This value was stored as the setpoint. As the robot moved forward, additional side wall distances were calculated and sent to the PID as the feedback. The output of the PID was added to all four wheels motion, with the sign of each representing a strafing motion of the mecanum wheels. The resultant control let the robot move in the forward direction while adding the ability to simultaneously strafe to maintain the side wall distance at the setpoint.

Initially, the wall tracking PID exhibited undesirable behaviour if an obstacle came between the wall and the side sensors. The obstacle distance would be read as the new wall distance and the bot would quickly strafe to match the setpoint. This issue was overcome by introducing a section of code that maintained the feedback at the previous value if a jump in sidewall distance occurred. The threshold was adjusted to the radius of the obstacles to ensure effectiveness (Figure 19).

The bearing PID read the current bearing from the MPU and used this value as the feedback. The setpoint was determined based on the current direction using global compass bearings (N, S, E & W) initialised on program start up. The output of the PID was also added to all four motor speeds, with their signs reflecting a rotation of the mecanum wheels. It was a concern that using two PIDs over the four motors would cause a conflict and oscillations or instability would ensue. However, the combination of the two controllers worked exceptionally well with very little tuning. Running the PID controllers in parallel while providing a set forward motion velocity provided robust control as it removed the need for the PID to overcome the deadband evident when driving the wheel servos. Fine increments in the motor speeds had immediate effect on the position and bearing of the robot. The robot was able to be forcibly shifted to any location in the testing area and it would be observed to return to the wall in the correct orientation at the correct bearing. An alternate arrangement was initially tested in a single PID controller was given control of the bearing (turning) of the robot with sidewall distance as the input. While this worked to a reasonable standard, unresolvable oscillations were observed in the robot's motion. This was due to the bearing not having a direct
correlation with the sidewall distance. The PID output could be driven to zero while the robot bearing was not facing in the correct direction.

4.2.2 Strafing
As with the control of the forward motion, the strafing control was implemented using two PID controllers and a preset speed value (Figure 20). The preset speed value was set to drive the robot in the left or right direction. Again a PID controller was given control over the bearing of the robot by measuring its bearing relative to the global target. The forward sensors were directed straight ahead and used as the input to the wall tracking PID controller. The performance of the strafing controller was comparable to that of the forward motion, as the control architecture was very similar. The strafing PID would only operate over short distances, as the motion was not required for covering long sections of the area. An additional routine was added to allow the robot to strafe to the wall on initialisation.

```c
// Update motor speeds
left_front_motor.writeMicroseconds(1500 + lPSpeed - fWdPIDoutput - bearingPIDoutput);
left_rear_motor.writeMicroseconds(1500 + lRSpeed + fWdPIDoutput - bearingPIDoutput);
right_rear_motor.writeMicroseconds(1500 - rPSpeed + fWdPIDoutput - bearingPIDoutput);
right_front_motor.writeMicroseconds(1500 - rFSpeed - fWdPIDoutput - bearingPIDoutput);
```

Figure 20: Multiple PID controllers

4.2.3 Turning
Turning of the robot was implemented using a single PID controller. A target bearing was fed to the controller on calling the turn function. The current bearing was then read from the MPU and passed to the controller as the feedback. The controller would turn the robot to the desired bearing, and then hold its position for a number of milliseconds. This ensured that the function did not flag as complete until the robot had truly stopped moving. Without the implementation of time delay, the robot may pass the target bearing and overshoot, while the state machine begins execution of the next set of commands. During testing, this resulted in unpredictable motion and inaccuracy in the odometry. When testing the code initially, erratic behaviour was observed. It was discovered that the PID was being sent absolute bearing values, causing it to malfunction when the current bearing travelled past 360° and overflowed to 0. This was addressed by implementing code that ensured that the bearing error was always between -180° and 180°.

