Abstract

SLAM (simultaneous localisation and mapping) involves the gathering of information in real time to navigate through an unknown environment whilst simultaneously building a model consisting of features within the environment. The localisation process comprises of determining robot position and orientation relative to environmental landmarks, whereas the process of mapping involves determining the position of landmarks relative to the robot.

Our aim during this project was to design and develop a prototype SLAM robot for the purposes of floor vacuum cleaning. The robot is designed to meet set number of specifications; these include wall sensing, object sensing, position sensing and orientation sensing. These parameters are then used to determine movement actions undertaken by the robot and generate a map of the robot’s path and features it has detected.

The main requirements were to maximise floor coverage whilst trying to minimise the time taken to achieve said coverage. Also the robot had to be fully autonomous and capable of avoiding obstacles within the environment.

In order to accomplish SLAM tasks, multiple sensory elements are required to determine the geometry of the environment and to track the changes in the robots position and orientation as it moves. Our robot utilizes a ball mouse with optical encoders to determine absolute position and an inertial measurement unit (IMU) to determine robot orientation. Ultrasonic and infrared displacement sensors are used by the robot to measure the distance of environmental features from the robot.

Some form of intelligence is also required to determine actions based on these sensor observations. Our system intelligence is implemented by an Arduino Uno development board, which runs software written in Wiring programming language. Data is transmitted wirelessly from using a Bluetooth module to perform real time graphical mapping via Processing (a graphical platform external of Arduino).

A prototype of our robot was assembled and tested, completing the task successfully on a 1.5m x 2.2m walled platform containing three randomly placed obstacles, giving consistent results for coverage in the range of 80 – 90 percent and times under 3 minutes.
Table of Contents
Abstract ................................................................................................................................. ii
Table of Figures ................................................................................................................... 2
1 Introduction ...................................................................................................................... 3
2 Problem description ......................................................................................................... 3
  2.1 Requirements .............................................................................................................. 3
3 Specifications ................................................................................................................... 3
  3.1 Performance Features ............................................................................................... 3
4 System design and development .................................................................................... 5
  4.1 Design Philosophy ..................................................................................................... 5
  4.2 DFRobotShop rover V2 and motor driver shield ...................................................... 6
  4.3 Gearbox ...................................................................................................................... 6
  4.4 Motors ......................................................................................................................... 6
  4.5 Mecanum Wheels ...................................................................................................... 6
  4.6 Microcontroller .......................................................................................................... 6
  4.7 External Clock ............................................................................................................ 7
  4.8 Schmitt Trigger .......................................................................................................... 7
  4.9 Multiplexer ................................................................................................................ 7
  4.10 Power Supply ........................................................................................................... 8
  4.11 Sensor Placement .................................................................................................... 8
  4.12 Sensor Filtering ....................................................................................................... 8
  4.13 Optimum Path Calculation ..................................................................................... 9
  4.14 Localisation ............................................................................................................ 10
  4.15 Position Tracking .................................................................................................... 11
  4.16 Orientation Tracking .............................................................................................. 12
  4.17 Data Processing and Mapping ............................................................................... 13
  4.18 Software .................................................................................................................. 14
    4.18.1 Line Following .................................................................................................... 14
    4.18.2 Wall and Corner Detection ............................................................................... 15
    4.18.3 Obstacle Detection and Avoidance .................................................................... 16
    4.18.4 System Routine ................................................................................................. 17
    4.18.5 Start-up Algorithm .......................................................................................... 18
    4.18.6 Traverse and Object Avoidance Algorithm ..................................................... 19
Table of Figures

