

Simultaneous localization and mapping (SLAM) with an autonomous robot



Inspiring Excellence

Md. Shadnan Azwad Khan 13321076

Shoumik Sharar Chowdhury 17141025

Nafis Rafat Niloy 13115001

Fatema Tuz Zohra Aurin 13101025

Supervisor: **Dr. Tarem Ahmed**

Co-Supervisor: **Moin Mostakim**

Submitted to:

Department of Electrical and Electronic Engineering

School of Engineering and Computer Science

BRAC University

Declaration

We, hereby declare that this thesis project, neither in whole or in part, has been previously submitted for any degree. It has been completed based on the knowledge we have acquired and implemented ourselves. Materials of work accomplished by others and the references gained from the official websites are mentioned by reference.

Md. Shadnan Azwad Khan

Shoumik Sharar Chowdhury

Nafis Rafat Niloy

Fatema Tuz Zohra Aurin

Under the supervision of,

Dr. Tarem Ahmed

Assistant Professor

Department of Electrical and

Electronic Engineering

Acknowledgement

We have been, throughout this thesis, fortunate to have the support of our friends, family and mentors. In particular, we would like to thank our supervisor Dr. Tarem Ahmed for guidance, patience and for taking immense interest in our work, and Moin Mostakim for sharing his knowledge and insight on some of the finer algorithms we had to work with. We are grateful for the support from the esteemed faculties and technicians of the School of Engineering and Computer Science. This thesis would not have been possible without their help.

List of Figures

1. Rover locomotion.....	10
2. A map produced by the Arduino Rover project.....	12
3. Raspberry Pi 3 Model B	14
4. Raspberry Pi 3 Model B pin diagram	15
5. Ultrasonic Range Sensor HC SR04	15
6. HC SR04 connection diagram for 3.3V GPIO operation	16
7. Inertial Measurement Unit (IMU)	17
8. Figure 8. Wheel encoder with wheel, motor and brackets	17
9. Oscilloscope capture of encoder outputs with the wheel spinning at 630 RPM	18
10. L298N stepper motor driver	19
11. Pseudocode for sensor sweep	25
12. Pseudocode for dead reckoning	26
13. The movement prediction using rotary angles	27
14. The block diagram of the Kalman filter designed for dead reckoning	29

15. Applying transform over robot position and sonar measurement to recognize occupied cells	31
16. Pseudocode i for navigation	33
17. Pseudocode ii for navigation.....	34
18. An example of what the Matplotlib library does.....	35
19. Actual map from survey of room.....	36
20. Map obtained from SLAM.....	37

List of Tables

1. GPIO connections to sensors and actuators	22
2 .Hardware expense.....	22
3. Sonar uncertainty.....	30

Content

Abstract	1
1. Introduction	2
1.1 Motivation	3
1.2 Thesis Outline	4
2. Literature Review	6
2.1 SLAM Approaches	6
2.2 Related Work	10
3. Hardware Specification	13
3.1 Introduction	13
3.1.1 Raspberry Pi 3 Model B	14
3.1.2 Sonar Sensor	15
3.1.3 Inertial Measurement Unit	17
3.1.4 Wheel, Gear Motor and Wheel Encoder	17
3.1.5 Motor Controller	19

3.1.6 Power Supply Setup	19
3.2 Implementation	20
3.2.1 Robot Setup	20
3.2.2 Hardware Expense	22
4. SLAM Implementation	23
4.1 Relative Positioning and Kalman Filtering	25
4.1.1 IMU data handling.....	26
4.1.2 Wheel encoder odometry.....	27
4.1.3 Dead Reckoning with Kalman filter	28
4.2 Occupancy Grid Mapping	29
4.2.1 Sonar data and uncertainty modelling	30
4.2.2 Occupancy grid map method	31
4.3 Navigation and Exploration	32
4.3.1 Motor controller operation	32
4.3.2 Exploration Strategy	32
4.4 Graphical Representation	35
5. Experimental Results and Evaluation	36
6. Concluding Remarks	38
6.1 Applications	38

6.2 Limitations and Future work	38
6.3 Conclusion	39
References	41

Abstract

Simultaneous localization and mapping (SLAM) is a problem in which an autonomous robot is required to initiate in an unknown location within an unknown map and incrementally build a map of its environment while also using sensory data and the map to find its location with very minimal errors. We present a solution for SLAM with an autonomous differential drive robot in an indoor environment. We extract data from sonars and analyze it to construct a map of the explored area using occupancy grid mapping. In addition, relative positioning approaches provide location using an inertial measurement unit (IMU) and wheel encoders. Localization and mapping are essential tasks for an autonomous robot's navigation or exploration without a prior map and the system is based on discrete step-wise or event modeling which guides its navigation. The systems result is compared for accuracy against an actual map.