A sequence was added to the turn function to aid in the alignment of the robot to the wall of the testing area. The robot would rotate clockwise while taking readings from the side sensors. Previous values were stored as each new side reading was read. The current value and the time delayed value were compared and a count incremented if the two values were in a threshold of 1 mm. If enough readings were taken within the threshold the motors would stop (Figure 21). Fairly reliable wall detection was achieved through this method, however braking time resulted in overshoot from the desired position. To address this, a timed pulse of the motors in the opposite direction was delivered immediate following the
stop condition. The result was a moderately reliable wall alignment function. However, this alone was not accurate enough for implementation. In the final wall alignment sequence, this operation was combined with a PID controller that examined the rate of change of the side wall distance in the same way, but was given control over the bearing of the robot as the robot was cyclically driven back and forth. The resultant was repeatable and accurate wall alignment.

```c
// Turn to near wall
if (type == 3) {
    if (abs(currentSideRange - c2Data) > 1 || (currentSideRange > 40) && (nearWall)) {
        left_front_motor.writeMicroseconds(1500 + LSPSpeed);
        left_rear_motor.writeMicroseconds(1500 + LSPSpeed);
        right_front_motor.writeMicroseconds(1500 + RSPSpeed);
        right_rear_motor.writeMicroseconds(1500 + RSPSpeed);
        return TURN_TO_WALL;
    } else {
        rangeCount++;
        if (rangeCount >= 20) {
            nearWall = true;
        }
        if (nearWall) {
            while (!turnCorrected) {
                if (millis() - tPrevious_millis > 10) { // ms per read
                    tPrevious_millis = millis();
                    tCount++;
                } //Serial.println("Reversing");
                left_front_motor.writeMicroseconds(1500 - 200);
                left_rear_motor.writeMicroseconds(1500 - 200);
                right_front_motor.writeMicroseconds(1500 - 200);
                right_rear_motor.writeMicroseconds(1500 - 200);
                if (tCount >= 22) {
                    turnCorrected = true;
                    tCount = 0;
                }
            }
            left_front_motor.writeMicroseconds(1800);
            left_rear_motor.writeMicroseconds(1800); //Original Control for bearing change
            right_rear_motor.writeMicroseconds(1500);
            right_front_motor.writeMicroseconds(1500);
            //delay(2000); // DEBUG - REMOVE <<<<<<<<<<<<<<<<<<<<<<
            nearWall = false;
            runTank3 = false;
            oneRightRunOnce = false; //Reset Setup condition
turnCorrected = false;
        turnToWall = true;
        return STATE_HANDLES;
    }
}
return TURN_TO_WALL;
```
The operation of the turning controller had several inherent issues. The most prominent of which was caused by the deadband in the servos driving the mecanum wheels. Unlike the forward and strafe controllers, which operate by adjusting the ideal motion of the robot, the turn function must drive the robot entirely through the output of its PID. No motion of the motors could be observed up to input values of 65ms, with rapid turning occurring at a value as low as 100. In order to compensate for this, deadband compensation code was included that added a flat value to the motor speeds, depending on the sign of the output of the PID controller. While this improved the performance, there was a limit to the effectiveness of the compensator. As the robot must stop completely at the target bearing, values too close to the edge of the deadband resulted in wheel creep during settling. Simple bang bang control was experimented with as an alternative to the PID control of the turning function. It was found to be inadequate due to the unpredictability of the braking time. Inconsistent results were observed in the time taken for the robot to stop turning, meaning a suitable offset to the target bearing could not be achieved. The integral term of the PID was found to be problematic, due to the difference in time taken to turn between 90° movements and 180° movements. After thorough testing, it was removed and a P controller was implemented.

Jerky motion was sometimes observed as the PID controller attempted to drive the robot the final degree to the target bearing. If the robot became stuck in this state, the MPU was observed to become more susceptible to drift. This was attributed to an increase in vibration and motor electrical noise. Careful tuning of the PID gains and deadband compensation levels removed this behaviour. Poor performance of the turning function had negative implications on both the path the robot takes to complete a run and the accuracy of the localisation. While the control was optimised to an acceptable level, it was decided that an alternate control method may be required to produce more suitable results.

```c
// Deadband compensator
if (bearingPIDoutput > 0) {
    LFSpeed = 35;
    LRSpeed = 35;
    RFSpeed = 35;
    RRSpeed = 35;
} else {
    LFSpeed = -35;
    LRSpeed = -35;
    RFSpeed = -35;
    RRSpeed = -35;
}
```

Figure 22: Deadband compensation code
4.3.0 Sequence Control
Sequencing of the motions of the robot was controlled via a series of state machines. The main sequences of robot’s motion can be broken down into align to wall, initialise, map walls, cover area and avoid obstacle. A graphical representation of the motion sequences is provided in (Figure 23 - Figure 26). The sequences direct the state of the motion handler, resulting in structured, sequential motion.

Figure 23: Robot motion during wall finding
Figure 24: Robot motion during wall mapping
Figure 25: Robot motion during map covering
Figure 26: Robot motion during obstacle avoidance
4.3.1 Align to Wall & Initialise

In the align to wall and initialise state machines, the robot drives to the nearest wall and rotates until it is nearly aligned with the wall. The robot is driven back and forward until correct alignment is confirmed. The robot then strafes to the wall and begins forward motion. The robot will avoid obstacles until a corner is found. Using the IR sensors, the robot will determine whether it is aligned with the short or long edge of the testing area. If it determines it is not correctly aligned, it will drive to the next corner in order to begin wall mapping on the long edge. This results in reduced time taken to cover the entire area (Figure 27).