Figure 1: Key features of robot ................................................................. 5
Figure 2: Arduino Uno pin configuration .................................................. 6
Figure 3: External 10Hz Clock ................................................................. 7
Figure 4: Schmitt trigger configuration .................................................... 7
Figure 5: Multiplexer configuration .......................................................... 7
Figure 6: Sensor placement and direction on robot ................................... 8
Figure 7: RC low pass filter ...................................................................... 8
Figure 8: Possible motion path algorithms ............................................... 9
Figure 9: Coordinate Systems ................................................................. 10
Figure 10: Ball mouse with custom encoders .......................................... 11
Figure 11: Position tracking development .............................................. 11
Figure 12: Orientation development process .......................................... 12
Figure 13: Mapping robot position and orientation ................................... 13
Figure 14: Map showing line following operating .................................... 14
Figure 15: Corner detection data ............................................................ 15
Figure 16: Main program finite state diagram ......................................... 17
Figure 17: Start-up routine flow diagram ............................................... 18
Figure 18: Traverse and object avoidance flow diagram .......................... 19
Figure 19: Displacement sensor characteristics ....................................... 20
Figure 20: Closed loop path following performance comparison ............. 21
1 Introduction
Vacuum cleaning is a domestic task carried out to remove unwanted particulates from floor surfaces. Robotic vacuum cleaners perform this task autonomously. In order to do this they require intelligent programming and utilize various combinations of cleaning systems such as mopping, UV sterilization and suction. Fully automated robotic vacuum cleaning has been around since the late 1990s, with the first production model released by Electrolux in 1997. However popularity with consumers was limited due to their high price.

The most successfully marketed vacuuming robot to date is the Roomba, a product form the American advanced technology company iRobot. Roombas rely on a set of simple algorithms to randomly change path and navigate around a room. Objects are detected using force sensors imbedded in the robot's bumper, this way the robot must collide with the object to observe its presence. Although this is effective at cleaning a room is not efficient as they take far longer to do the task as a person would. This is due to the fact that it crosses its previous path multiple times. In order to improve this efficiency the robot could perform mapping of the room they are cleaning. This way more sophisticated control could be implemented to ensure the robot does not cover ground that has already been covered and remember the location of obstacles.

2 Problem description
The objective of this project was to develop a small autonomous SLAM robot dedicated to the purpose of interior room vacuum cleaning. The test environment consisted of a platform with boundary walls. Three cylindrical obstacles of similar in size to cans and bottles were to be placed at random positions upon the platform.

2.1 Requirements
- Robot is mobile and autonomous.
- Floor coverage is maximised.
- Total run duration is minimised.
- Collision with obstacles and boundary walls is avoided.
- Robot records and maps the path it has travelled and objects and walls detected.

3 Specifications

3.1 Performance Features
There are a number of innovative features built in to the robot architecture in order to enhance the effectiveness of the SLAM process.

Wall Detection

When the robot is turned on, it spins around 360° and takes a number of sample points from the laser sensor. Using these points, a regression technique groups the points and determines a line of best fit connecting them. The technique is then run again on the generated lines, checking if any of them have the
same gradient. If so a wall has been found. To detect walls while moving we sample twice, once at the heading position and once at an offset to the right. If an object is detected in both of these positions, then the obstacle must be a wall.

**Corner Detection**

To detect corners we spin a complete revolution and use a regression technique similar to that used in wall detection. This time instead of looking for lines with similar gradient it looks for lines that intersect at an angle of approximately 90°.

**Real-time Mapping**

Real-time mapping is undertaken by the robot to track its path as well as plot detected walls, corners and objects. Map data generated by the robot is sent over Bluetooth and displayed graphically on a computer via a Processing sketch.

**Controlled Movement**

All movements used PID controllers on both heading and distance to ensure accurate output with controllers tuned to achieve minimum overshoot and a fast response time. The active braking is implemented to stop movement by powering the motors in reverse for a fraction of a second to stop sharply.

**Wall Alignment**

The robot aligns to wall using the two left facing IR sensors and pivots over the front wheels to ensure correct alignment. The alignment occurs when both sensors are below 4cm from the edge of the car.
4 System design and development

4.1 Design Philosophy
Importance is placed on creating a robust system both in the area of hardware and software. Our robot is designed to possess sufficient functionality to achieve two-dimensional SLAM tasks in an effective and efficient manner. There is also a basic level of redundancy built into the system to enable adaptability to changes in the operating scenario. The summary of the key system features is shown in Figure 1.