Keywords: simultaneous localization and mapping, Kalman filter, autonomous mobile robot, dead reckoning, occupancy grid map, motion planning

Chapter 1

Introduction

Simultaneous localization and mapping or SLAM is a key problem within mobile robotics as it can provide for a way towards truly autonomous robots or vehicles. SLAM is the problem in which a mobile robot when placed in an unknown environment must find a way to explore and navigate around the environment to produce a map of the environment and at the same time to estimate its position by only using on board sensors. A robot given a prior map would not have too much of a problem figuring out its location within the environment by recognizing specific features. Similarly, a robot which is given accurate position data would be able to produce a map of the environment. The main difficulty in SLAM lies in the absence of both accurate data on the map of its environment or position within it. This results from several smaller complications, each of which in some way arises from inherent uncertainty pertained to the problem, the major factors of these being,

1. **Limitations of sensors:** Sensors are subject to biases in measurements arising from systemic biases that constrain resolution or range of the sensors as well as noise which affects the measurements in unpredictable ways. Despite efforts to calibrate, information received from sensors are subject to uncertainty.

2. **Dynamic environments:** Environments are rarely structured and stable. Highly dynamic environments as is in real life mean that portions of the environment may change with respect to time. This results in added unpredictability.
3. **Motion model:** The vehicle motion model incorporated within the SLAM approach would rarely be completely accurate. Physical processes would need approximations when dealt with in computation.
4. **Algorithmic inaccuracy:** Computationally heavy tasks can take a lot time, hence, usually accuracy is sacrificed for faster responses within environments.
5. **Actuator faults:** Low cost mobile robotics mean that actuators in use can have a high degree of control noise. Problems could arise from wear and tear as well. This can augment sensor errors.

The consequence of these antagonistic issues is that nothing regarding the robot's position or environment can be known with absolute certainty. Therefore, the SLAM problem is solved by keeping the uncertainties in mind and employing probabilistic methods which give the best guess or estimates of the robot's position. Hence, SLAM solutions are usually more concerned with operational use than with extreme accuracy.

1.1 Motivation

We wish to solve the problem for exploration and mapping of an unknown environment. This would help in all kinds of future developments such as search and rescue missions and

reconnaissance operations. LiDAR or cameras are usually employed in mapping the environment because of their more complex and accurate data. Ultrasonic range sensors, or sonars, on the other hand, are inexpensive and hence, make it a more realistic approach to take for developing countries like Bangladesh. This necessitates a different approach to SLAM in which the map produced is within an acceptable threshold of error. We wish to move away from feature based SLAM to a grid based SLAM, and develop a novel way to figure out occupancy grid mapping.

1.2 Thesis Outline

This thesis is organized into several chapters:

1. In chapter 2, past research work and related papers are discussed. We also explore some SLAM approaches as found in prior work that can be of use.
2. Chapter 3 introduces the hardware parts and the related sensors and actuators that are needed to build an autonomous robot. The specifications for each equipment is discussed and the setup of the robot is explained.
3. Chapter 4 explains the core algorithm at work: the approaches taken for localization and mapping are discussed separately. Exploration strategy to traverse the environment is outlined. Additionally, the method for graphical representation of the complete map is illustrated.

4. In chapter 5, the results of the experiment of the implementation is cross-validated with a known map. The resulting map is shown in comparison to the actual map.
5. Finally, chapter 6 summarizes the thesis and discusses the possible applications of SLAM along with limitations in our experiment. Future work that we intend to do is also discussed.

Chapter 2

Literature Review

SLAM approaches from the robotics community are delineated and explained. This pertains to the wider scope of work done on the topic. We also delve into related work that is much more readily comparable to this thesis.

2.1 SLAM Approaches

Laser ranging systems are accurate active sensors which in its most common form operates on the time of flight principle of a laser pulse and its reflection from an object with a measure of the time taken by the pulse to have gone and reflected back. Sonar-based SLAM solutions focus on information that is measured fast but lack the appearance data found. However, a small error can have large effects on later position estimates due to its dependence on inertial sensors such as odometers. On the other hand, Vision systems are passive, they have long range and high resolution, but the computation cost is considerably high and good visual features are more difficult to extract and match. Vision by means of stereo camera pairs or monocular cameras is used to estimate the 3D structure, feature location and robot pose.

The use of an Extended Kalman Filter (EKF) to estimate a state vector that contains both the robot pose (which includes position and orientation) and the landmark locations, remains one of the most popular strategies for solving SLAM [1]. According to Riisgaard and Blas [2], “An EKF (Extended Kalman Filter) is the heart of the SLAM process. It is responsible for updating where the robot thinks it is based on these features.” Smith and Cheeseman [3] introduced a general method to using the Kalman filter for estimating the expected error (covariance) and nominal relationship between coordinate frames. These coordinate frames, which represent the relative locations of objects, may be known only indirectly through a series of spatial relationships. The writers claimed that the calculated estimates from this method agree well with those from an independent Monte-Carlo simulation and that by using this method, it is possible to decide, in advance, whether or not an uncertain relationship is known accurately enough for the robot to execute a given task and, if not, how much of an improvement in locational knowledge will be provided via a proposed sensor. Guivant and Nebot [4] believe that future work will likely address an extension of the Kalman filter that may result in a more decentralized SLAM implementation that will demonstrate different devices updating their own maps via their respective sensors while simultaneously transferring all that accumulated data to the rest of the system.