![Figure 27: Align & initialise FSM](image)

The initial alignment of the robot with the side wall was a key task for proper execution of the program. The global bearing for the robot was set based on this alignment. Any inaccuracies would cause the robots localisation to become less effective. Wall tracking algorithms allowed the robot to follow the wall if misalignment or drift of the MPU occurred. The rotate to wall function aligned the side of the robot to the wall, but not to a high degree of accuracy or repeatability. To ensure correct alignment, the robot was driven back and forth and a PID controller fed a timed delay of the side wall reading. The PID would then try to drive the rate of change to zero. This proved to be an effective method for wall alignment. Many attempts to achieve this were tried and it took the combination of the two methods and a PID controller with rate of change as its input to achieve adequate results.

```c
if ((abs(currentSideRange - t4sr) < 0.01) && (tsr ! 0)) { //Stop condition
    alignCount++;
    if (alignCount >= 50) {
        bearingFound = true;
    }
}
```

![Figure 28: Time delay and stop condition of wall alignment code](image)
4.3.2 Map Walls
After initialisation is complete, the robot will begin a mapping run of the walls (Figure 29). It will trace the perimeter of the area, avoiding any obstacles and sending localisation and mapping data via Bluetooth for display. In this state the robot uses its side IR sensors during forward and turning motion to trace the location of the walls. The forward IR sensors provide mapping of any obstacles encountered.

4.3.3 Cover Area
After mapping of the walls is complete, the robot begins to cover the remainder of the test area (Figure 30). A variety of obstacle avoidance scenarios are accounted for, defining the motion for obstacles at all positions of the test area. The robot continues to cover any uncharted area until the stop condition is met (a limiting wall is found).
4.3.3 Avoid Obstacle

Avoid obstacle may be called from any of the previously defined state machines and handles both the avoidance of the obstacle and the return of the correct operating state. This state may be altered based on the actions of the obstacle avoidance handler. The sequence of operations of the implemented obstacle avoidance FSM is shown in Figure 31. Avoid obstacle ensures that the robot is driven into uncharted territory during each event, preventing the robot from covering parts of the area that have already been explored. It also handles events where walls are encountered while avoiding obstacles and multiple obstacle instances. The sequence includes different reactions for each state from within each individual state machine.

Figure 31: Obstacle avoidance sequence diagram

For the standard obstacle avoidance sequence, the robot would turn 90° toward the direction of uncharted area. This ensured it could move around the obstacle without colliding with a wall or another obstacle. It would then move forward enough distance to clear the detected object. The robot would then turn back to the original bearing, move past the object and strafe back to the original line of travel. If a wall was detected in the first part of the motion, this would signify a final run flag to be set, which would then jump to a state in which it covers the remaining length of the track and stops. If a wall was discovered during the second part of the motion, there were two possibly outcomes. If the robot was performing a mapping run, the robot would turn 90° and continue motion. If the robot was in area coverage mode, the robot would turn 180°, with the obstacle handler returning the robot to the appropriate state to continue motion.
In order to determine whether the object detected by the robot was a wall or obstacle, a function was implemented called isWall(). This function swept the forward sensor across the object ahead, taking accurate readings of distance. Each reading was compared with the last. If the forward value changed greater than the threshold value, the function would register the object as an obstacle. The calibrated servo angle was used with a cosine function to ensure the read distance was always relative to the front edge of the robot (Figure 32).

```c
// Register read, triangulate and return forward distance
if (readReady) {
    if (servoPos >= (70 - leftMax)) {
        servoPos = servoPos - 10;
    } else {
        sensorServoPosition(70);
        complete = true;
    }

    // Update global variable
    currentForwardServoPos = servoPos;
    calibratedAngle = (abs(servoPos - 70) * 0.0727) * 3.14159265359/180; // Calibrated angle in radians
    previousForwardRange = currentForwardRange;
    currentForwardRange = currentSensorRange * cos(calibratedAngle); // Forward distance from bot frame
    if (abs(previousForwardRange - currentForwardRange) > 5) { // threshold value
        isWall = false;
    }
}
readReady = false;
```

Figure 32: Section of isWall() code
4.4.0 Localisation and Mapping

Both localisation and mapping were conducted on board the robot using the Arduino Mega 2560. The calculated values for robot position and wall data were sent periodically to a PC for display via a Bluetooth module. The data was then processed and output into an easy to view format. All mapping data was generated entirely by the sensors on the robot. No shape of the testing area was assumed.