Figure 1: Key features of robot
4.2 DFRobotShop rover V2 and motor driver shield
The robot is constructed upon a DFRobotShop Rover V2 base chassis and circuit board kit. This has an on board voltage regulator circuit and pin breakouts designed for the Atmega 328P microcontroller. There is a L298P dual H-bridge motor driver on the base board and another located on the motor driver shield these are controlled using a pulse width modulated (PWM) signal from the microprocessor.

4.3 Gearbox
The Tamiya twin motor gearboxes were able to be assembled in either a high speed or high torque configuration. In the early stages of the design the high speed configuration was used. This configuration proved to be unsatisfactory in producing precise movements at low speed and eventually was discarded in favour of the high torque 203:1 ratio. The gearbox was custom mounted on the rover in the inverse direction than intended. This allowed a shorter wheelbase to be achieved, the speed of rotational movements and reducing the amount of slip between the wheels and floor surface.

4.4 Motors
Each side of each gearbox is driven by a 5V DC motor. The speed and direction of rotation of these motors are actuated by pulse width modulated (PWM) signals sent through the microprocessor controlled H-bridges.

4.5 Mecanum Wheels
The Mecanum wheels attached to the each axle in an X configuration allows for omnidirectional movement.

4.6 Microcontroller
The robot is controlled using an Atmega 328P Microcontroller, running on the Arduino development platform. This programming environment was simple to use and was familiar to all team members. A challenge presented with using this microcontroller was the limited number of pins available. Access to timer interrupt pins was unavailable as these were already reserved for the PWM inputs of the motor drivers.

The final pin configuration is shown in Figure 2.

Figure 2: Arduino Uno pin configuration
4.7 External Clock
A solution to the unavailable timer interrupt pins was created by using an external clock; this allowed us to create a regular interrupt to perform tasks at set intervals. The custom timer circuit was made to generate a 10Hz clock signal. This circuit consisted of a NE555 IC configured as an astable multi-vibrator. The schematic of the clock is shown in Figure 3.

![Figure 3: External 10Hz Clock](image)

4.8 Schmitt Trigger
The analog outputs from the position encoders are run through a HEF40106BP a Schmitt Trigger. The Schmitt Triggers toggle high when the encoder output drops below 2.2V and go low when the output rises above 3.0V. This helps to eliminate the undetermined states when the encoder is on the edge of a state change. The configuration of the Schmitt trigger is shown in Figure 4.

![Figure 4: Schmitt trigger configuration](image)

4.9 Multiplexer
Due to the limited number of analog input pins on the Atmega 328P microprocessor we chose to run the displacement sensors through a MC14051bpp multiplexer IC. This reduced the required number of input pins from four to three, freeing up an extra pin for other applications. The configuration of the multiplexer is shown is Figure 5.

![Figure 5: Multiplexer configuration](image)
4.10 Power Supply
The robot is powered by five Sanyo Eneloop nickel metal hydride batteries. In total 6V and 2000mAh is provided to the regulator on the rover circuit board.

4.11 Sensor Placement

The sensors are arranged to get a broad coverage around the robot. Ultrasonic sensors are positioned at the front and back of the robot to detect walls and objects in the path of movement. The ultrasonic sensors have a dead band within the first 15cm of range. To compensate for this the sensors are offset on the robot so the usable range begins at the robot perimeter.

Long range infrared sensors are position on either side of the robot. These are used for precise distance measurements during the start-up line detection routine and to measure distances from the wall. The dead band of these sensors is approximately 20cm, thus they have been offset to reduce the chance of objects being.

Medium range infrared sensors are also mounted on the each side. They are used to in coalition with the long range sensor on each side to align parallel detected walls.

4.12 Sensor Filtering

Testing of the displacement sensors revealed that the signal contained a lot of high frequency noise. To reduce this noise the sensors were filtered in both hardware and software. A low pass RC filter was made and connected between the output of the sensor and the input to the Arduino as shown in Figure. By choosing a low value for the resistor so as not to decrease sensitivity and a large value for the capacitor using Equation 1 we were able to set a low cut-off frequency. Filtering was implemented in software using a rolling circular buffer filter. This averaged a set number of past readings with the current reading. This smoothed the data and eliminated any outlying spikes in the data. It was also more efficient than other low pass filter algorithms as a filtered result was produced for each sample taken, instead of every n number of samples.

\[ f_c = \frac{1}{2\pi RC} \]  

(1)

Figure 6: Sensor placement and direction on robot

Figure 7: RC low pass filter
### 4.13 Optimum Path Calculation

An important consideration for smart devices such as robotic vacuum cleaners is to cover as much area as quickly and efficiently as possible. For centuries, farmers have ploughed their fields in regular straight rows as it is regarded as the fastest way to cover a given area.