A paper by Montemerlo et al. [5] presented FastSLAM, an algorithm that recursively estimates the full posterior distribution over robot pose and landmark locations, yet scales logarithmically with the number of landmarks in the map whereas Kalman filter-based algorithms, for example, require time quadratic in the number of landmarks to incorporate

each sensor observation. This FastSLAM algorithm is based on an exact factorization of the posterior into a product of (conditional) landmark distributions and a distribution over robot traversed paths. The algorithm is claimed to have run successfully on as many as 50,000 landmarks, in environments far beyond the reach of previous approaches. Furthermore, experimental results demonstrate the advantages and limitations of the FastSLAM algorithm on both simulated and real-world information.

Another paper by Self, Smith & Cheeseman [6] describes the importance of a stochastic map, which is basically a representation for spatial information, and its associated procedures for building it, reading information from it, and revising it incrementally as new information is obtained. The stochastic map contained estimates of relationships among the objects identified within the map along with their uncertainties, given all the available information. This procedure provides a general solution to the problem of estimating uncertain relative spatial relationships between objects in a given map. These estimates are probabilistic in nature and is an advance over the previous, more conservative, worst-case approaches to the problem. The procedures are claimed to have been developed in the context of state-estimation and filtering theory, which provides a solid basis for numerous extensions.

A hierarchical SLAM framework, presented by Estrada et al. [7], consists of a set of local maps that are connected by arcs labelled with relative location between the maps. In comparison to previous approaches, it effectively maintains loop consistency when calculating the optimal estimate at a global level. Hierarchical SLAM, addresses the complexity of mapping large areas by building a set of independent local stochastic maps. This procedure

can be carried out in constant time per step, by limiting the maximum local map size. An upper global level contains a graph whose nodes correspond to the local maps and whose arcs represent adjacency relations. Using a stochastic map representation, the metric information about the relative location between adjacent maps is also stored at the global level. An additional advantage of this approach is that several robots can contribute new local maps or refine previously built local maps.

Considerably, impressive progress has been witnessed over the decades. However, much is still to be accomplished, given the exponential rate at which technology is advancing. Future attempts may incorporate SLAM in multiple autonomous mobile robots that collectively work to complete a given task, a full relative stochastic map framework that handles interoceptive and exteroceptive measurement types, a means to incorporate global measurements in the relative framework when available, advanced real-time implementations & optimized loop closure mechanisms. Guivant and Nebot [3], who use the Extended Kalman Filter to implement the SLAM algorithm in real-time, believe that “Future work will address the extension of the compression filter results in decentralized SLAM where different platforms can update their own map with a particular sensor and then transfer all the information gained to the rest of the system”.

2.2 Related Work

The following projects are similar to our own and provided key insights for this thesis:

1. Implementing Odometry and SLAM Algorithms on a Raspberry Pi to Drive a Rover. [8]

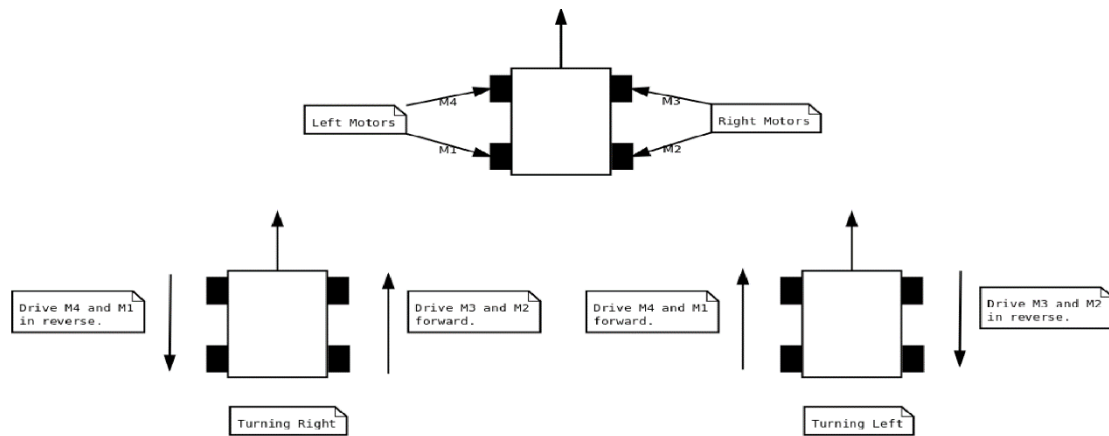


Figure 1. Rover locomotion

Main features of the project:

- a. An onboard Arduino controls the rover while it is sent control commands by a Raspberry Pi which carries out SLAM computations.
- b. A Laptop in turn communicates with the Raspberry Pi through wireless communication.
- c. The rover reports two different types of events; its change in odometry and the results of a scan during which the rover collects 180° readings from its servo mounted sonar.