4.4.1 Localisation

The localisation takes advantage of the dead reckoning system put in place. This uses the data that is output by the dead reckoning equations found in section 4.1.4 and converts it into an X and Y coordinate system that takes the robots bearing into account through the MPU. Each time the distance travelled is updated using the wheel odometry, the current bearing of the robot is computed relative to north. This distance is then added or subtracted from the previous value, based on the current bearing of the robot. Such operations occurred every 10-15ms. This provides a constant reference to the position and path travelled by the robot. Once the walls have been detected the IR sensor is cross referenced to check that the robot's positioning is the correct distance from the wall once it enters its sweeping phase. All of this data is stored and sent to the computer via Bluetooth where it is used to map the robot's path and progress.

A solution was implemented that used both the IRs and the wheel servos to compute odometry. The forward distance at the beginning of any given motion was compared with the forward distance at the end of the motion. If the readings from the IR values were within a suitable range (10cm accuracy), the IR and servo distances were averaged and the new bot position updated in the localisation data. The position of the robot was updated approximately once every 10-15ms. This new position of the robot was sent to a computer for display once an X or Y coordinate had incremented by a centimetre or more. This reduced that amount of data that needed to be sent and resulted in a more responsive mapping program.

4.4.2 Mapping

While localisation data was being sent via Bluetooth, mapping information was sent concurrently. All mapping data was generated by using a combination of the four IR sensors. This was achieved by considering the orientation of the sensors being read relative to the bot frame, then the position of the bot frame relative to the North bearing (Y axis). As one of the sensor clusters was attached to a servo, some careful trigonometry was required to ensure a vector relative to North within a range of 0 to 360°. This vector value was sent along with a distance to the detected wall or object and the localisation data for mapping (Figure 33).
4.4.2.1 Data Acquisition

On the Arduino side the X and Y coordinates of the robot in centimetres, as well as the bearing in degrees and distance of objects seen with the IR sensor in centimetres are sent via Bluetooth iteratively. The Arduino prints a ‘$’ symbol with the println function to send with a carriage return and newline. The X, Y, bearing and distances are sent as signed integers using the same command individually. A ‘%’ Symbol is then sent to show that all the data has been sent. On the C# side, the main program thread loops checking if any data is available in the serial buffer. If it is it compares the first line to ‘$’, if it’s equal it reads the next 4 lines and takes the X, Y, bearing and distance values into their own variable and prints a block of red where the robot's position is, the location of the object seen is calculated using the robot's position the bearing of the sensor used and the distance away to form a smaller black block where the object is, be it wall or obstacle. If any problems occur in sending data the container symbols should throw a Read Unsuccessful error in the text terminal and not use any read data as it may be wrong. Any lines printed to the Bluetooth that are not of this format are printed straight to the terminal as plain text.

Figure 33: Robot map in progress
If the location of the robot is within 3 centimetres in x or y, the position is not printed, this means that during turning the robot doesn’t flood the Bluetooth with overlapping positions that need to be printed. When an object isn’t visible to the IR sensors a distance and bearing of 0 is sent, so any distance of 0 is ignored on the c# side and no obstacle is drawn. Every time the map is edited with a location box the latest location of the robot is displayed as the chosen sprite on the map. This way you can see the current location of the robot in real time. When printing to the picture box, the map bitmap is cloned as the refreshing of the picture box accesses the bitmap. Because the editing of the bitmap is continues this throws errors without the clone function inbuilt. The second thread on top of the forms thread means that a reading and writing of data from the serial into the bitmaps are not interrupted by the continuous refreshing of the forms components and vice versa. When the form is closed, the program thread finishes working and the serial port is then closed. Then the application closes.

4.4.2.2 Graphics
The data is received by a program named 'SLAM Terminal' where it uses coding done in Visual C# to take in the given data and transforms it into a map. The data that the program receives via Bluetooth is a combination of X and Y coordinates as well as data from the IR sensors and MCU to allow for object placement and wall placement. The programming initially starts up with a settings box.

![Figure 34: Serial tab](image-url)
Under the serial tab there are a number of settings associated with the serial connection of the program. The top textbox allows the user to view all available com ports and select the one associated with the Bluetooth connection between the PC and the robot, with a ‘Set Port’ button to confirm the selection (Figure 34). A ‘Refresh’ button is also clickable to repopulate the list with any new serial ports available. The user also selects the Baud Rate of the connection which has been set up on the Arduino at a value of 115200, so the programming defaults to that. A button is also present to confirm this selection if it’s changed. A check box for choosing if ‘Data Terminal Ready’ is used in the communication is available. Once a port and Baud Rate have been set, a label will show this selection at the bottom of the panel and the ‘Continue’ button will enable (Figure 35).