The traverse function achieves full coverage of the environment by making a series of parallel sweeps, and avoids turning. This algorithm is considerably more efficient than other algorithms such as spiral motion show in Figure 8 (A). By comparing the effective overall lengths of paths using these two algorithms, it is clear that the proposed motion algorithm offers significant time savings.

Figure 8 (B) illustrates the optimum path achieved using the algorithm discussed above.

![Figure 8: Possible motion path algorithms](image)

The total path length, $S_{1\text{tot}}$, can be found by summing the x and y components of its motion, where:

$$ S_{1x} = (w - b) $$

$$ S_{1y} = \left(\frac{w}{b}\right) (l - a) $$

Substituting in the dimensions of the table and robot respectively we can determine $S_{1\text{tot}}$:

$$ S_{1\text{tot}} = 11.46m $$

Figure Y shows the path traced out using the spiral motion algorithm. The x and y components can be found as follows:

$$ S_{2x} = \sum_{i=0}^{i_{max}} 2(w - ib - b) $$

$$ S_{2y} = \sum_{j=0}^{j_{max}} 2(l - ja - a) $$

Where $i$ and $j$ are iterators denoting the number of full loops the robot has made. Solving for total distance travelled:

$$ S_{2\text{tot}} = 17.96m $$

This represents an overall time saving of 36%
4.14 Localisation

Two coordinate systems are used to track the orientation and position of the robot as shown in Figure 9. The origin of the global coordinate system is set after the start routine when the robot has positioned itself in a corner. The origin of the local coordinate system is aligned with the IMU which itself is positioned in approximately the centre of the robot. The heading or yaw of the robot is indicated in the figure as gamma (γ), the rotation angle offset from the global coordinates.

A rotation matrix (Equation 3) is used to transform the data collected in reference from the sensors coordinate system, to data in reference to the local coordinate system.

$$^A P = R^B P$$ (2)

$$\begin{bmatrix} ^A P \end{bmatrix} = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & A_{X_B \text{ org}} \\ \sin(\gamma) & \cos(\gamma) & A_{Y_B \text{ org}} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} ^B P \end{bmatrix}$$ (3)
4.15 Position Tracking
Initially the acceleration data from the IMU was used to calculate x and y distance travelled by the robot. In order to do this the raw values were run through a circular buffer filter and integrated twice. The error using this method was large and increased over time therefore it was deemed not suitable for our purposes. Next optical encoders were installed on all four wheel axles in order to measure wheel rotation and thus determine vehicle kinematics. Because of the tendency of the Mecanum wheels to slip during most types of movement an accurate level of position tracking was not achievable. In order to solve this problem another approach was taken. Using the body and ball of a computer mouse with two of the provided optical rotary encoders, absolute x and y position can be obtained. The ball mouse setup works well on the flat hard surface of the test platform. This setup proved to be very accurate and gave approximately 1cm precision.

**Stage 1**

**Implementation:**
X and Y acceleration values from IMU are filtered then integrated twice.

**Observations:**
Double integration resulted in large error.
Error increased over time. Achievable accuracy was poor.

**Stage 2**

**Implementation:**
Optical encoders attached to each axle. Distance moved by each wheel determined by forward kinematics.

**Observations:**
Mecanum wheels had a large amount of slip.
Slip was worst when performing strafing movements.

**Stage 3**

**Implementation:**
Mechanical ball mouse modified to use the provided optical encoders. Mounted underneath the robot with the ball in the centre.