- d. The SLAM algorithm deployed is based on the Extended Kalman Filter and Random Sampling Consensus (RANSAC) techniques. It was designed to extract two types of landmarks (straight lines and spikes).

2. An autonomous, Arduino-based rover that creates maps of its surroundings using sonar sensors. [9]

Main features of the project:

- a. An Arduino is responsible for collecting sensor data and sending it to a computer. It receives control commands from the computer and works as onboard controller for the rover.
- b. A simple Mac application in Swift that both communicates with the robot via Bluetooth and visualizes the received data in a simple UI.
- c. All algorithms have been implemented in C++ with the code requiring the boost 1.59 and OpenCV 3.0 libraries.
- d. Uses probabilistic occupancy grid mapping and a simple edge following map making strategy. The robot tries to drive past obstacles at a short distance until all obstacles have been visited.

- e. The rover is based on the Dagu Rover 5 platform with 4 motors and encoders, an Arduino Mega, a Redbear BLE Shield, 3 SR04 sonar sensors and the Polulo MinIMU compass and gyro.

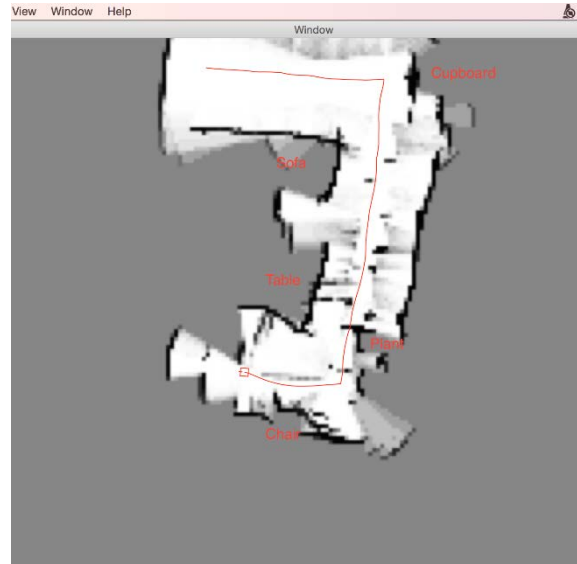


Figure 2. A map produced by the Arduino Rover project

Chapter 3

Hardware Specifications

We have assembled a robot with the intention to fulfill all the requirements towards a proper solution for SLAM whilst keeping the project inexpensive. The resulting differential drive mobile robot is described in the following subsections with descriptions of the specific hardware parts in use.

3.1 Hardware Introduction

A Raspberry Pi 3 Model B is used as the main control board of the robot. Several sensors like sonars, inertial measurement unit (IMU) and wheel encoders are attached to it. A motor controller and gear motors are used as actuators. A power supply unit is setup to power all the devices and sensors.

3.1.1 Controller and Processor



Figure 3. Raspberry Pi 3 Model B

A Raspberry Pi is a small single-board computer (SBC) which is used as the main computational block of the robot. The Raspberry Pi 3 Model B has a 1.2GHz 64-bit quad-core ARMv8 CPU with 1GB RAM [10]. There are 40 General Purpose Input-Output (GPIO) pins in the device which are used to communicate with the sensors and actuators. 4 of the pins are used as power outputs and 7 of them can be used for grounds. Some pins can also be used for specific purposes like Inter-Integrated Circuit (I2C), Pulse-Width Modulation (PWM), etc. of which some of those are used in the making of the robot. The Pi runs on a Debian based operating system called the Raspbian Jessie that is installed on an 8GB SD card.

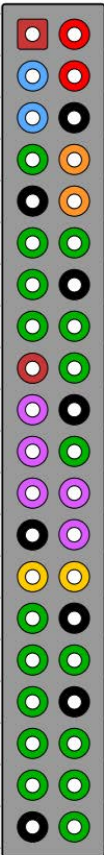
Raspberry Pi 3 GPIO Header				
Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)		DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)		(I ² C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40
Rev. 2 29/02/2016		www.element14.com/RaspberryPi		

Figure 4. Raspberry Pi 3 Model B pin diagram

3.1.2 Sonar Sensor



Figure 5. Ultrasonic Range Sensor HC SR04

Ultrasonic range sensors, or sonars, of model HC-SR04 are used to detect obstacles and walls. There are 4 pins in the sensor: VCC (Power), Trig (Trigger), Echo (Receive) and GND (Ground) [11]. The Trig transmits a pulse and the Echo receives the pulse. The time in between the transmission and reception is measured and the distance of the obstacle is measured by multiplying the speed of sound with (time/2).

Ideally, the range of the sensor is 2 cm to 400 cm and the ideal measuring angle is 15 degrees [11] but an experiment was carried out by us which gave us a more realistic range and angle and so the uncertainty and range of the sonar was obtained.