On the mapping tab of the settings a number of graphical options are available to the user. All of these have a text box or drop down box for setting the variable, and an associated button for setting the value. ‘Height’ and ‘Width’ setup the map in centimetres, while ‘Map Offset’ sets additional space around the map also in centimetres. ‘Bot Size’ indicates the width and height of the robot in centimetres. The ‘Grid Width’ and ‘Grid Height’ sets the width and height of a single square on the background grid respectively in centimetres. The ‘Grid Colour’ lets the user select from all system named colours to choose the grid colour on the map. The ‘Bot Sprite’ lets the user select from a list of available bitmap sprites for the representation of the robot on the map. When the Continue button is clicked, all the settings for the serial connection and map attributes are passed through to the next form and the settings box is closed.
The main program form is displayed in the centre of the screen showing a large picture box, a multiline rich text box on the right, and a few descriptors and settings along the bottom (Figure 36). As the form loads, the map attributes set from the previous form set the height and width of a bitmap image in white, plus a scaling factor of 3 so that every 3 pixels of the bitmap is one centimetre of the map. The sprite bitmap is also drawn in from a file on the PC. The map is then displayed on the picture box. The grid then populates slowly to emulate an animation with vertical lines first then horizontal, it does this by drawing to the picture box iteratively as it’s produced on the bitmap. The grid scale is shown in the bottom left as plain text based on the grid settings. A save map button can be clicked to save the contents of the picture box as a jpeg file with a timestamp appended to it. All files are saved to C:\SLAM folder and this is where all Bot Sprites are found.

The scrolling multiline textbox displays ASCII art of ‘SLAM PROJECT’ and then the program opens the com port and begins a loop of listening for incoming bytes. The information as it sets up is displayed in the textbox line by line. An ‘Auto scroll’ check box can be checked or unchecked to set or stop the textbox from moving to the bottom whenever a new line is added to it.
5.0 Testing
Testing began at the initial project meeting and was a continuous process throughout the entire design. As an initial task, interfacing with each sensor was performed, its performance recorded and a short interface control document (ICD) drafted to allow the team to have quick access to using any sensor while programming. The testing of each component was critical to the robot’s ability to navigate and manoeuvre, and drove the formulation of our overall solution. During each stage of the process of developing the robot, the performance of individual modules was evaluated. On satisfactory performance, the modules were integrated with the overall system and additional testing performed to confirm adequate performance levels.

5.1 Infrared Sensors
For the infrared sensors they each were run through the testing code given; this took readings directly from the sensors chip, incremented them 30 times and averaged the result giving a distance. Each infrared sensor was tested for its type, functionality, working range and accuracy.

As the design of the robot progressed, the performance requirements of the infrared sensors increased. Initially they were to be used solely for wall detection, which involved a simple distance check without a high degree of accuracy, a common use for the sensor type. The sensors were then used to keep track of walls, meaning data was stored and compared. The IR sensors were calibrated by recording the raw data from the sensors and graphing the results against actual distance. A curve was then fitted to the data for each sensor type, allowing for more accurate distance readings.

As the sonar sensor was removed from the system, the infrared sensors were then required to handle obstacle detection. To achieve this, a pair of the IR sensors were swept in front of the robots paths using a servo motor. It was observed that the sensors were not able to take in data quick enough, leading to inaccurate results. This was optimized by slowing the servo down, still allowing for enough speed to pick up and obstacle, which greatly improved the performance. During testing it was noted that the IR accuracy was a time driven function. The faster you moved, the lower the accuracy of the reads. Slowing the robot down gave more accurate readings but was undesirable for the robot's performance. Different levels of speed and accuracy were implemented depending on the requirements of the task at hand in order to yield the best results.

It was realized that the sensors would interfere with each other as they were in very close proximity. A shield was designed to reduce the level of interference, they can be seen in Figure 37.
5.2 Sonar Sensor

The second sensor to be tested was the sonar, code was written to enable for raw data reading and converting it into a distance. The data was very jumpy and more coding was done to try and correct this, it was found that the sonar had a very slow report rate. In the end the sonar was deemed to be unsuitable for the task and was replaced by two infrared sensors which have greater accuracy and a higher report rate.