**Observations:**
Works well on hard flat surface. No slip therefore very accurate.
Approximately 1cm precision achieved.
4.16 Orientation Tracking

Initially the magnetometer on board the MPU-9150 was used to calculate the heading angle of the robot. The magnetometer required calibration each time it was used. The raw values it gave were not steady and required extensive filtering. The advantage of the magnetometer was that it gave an absolute heading reference, however it was found to be highly affected by surrounding magnetic fields and was thus not usable on its own. Next the gyro was tested to see if it could give a more stable heading. The gyro data from the IMU gives angular rotation about z axis in deg/s. This is integrated to give change in heading in degrees. The gyro gave very precise changes in heading however suffered from drift over time and could not track fast changes in heading.

In order to combine the advantages from both the magnetometer and gyro we fused the data using a complementary filter as described in Equation 4. The complementary filter worked by calculating heading as a factor of gyro and magnetometer. A ratio of 95% gyro to 5% was implemented in order to reduce the gyro drift over time.

Complementary Filter:

\[
\text{filtered heading} = \text{factor} \times \text{gyroscope heading} + (1 - \text{factor}) \times \text{magnetometer heading} \quad (4)
\]

<table>
<thead>
<tr>
<th><strong>Stage 1</strong></th>
<th><strong>Stage 2</strong></th>
<th><strong>Stage 3</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Implementation:</strong></td>
<td><strong>Implementation:</strong></td>
<td><strong>Implementation:</strong></td>
</tr>
<tr>
<td>Raw magnetometer data from IMU filtered using a circular buffer filter.</td>
<td>Raw gyro data filtered and then integrated to give change in orientation.</td>
<td>Raw data gyrometer and accelerometer data from the IMU is filtered and then fused using a complementary filter.</td>
</tr>
<tr>
<td><strong>Observations:</strong></td>
<td><strong>Observations:</strong></td>
<td><strong>Observations:</strong></td>
</tr>
<tr>
<td>Provides absolute position reference.</td>
<td>Gives very precise changes in heading.</td>
<td>A 95% factor of the gyro heading is used along with a 5% factor of the magnetometer.</td>
</tr>
<tr>
<td>Requires time consuming calibration on startup.</td>
<td>Accuracy drifts a small amount over time.</td>
<td>Observations: Fused data gives the “best of both worlds” in terms of combining the benefits of the magnetometer and gyro.</td>
</tr>
<tr>
<td>Requires extensive filtering.</td>
<td>Easy to lose track of heading during fast turning movements.</td>
<td>Heading output is stable and accurate.</td>
</tr>
<tr>
<td>Readings are highly affected by magnetic fields and ferrous materials surrounding the test platform.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 12: Orientation development process
4.17 Data Processing and Mapping

A program external to Arduino was developed to generate a real time visual representation of what the car sees and where it has been. The software was developed within the Processing development environment.

Processing programs can communicate via serial with Arduino devices so this allowed the robot to serial print information which can then be interpreted. A serial event interrupt within processing was used to handle the serial communication, reading in strings which are printed in a known order corresponding to certain bits of information needed to map. No information is sent back to the robot and the Processing program in no way enhances the robots ability to map and localise itself as it moves.

The interface allows the view of the current position of the car and any readings it sees from its sensors. When the record button is pressed all information plotted on the screen is recorded meaning that the cars movement can be traced and all the range findings can build up the environment.

The car was modelled as a scale rectangle, representing its overall coverage of the ground. An arrow is drawn in the middle of the car showing the current heading. As the heading of the real car changes a rotation matrix is applied with that current heading to update the image on the screen. Detected walls are plotted as strings of red dots. High densities of these dots indicate the strong presence of a wall. Green squares are used to plot the position of detected corners. These are placed on the map when a wall is detected close by on perpendicular side of the robot.

![Image](image.png)

*Figure 13: Mapping robot position and orientation*
4.18 Software

4.18.1 Line Following
We implemented a function which takes as its inputs a desired heading in degrees as well as a set distance from the wall. Both of these inputs are PID controlled to improve rate of response and minimise steady state error.

A heading error is generated when the robot’s current yaw orientation (from the gyro) diverges from the desired orientation. To correct the error, one side of the robot is given a higher PWM signal to its motors, with this offset a function of the PI gains. The desired heading is calculated from the gradient of the longest wall.