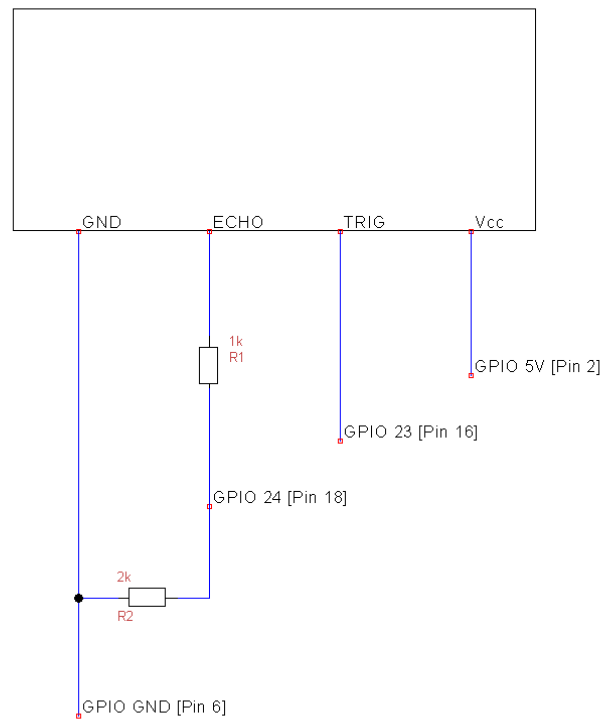


Figure 6. HC SR04 connection diagram for 3.3V GPIO operation

3.1.3 Inertial Measurement Unit

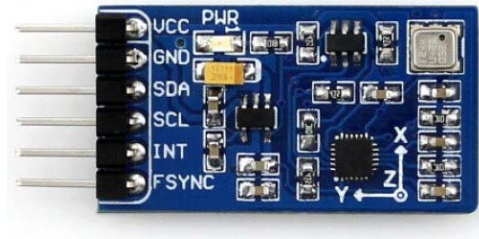


Figure 7. Inertial Measurement Unit (IMU)

A 10 DOF Inertial Measurement Unit (IMU) is used in the robot. Only 8 degrees of freedom (DOF) are needed: the MPU 9255 which consists of a 3 axis-gyroscope, 3-axis accelerometer and 2-axis digital compass/magnetometer is used. The IMU provides measurements in acceleration, angular rates and magnetic field around the robot [12]. These are used to measure the position and heading of the robot and helps in localization, together with the wheel encoders.

3.1.4 Wheel, Gear Motor and Wheel Encoder



Figure 8. Wheel encoder with wheel, motor and brackets

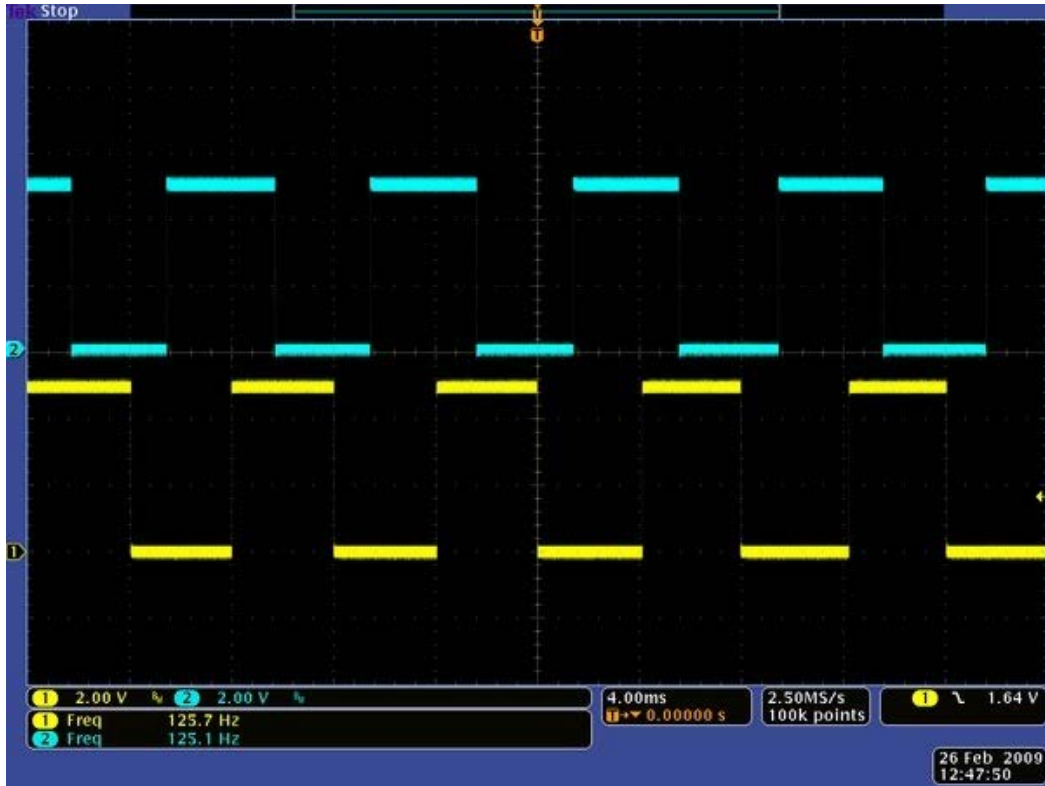


Figure 9. Oscilloscope capture of encoder outputs with the wheel spinning at 630 RPM