5.3 MPU

Due to the number of sensors within the MPU the testing task was significantly harder and more fraught. The sensor is entirely digital, with all data being either read or written via I2C. The first step in testing the MPU was to ensure the Arduino was correctly communicating with the chip. A simple I2C scanning code was loaded onto the Arduino that sent address numbers along the data line and waited for an acknowledgment from the chip. Due to the hot-swappable nature of I2C the data and clock lines were found to be switch. These were corrected and then an acknowledge signal was returned by the MPU. Once correct communication was established the more specific methods of accessing the required data streams from the sensor and programming the digital motion processor was undertaken. This required sending a data request as a command addressed to the sensor, with a specific register address to read or write to. This data was then returned by the IMU and interpreted by the Arduino. The Accelerometers, Rate Gyros and Magnetometers are all read from different registers. The DMP includes an interrupt output that was connected to the Arduino to trigger an interrupt whenever the FIFO buffer for the fused data was available to be read. Due to the data fusion algorithm contained in the DMP the current and previous output influenced the following result. Due to the high rate of heading information being calculated 200 heading calculations per second, often the Arduino would be too busy to access the data before in the register before it was overwritten by a new result. Thus being
able to have the DMP store the output in a running buffer enabled the Arduino to only poll the data when needed as the DMP stored the previous states needed for the fusion algorithm. The ability to offload the processing of the data fusion is thus a significant advantage of an IMU with a digital motion processor compared to discrete sensors and undertaking the fusion calculations on the primary microprocessor. While the rate-gyro sensors were simple to calibrate due to the ability to simply hold the IMU still, the accelerometers were more difficult, and the magnetometers impossible.

### 5.4 Bluetooth

Bluetooth was then tested by connecting the given Bluetooth module to the PC, the robot was powered on and found quickly by the PC where it was connected. The robot was then able to send and receive data through the Bluetooth and was given a code to allow for the control of the servo motors. The Bluetooth was the main way to test the servo motors during the dead reckoning process and was used to receive data directly from the robot as well as transmit starting acknowledgements.

When trying to access the Bluetooth through the original platform of the mapping program a number of problems occurred. This was due primarily to Matlab software on the university computers not having the serial communication toolbox installed. While the toolbox was not installed, a handful of the functions were still available for use. This gave the appearance of full functionality, and significant time was spent attempting to debug the unresolvable connection issues.

To resolve the issues with Matlab, Visual C# was used. The Bluetooth COM port was able to be accessed and read from using C# IO functions. After spending time converting the original Matlab program to Visual C# code, more Bluetooth connection issues were encountered. This was identified as an administration privilege issue, in which the user was unable to disconnect and reconnect a device from within the Windows operating system. The result was that the user would be locked out of using Bluetooth intermittently. To overcome this, running of a personal laptop was adopted as practice for the running of the mapping software. It is strongly recommended that Bluetooth privileges be adjusted before next year’s project to avoid a repeat of the same issue.

### 5.5 Mapping program

Originally designed on MATLAB the mapping program ran in to the above problems about Bluetooth, and was moved to Visual C#. The use of serial connection between the Arduino and Visual C# had a lot of testing and research around it in terms of correct terminator use for sending data via Bluetooth. The serial.readline function was used to read up to a newline terminator of the Arduino’s Serial1.println function. The testing of code for the mapping program was a continuous activity as a way of troubleshooting bugs and testing if the correct outputs are done. Bugs were reported in map size, program stability, starting position, scale and consistency and were systematically logged, addressed and resolved.
5.6 Servo Motors and Mecanum Wheels
The range of speed inputs to the servo motors and the consistency of motion were measured. The provided testing code was placed onto the Arduino where certain key presses were able to control the direction of the robot and the speed given to each motor. It was observed that the robot did not function well under a speed of 70ms deviation. One of the servo motors was found to be sticky. This was found to be an issue with the drive shaft collar rubbing on the robot frame. The robot’s speed would increase up to an increment of 300ms where the increase in input would not change the robot’s speed. This code was initially used to gather information for the dead reckoning system where the robot went through a series of tests at certain speeds to give the velocity at each point.

5.7 Assembled Performance
Details of the assembled robot are provided in Figure 38.

After the individual performance of the parts required to build the autonomous robot system were tested and calibrated, the performance of the full system could be tested and evaluated. This formed the majority of the testing time. Control schemes and individual functions implemented in standalone code were found to perform differently when integrated with a larger system. This was due primarily to increases in processing time brought on by heavier programming. Optimisation was implemented to reduce that amount of time taken for the robot to complete a full cycle. Serial prints were found to contribute heavily to processing time and were removed wherever possible. Some functions had to be adjusted to account for the increases in programming time. This was particularly true for the stopping distance and sweeping rate of the sensor servo. Optimising the time of the
execution cycle had unexpected negative effects as well. The Arduino function millis() which
was used for time approximation was found to be very inaccurate if called too quickly. This
was attributed to accumulated rounding error and had a profound effect on the accuracy of
the odometry. After the speed of execution was optimised, additional delays had to be
reintroduced into the program to ensure that the millis() function returned usable results. A
superior alternative would have been to implement the micros() function, however this
behaviour was not noticed until late in the project and finding the source of the bug took
considerable time.