Distance from the wall is controlled in a similar manner. These inputs are controlled in series within the same function. By incrementing the set distance from wall at each length of the room, the robot is able to efficiently cover the entire environment in the shortest time possible.

![Image of line following](image)

*Figure 14: Map showing line following operating*
4.18.2 Wall and Corner Detection

In order to give the robot a more intelligent sense of its environment, an algorithm was implemented to interpret certain sensor readings into information about where corners and walls are located. The algorithm is based off the assumption that a 'wall' is any significantly long and straight feature. Similarly a 'corner' is defined as the intersection of two walls.

This algorithm works by performing a rotational movement around a fixed point whilst simultaneously collecting a dataset of range measurements from an IR sensor. Each data point is converted into its correct Cartesian coordinates using the transformation matrix described in Section 4.14. These coordinates are then passed through a linear regression function to convert points into mathematical equations representing lines.

The aim is to fit a linear model, $y = mx + c$ to the data points divided into 7 groups of 7. The constants are calculated as such:

$$m_j = \frac{7S_{xy} - S_xS_y}{7S_{xx} - S_x^2} \quad (5)$$

$$c_j = \left(\frac{S_y}{7}\right) - \left(\frac{m_jS_x}{7}\right) \quad (6)$$

Where,

$$S_x = \sum_{i=1}^{i=7} x_i \quad S_y = \sum_{i=1}^{i=7} y_i \quad S_{xx} = \sum_{i=1}^{i=7} x_i^2 \quad S_{xy} = \sum_{i=1}^{i=7} x_iy_i \quad S_{yy} = \sum_{i=1}^{i=7} y_i^2$$

By rejecting any results with low $R^2$ values, and averaging similar equations together, lines can be interpreted accurately. Corners are found by equating two line equations with significantly different gradients.

The algorithm was validated using simulated and experimental datasets. The results for a real life dataset are shown in Figure 15 below.

![Figure 15: Corner detection data](image)
This algorithm enables a significant reduction in memory required to store information about the environment as only 4 coordinates are necessary to describe a rectangular enclosed space. It has been optimised to run efficiently on an 8 bit microcontroller and takes less than 500 milliseconds to execute. Furthermore this algorithm enables an ‘object’ to be easily distinguished from a ‘wall’ by assuming that any detected feature that lies significantly outside the equation for a wall is an object.

4.18.3 Obstacle Detection and Avoidance

All of the range sensors are used to detect anything that comes within a dangerous boundary around the robot. When it does see something the robot will perform a movement to determine what it is that it has found and take the necessary action to move on from that. The obstacle detection algorithm was designed to handle a number of scenarios, and had built in redundancy to account for obstacle situations that would be found in real life environments.

When anything is found in front of the robot while it is moving forward it will stop, move sideways and then check whether there is something still in front. The assumption was made that no object was larger than a coke can so we move sideways far enough to make sure that if there is a coke can in front then the robot will be able to move forward clear of it afterwards. In that case the robot will move forward a distance until it is clear of the object and then move back into the original path that it was on. If after moving sideways in the initial detection there is still something in front of the sensor it is safe to assume that the robot has in fact found a wall and would proceed with traversing the area accordingly.

Similarly the robot will take action when anything is found on the side of the robot while it is moving sideways. This follows the same logic whereby the robot will move forwards to check what it has found on its side and move off from there to follow its path without hitting anything.

Multiple objects in close proximity to one another were navigated using a recursive avoid function. This would be applicable to real world scenarios where the environment is completely unknown and changing. Objects placed against walls were also accounted for by treating them as a possible wall then sweeping the sensors from the side to side to distinguish whether open space lay behind. In the case of a wall there would be no open space.
4.18.4 System Routine

The main program loop is presented in Figure 16 in the form of a finite state diagram.

![Diagram of Main Program Finite State Diagram](figure16.png)
4.18.5 Start-up Algorithm

The start-up routine is presented in Figure 17

![Start-up routine flow diagram]

Figure 17: Start-up routine flow diagram
4.18.6 Traverse and Object Avoidance Algorithm

The traverse and object algorithm is shown in Figure 18.