A quadrature encoder board is used to measure the number of revolutions that the wheels make in a given time. The Pololu wheel encoders work with micrometer gear motors by holding two infrared reflectance sensors inside the hub of the 42x19mm Pololu wheels that are used in the robot. The wheels have 12 teeth each along the rims. The sensors are placed to provide waveforms 90 degrees out of phase, allowing for a resolution of 48 counts per wheel rotation and determining a direction of the rotation [13]. The encoder is used along with the IMU for localization [13].

Two 30:1 micro metal gear motors are placed inside the wheels to drive the robot. The gear motors are 6V high-powered brushed DC motor with a 29.86:1 metal gearbox [14]. At 6V, the motors run at 1000 rpm and 120mA with no load [14].

3.1.5 Motor Controller

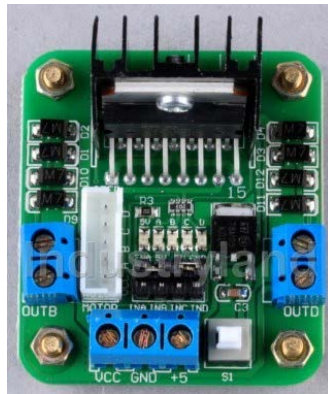


Figure 10. L298N stepper motor driver

An L298N dual motor driver is used to run the gear motors. This motor controller also enables us to use pulse-width modulation (PWM) which is used to control the speed at which the motors rotate. In addition, the motor controller allows control on the direction of the two wheels to allow for differential drive.

3.1.6 Power Supply Setup

A 2.0A portable power bank is used to power the Raspberry Pi. The motors are powered via the motor controller with a 9V external battery. The sonars, wheel encoders and IMU is powered by a breadboard power supply stick supplying 3.3V and 5V. The breadboard power

supply stick is converts voltage from a 9V battery. The Raspberry Pi has 3.3V and 5V GPIO pins to supply power but is limited by a maximum current that can be drawn, and running so many sensors can be dangerous for the Raspberry Pi. The devices and the Raspberry Pi share a common ground nevertheless.

3.2 Hardware Implementation

3.2.1 Robot Setup

Raspberry Pi pin number			Raspberry Pi pin number
1	3.3V power (not used)	5V power (not used)	2
3	SDA of IMU	5V power connected to (not used)	4
5	SCL of IMU	Ground (not used)	6
7	Motor controller: used to make left wheel go forward	Trig of Sonar 2	8
9	Ground connected to breadboard power stick ground	Echo of Sonar 2	10

11	Motor controller: used to make left wheel go backward	Enable A of motor controller. Used for PWM.	12
13	Motor controller: used to make right wheel go forward	Ground (not used)	14
15	Motor controller: used to make right wheel go backward	Trig of Sonar 1	16
17	3.3V (not used)	Echo of Sonar 1	18
19	Trig of Sonar 3	Ground connected to	20
21	Echo of Sonar 3	Not used	22
23	Trig of Sonar 8	Echo of Sonar 8	24
25	Ground (not used)	Not used	26
27	Not used	Not used	28
29	Trig of Sonar 4	Ground (not used)	30
31	Echo of Sonar 4	Trig of Sonar 5	32
33	Enable B of motor controller. Used for PWM.	Ground (not used)	34
35	Trig of Sonar 7	Echo of Sonar 5	36
37	Echo of Sonar 6	Trig of Sonar 6	38

39	Ground (not used)	Not used	40
----	-------------------	----------	----

Table 1. GPIO connections to sensors and actuators

The wheels, motors, IMU, motor controller, 9V battery pack, and wheel encoders are placed on the robot chassis (Pololu 5" Robot Chassis RRC04A) . A ball caster is attached underneath the chassis for stability. A second platform is attached to the chassis for the Raspberry Pi, sonars and for the power bank. Eight ultrasonic range sensors, or sonars, are used instead of one sonar mounted on a servo to reduce error caused by control noise and give faster reading. The sensors and actuators are connected to the Raspberry Pi as detailed in the table above.