Each state machine was tested thoroughly on implementation. This was an iterative process
which involved interrupting the bot with as many simulated scenarios as possible and
observing the response. While the flow of operations of each state machine was fairly
straight forward, the use of global bearing targets resulted in many different iteration of
each movement function at each stage of motion. While using the global bearings added
greatly to the complexity of the state machines, it was a challenge that was worthwhile as
cumulative error in the robot bearing was eliminated entirely.

The robot was tested on the constructed arena as much as possible. This was necessary as it
was observed that different environmental factors such as ground material, lighting and wall
material and colour had a significant impact on the response of the robot. A particular
problem observed was nuisance tripping of IR based routines. This was observed to happen
repeatedly in certain places on the testing arena. It was hypothesised that a number of
environmental factors could have been causing the tripping to occur. Sources identified
were the fluorescent lighting, which contain an element of IR light, the IR light on the testing
camera, the material of the arena walls and problems in the developed code. To compensate
for this, changes were made to the physical structure of the robot and within the
programming to reduce the probability of erroneous data.

5.8 Obstacle Detection
During tested it was observed that the performance of the obstacle detection functions
were tightly coupled with the wall detection routines. If the detection routines were
optimised to detect all obstacles at any given position, the performance of wall detection
was negatively affected and vice versa. This was due to the obstacles being of smaller size
than the wall. The probability of detecting the obstacle as soon as it came into range of the
robot’s sensor threshold was lower. If the range was optimised to suit the obstacles, the
robot would stop too far away from a wall. To resolve this problem, a balance between wall
and obstacle reaction time was found. It proved to be effective, but not the most robust
solution. Ideally the robot would have two separate sequences for obstacles and mapping
and drive closer once the detected object was identified as a wall. This was not implemented
due to the additional complexity required to alter the state machine. The robot worked well
in a balanced operation mode.
6.0 Results

The robot was proven to have the capability of covering the total area and producing an accurate map of the surroundings. The best evaluation run was completed in 3 minutes and 32 seconds. The output map of the evaluation run can be seen in Figure 39.

During the evaluation run, the robot collided with one obstacle. It was a very minor collision however, as the edge of a mecanum wheel gently grazed the edge of an obstacle. A small section of the map was missed due to the MPU not being aligned to 100% accuracy. The robot had been proven to work well prior to evaluation and this was attributed to an unexpected error. The robot dodged and mapped all obstacles, coming to a complete stop at the end of the run without operator intervention.
7.0 Discussion

The robot was proven to work well during testing. Once placed in a random location, the robot was able to find a wall, align to that wall, find a corner and begin sweeping and mapping of the room. The localisation data was proven to be reliable, provided the robot was driven within the calibration parameters. The mapping data output gave an accurate representation of the testing area and correctly identified objects. During the evaluation run, the bearing was not aligned correctly, causing the robot to miss coverage of a small sliver of the testing area. A minor interference with an obstacle was also observed. Overall the device performed well on the day.

7.1 Successes

Over the short term the IMU performed exceptionally well, with very accurate and repeatable headings. This was instrumental in allowing precise control over the robot's heading and accurate implementation of SLAM. Gathering map data using primarily the short range IR sensors was a good tactic, as the produced map was accurate and repeatable. Very few errors could be seen in the final result. Only mapping the environment as necessary helped to produce a map that had fewer erroneous values. Sending the mapping data only as required gave a responsive map of the robot in real time and improved the execution time of the robot’s programming sequence.
7.2 Limitations
There were a number of limitations to the design. The control over the motion of the robot was implemented using a rigidly defined state machine. The state machine was effective at ensuring that all of the target area was covered. However, as the motion was very structured and rigid, the speed was reduced. Making changes became more difficult as the state machine grew in size and complexity. The motion of the robot was kept to simple forward, turn and strafe actions. This was in order to accurately implement localisation from the wheel odometry. However, this prevented the full capability of the platform from being realised and dramatically reduced the speed of operation.

The speed at which the front of the robot would be scanner for obstacles was also a major limiting factor. This was due to the choice of the narrower beam IR sensors as the forward facing detectors. Initially, the sonar was excluded from the build as it was deemed to inaccurate. However, late into implementation it was realised that the broad beam of the sonar may have been used to detect obstacles more easily. Using only the IR sensors resulted in a large swing to be required to cover the full area in front of the bot. The increase in time required to take these readings slowed down the maximum speed of the robot and introduced additional risk of not detecting obstacles fast enough.