Figure 18: Traverse and object avoidance flow diagram
5  Testing and Results

5.1  Displacement Sensors
The output from the infrared sensors proved to vary depending on the colour of the object they were detecting. They were also affected by the shape of object surface. Object surfaces that were parallel with the sensor gave much higher readings.

Preliminary testing of the sensors revealed that the ultrasonic sensors had a large coverage angle and therefore were not useful in determining precise location of objects, but could be used to detect that an object was close to the robot. A cone device was attached to the ultrasonic sensors in an attempt to improve accuracy however results were inconclusive and the idea discarded.

The individual infrared sensors had substantially different characteristic output curves, shown in Figure 19. The individual ultrasonic sensors had an almost identical characteristic and could be approximated by the same characteristic equation also shown in Figure 19.

The characteristic equations of the displacement sensors are:

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Fitted equation</th>
<th>R² value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left Medium Range IR</td>
<td>( distance = 632208 \times voltage^{-1.423} )</td>
<td>0.995</td>
</tr>
<tr>
<td>Right Medium Range IR</td>
<td>( distance = 57533 \times voltage^{-1.139} )</td>
<td>0.989</td>
</tr>
<tr>
<td>Left Long Range IR</td>
<td>( distance = 830873 \times voltage^{-1.306} )</td>
<td>0.993</td>
</tr>
<tr>
<td>Right Long Range IR</td>
<td>( distance = 752052 \times voltage^{-1.302} )</td>
<td>0.994</td>
</tr>
<tr>
<td>Ultrasonic</td>
<td>( distance = 13.058 \times voltage + 53.995 )</td>
<td>0.999</td>
</tr>
</tbody>
</table>

Figure 19: Displacement sensor characteristics
5.2 Position and Orientation Tracking

The traverse algorithm requires incrementing distance from adjacent wall by a fixed distance (15 cm) at each length. Therefore some form of motion in the x axis was required to increment the robots distance. Two possible methods were compared to determine the most suitable one for the application. The first method involves strafing along the x axis, whilst the second method involved performing a 90 degree rotation followed by a straight motion along the robot’s y axis. In order to validate the options, a square path was programmed to assess closed loop path following performance. The results of this test are shown in Figure 20. For small strafing distances, the first method has a small performance advantage over the second. The x and y trajectories follow the programmed path well. Furthermore there was a 21% speed advantage, as the rotations cause significant delays.

![Diagram of closed loop path following performance comparison](image-url)

Figure 20: Closed loop path following performance comparison
5.3 Results of final run

The robot achieved approximately 80% coverage of the environment, and correctly mapped its trajectory as well as identifying and plotting features such as corners and objects. The wall finding algorithm proved its long range capabilities by correctly determining an appropriate wall to head towards at the start.

However, the gyro showed signs of drift during the course of the run which contributed to the overall loss of coverage particularly in one corner.

6 Discussion

Whilst the performance of the robot was satisfactory in both its mapping and coverage capabilities, further improvements are required in order to achieve a full SLAM robot.

Most of the gyro drive was attributed to the fluctuations in the 555 timer’s period. A more stable timer would be required for further accuracy.

Additional sensors are required to achieve 360 degree detection. The method in which the robot traversed the environment required sensor placement on all sides to detect objects. An alternate path whereby the robot span 90 degrees at each wall would allow for less overall sensors more focused on looking forward.

The real-time, Bluetooth mapping was successful.

7 Conclusions

- An autonomous SLAM robot was designed and developed for the purpose of vacuum cleaning.
- The robot fuses sensors and intelligent control to successfully navigated an unknown environment whilst simultaneously performing real time mapping and obstacle avoidance.
- The robot is designed to achieve full coverage of a room in the shortest time possible by performing a series of parallel movements.
- An algorithm combines long range sensor data to interpret the locations and headings of regular environmental features such as walls.
- The robot prototype was tested and validated against a range of possible scenarios to ensure a robust solution.
- The final run time was performed within 3 minutes with approximately 80% coverage.