3.2.2 Hardware Expense

Hardware Parts	Quantit y	Subtotal (BDT)
Raspberry Pi 3 Model B	1	3800
Sonar Sensor (HC-SR04)	8	1467
Sonar Sensor Base	8	242
Motion Sensor Monitor 10 DOF IMU Sensor (B)	1	1450
Pololu 42x19mm Wheel and Encoder Set	1	3829
30:1 Micro Metal Gearmotor HP	2	2000
Pololu Ball Caster with 1" Plastic Ball	1	351
L298N Stepper Motor Driver	1	409
Pololu 5" Robot Chassis RRC04A	1	471
Jetech PowerBank, 9V batteries, Breadbaord powersupply	1	1008
Total (BDT)		15026

Table 2: Hardware expense

Chapter 4

SLAM Implementation

The SLAM implementation is divided into three parts:

1. **Relative positioning** with Kalman filter (using IMU and wheel encoder odometry data)
2. **Occupancy grid mapping** using sonars (8 sonars divided between 360 degrees)
3. **Navigation** based on stepwise update of path

The process works in incremental steps in which the robot reads its localization sensors to update its position in each measurement. The sonars measurements are done in the same code for a complete sweep of the sensors. This information is handled by the Occupancy grid map and Relative positioning algorithms. This completes a step for the robot after it reaches its trajected position within the map. Then the navigation algorithm decides the next trajected step. The system continues in this way until termination conditions are fulfilled.

The system is implemented using python scripts and data stored in CSV files on the Raspberry Pi that could be initiated by a Linux computer or laptop connected in the same wireless network as the Raspberry Pi. The main components are divided as such:

1. **sensorsweep.py**: for running the code that reads the IMU, wheel encoder and sonar data.
2. **sensordata.csv**: for storing the sensor data obtained from sensorsweep.py.
3. **deadreckoning.py**: for applying a Kalman filter over IMU and wheel encoder data.
4. **statepos.csv**: for storing the location and pose of the robot.
5. **occupancy.py**: for running the occupancy grid algorithm over sonar data.
6. **nav.py**: for running navigation algorithm.
7. **slam.py**: the main algorithm that handles the rest within its event model, also including the starting and terminating conditions.

The specific pseudocodes and calculations and their explanations is expounded in the other sub-sections. The pseudocode for the sensor sweep is provided below:

- | |
|--|
| <ol style="list-style-type: none">1. initialize the sensors2. initialize the FaBo9Axis_MPU9250 library3. function wheelencoder()4. function read8axis()5. call the FaBo9Axis_MPU9250 library |
|--|

```

6.     return x,y,z values from accelerometer, gyroscope and x,y values from magnetometer
7.     function sonar(sonar_number)
8.     WHILE True:
9.         call wheelencoder()
10.        call Sonar(sonar_number) for the eight sonar modules
11.        call read8axis()
12.    WRITE to sensordata.csv

```

Figure 11. Pseudocode for sensor sweep

4.1 Relative Positioning and Kalman Filtering

```

1.     initialize FilterPy library
2.     function robot_moving()
3.     function calc_velocity_orientation()
4.     function measure_velocity_orientation()
5.     import IMU and wheel encoder data from sensordata.csv
6.     if robot_moving() is true:
7.         get inertial data
8.         if calc_velocity_orientation() is 4Hz:
9.             get data from magnetometer and encoder
10.            measure_velocity_orientation()

```

```
11.          call FilterPy
12.          return state_estimation
13.      else:
14.          estimate position and heading angle
15.          WRITE to statepos.csv
16.  else:
17.          return to 6.
```

Figure 12. Pseudocode for dead reckoning

The above pseudocode explains the process behind dead reckoning that we have implemented.

The more intricate mathematics behind it is discussed below.

4.1.1 IMU data handling

Absolute three-dimension acceleration with respect to the body frame is measured by the tri-axial accelerometers. The inclination of the mobile robot is estimated with three orthogonal accelerometers because the tri-axial accelerometers when the mobile robot is stationary provide very accurate information. Additionally, single and double integration of the accelerations provide the velocity and position, respectively. The three orthogonal gyroscopes provide the angular rates with regard to the three axes. The integration of the angular rates provides the orientation of the mobile robot. In addition, the integration of the tri-axial

gyroscopes provides the orientation not only when the mobile robot is stationary, but also when it is moving. The mangetometer data helps correct angular rates provided by the gyroscope.

The MPU9255 a 3-axis gyroscope, 3-axis accelerometer, and 3-axis magnetometer is the IMU we are working with. An open-source python library, FaBo9Axis_MPU9250, is used to detect the readings of the IMU.