7.3 Future Improvements
If a new robot were to be rebuilt, the greatest addition would be an optical sensor for calculating position (localisation). This would immediately allow the robot much more freedom of movement and allow more complex motion and avoidance routines. If a new control scheme were to be implemented, a more flexible solution would likely produce better results. A map driven control system could provide a faster response, allowing for greater freedom of movement and shorter completion time.

In the future the addition of an accurate external referencing bearing would be advantageous. While the magnetometer in the specific IMU we were allocated was phenomenally unstable and temperamental, the version of the chip that is stand-alone is significantly more robust. Regardless of the specific method employed, the ability for the robot to know the true bearing it is currently tracking provides a more secure and drift-proof method of remaining on the correct path and thus more comprehensively transiting the entire area.
8.0 Conclusions
A fully functioning autonomous SLAM robot was successfully designed, built and programmed. The robot was capable of deciding its path of movement and navigating through a simulated room with obstacles. The robot was able to simultaneously build a map of the environment while tracking its position. With the application of room cleaning in mind, complete coverage of the area was successfully conducted and the operation of the robot optimised to reduce the time taken to complete its task. Evaluation of the robot's performance provided evidence of satisfactory coverage, speed and obstacle avoidance. The map constructed was an accurate representation of the testing environment.

In order to achieve fully autonomous SLAM, IR sensors, servo motors and a motion processing unit were integrated with an Arduino controller on a mobile platform. The project provided a practical application of motion actuation, sensing and signal processing while combing mechanical, electrical and software design principles.

The design was successful in meeting the requirements of the project during testing. A perfect run was not achieved during evaluation, with a slight interference with an obstacle being observed along with a small section of the track being missed. These impacts on the performance were very minor and overall the device performed well. Limitations of the robot were in the limited localisation provided by using wheel odometry and the rigidity of the designed state machine. Recommendations for future improvements are to adopt a map driven decision making algorithm and to implement an optical sensor for localisation. A stable reference bearing may also allow the device to run for a longer period by compensating for gyroscopic drift.
9.0 References


10.0 Appendices

Appendix A: Bill of Materials

<table>
<thead>
<tr>
<th>Description</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mecanum wheeled robot chassis</td>
<td>1</td>
</tr>
<tr>
<td>Short range IR sensor - GPD2120XJ00F</td>
<td>2</td>
</tr>
<tr>
<td>Long range IR sensor - GP2Y0A02</td>
<td>2</td>
</tr>
<tr>
<td>Bluetooth module</td>
<td>1</td>
</tr>
<tr>
<td>Servo motor</td>
<td>1</td>
</tr>
<tr>
<td>MPU 9150</td>
<td>1</td>
</tr>
<tr>
<td>Spiral cable wrap - 1416087</td>
<td>1m</td>
</tr>
<tr>
<td>3300μF Capacitor</td>
<td>1</td>
</tr>
<tr>
<td>Level shifter - Logic_Level_Bidirectional</td>
<td>1</td>
</tr>
<tr>
<td>Front IR housing</td>
<td>1</td>
</tr>
<tr>
<td>Side IR housing</td>
<td>2</td>
</tr>
<tr>
<td>MPU Stand</td>
<td>1</td>
</tr>
</tbody>
</table>
Appendix B: Servo Calibration Data

Figure 41: Bot speed vs. motor speed - forward motion

\[ y = 3\times10^{-8}x^3 - 2\times10^{-5}x^2 + 0.0059x - 0.2549 \]

![Graph showing bot speed vs. motor speed for forward motion](image)

Figure 42: Bot speed vs. motor speed - reverse motion

\[ y = 3\times10^{-8}x^3 - 3\times10^{-5}x^2 + 0.0065x - 0.2985 \]

![Graph showing bot speed vs. motor speed for reverse motion](image)
Figure 43: Bot speed vs. motor speed - left strafe

$y = 3E^{-0.08x^3} - 2E^{-0.05x^2} + 0.0055x - 0.2847$

Figure 44: Bot speed vs. motor speed - right strafe

$y = 4E^{-0.08x^3} - 3E^{-0.05x^2} + 0.0062x - 0.3129$
Figure 45: Front IR housing mechanical drawing
Figure 46: Side IR single housing mechanical drawing
Figure 47: MPU stand mechanical drawing
Appendix D: Hardware Schematics

Figure 48: Level shifter schematic

Figure 49: IMU block diagram