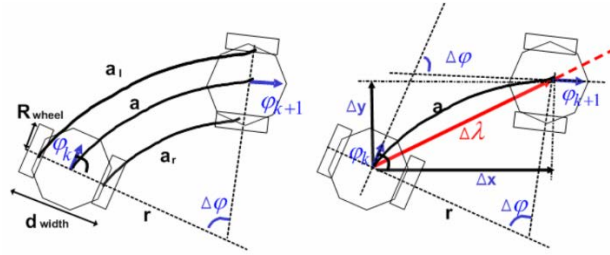


Figure 13. The movement prediction using rotary angles

4.1.2 Wheel encoder odometry

The wheel encoder is the system that converts the angular rate of the rotor into a digital signal. The encoders are attached onto the wheels of the mobile robot; the wheel's rotary angle is measured by the encoder. The rotor encoder generates 48 counts while the wheel rotates 360 degrees. This information on the acceleration of the left and right wheels along with the wheel radius and robot diameter (between the wheels) are used to calculate the mobile robot's yaw angle rate ($\Delta\phi_k$).

$$\Delta\phi_k = \frac{a_{k,l} - a_{k,r}}{d_{width}}$$

According to the cosine law, the mobile robot's position rate ($\Delta\lambda_k$) is calculated. In turn the mobile robot's position rate is transformed into the navigation frame.

$$\begin{aligned}\Delta P_{N_k} &= \Delta\lambda_k \cos\left(\varphi_{k-1} + \frac{\Delta\varphi_k}{2}\right), \\ \Delta P_{E_k} &= \Delta\lambda_k \sin\left(\varphi_{k-1} + \frac{\Delta\varphi_k}{2}\right) \\ \Delta\hat{P}_{N_k} &= \Delta P_{N_k} \cos\theta + \Delta P_{E_k} \sin\phi \cos\theta, \\ \Delta\hat{P}_{E_k} &= \Delta P_{E_k} \cos\phi.\end{aligned}$$

The mobile robot's position and yaw direction is defined as:

$$\begin{aligned}P_{N_{k+1}} &= P_{N_k} + \Delta\hat{P}_{N_k}, \quad P_{E_{k+1}} = P_{E_k} + \Delta\hat{P}_{E_k}, \\ \varphi_{k+1} &= \varphi_k + \Delta\varphi_k.\end{aligned}$$

4.1.3 Dead Reckoning with Kalman filter

Both positioning systems using encoders and that using inertial sensors have accumulative errors, but caused by different reasons. In the method using inertial sensors, integration of the outputs is the cause of the accumulated errors. In the method using encoders, systematic and nonsystematic errors are accumulated through calculation. A linear Kalman filter is used to combine these features.

The specific equations for the time update (predict) and the measurement update (estimate) are:

$$\begin{aligned}
\hat{\mathbf{x}}_k^- &= \mathbf{A}\hat{\mathbf{x}}_{k-1} + \mathbf{B}\mathbf{u}_{k-1}, \\
\mathbf{K}_k^- &= \mathbf{A}\mathbf{K}_{k-1}\mathbf{A}^T + \mathbf{Q}, \\
\mathbf{G}_k &= \mathbf{K}_k^- \mathbf{H}^T (\mathbf{H}\mathbf{K}_k^- \mathbf{H}^T + \mathbf{R})^{-1}, \\
\hat{\mathbf{x}}_k &= \hat{\mathbf{x}}_k^- + \mathbf{G}_k (\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_k^-), \\
\mathbf{K}_k &= (\mathbf{I} - \mathbf{G}_k \mathbf{H}) \mathbf{K}_k^-
\end{aligned}$$

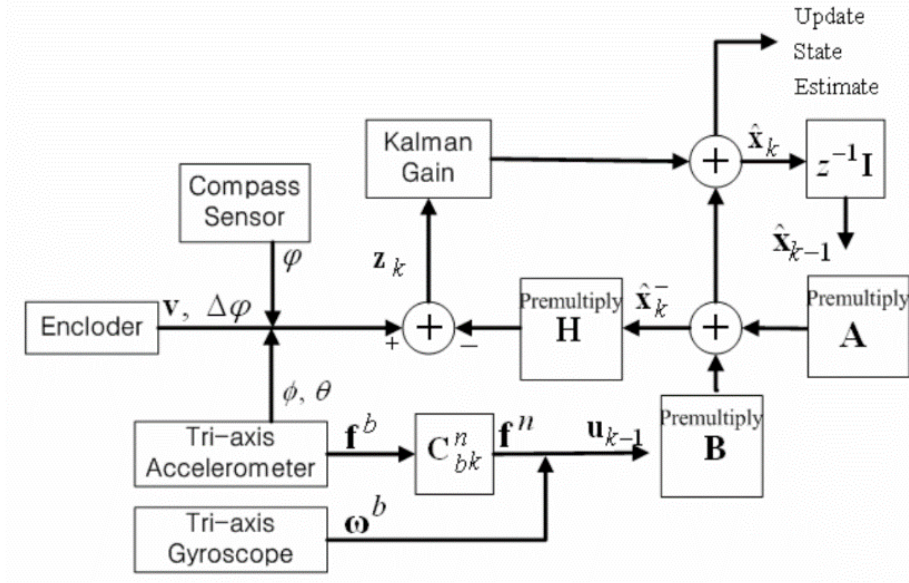


Figure 14. The block diagram of the Kalman filter designed for dead reckoning [15]

The python library FilterPy is used for a complete implementation of the Kalman filter which we plan to use for the dead reckoning algorithm.

4.2 Occupancy Grid Mapping

This algorithm is used to obtain a simplified version of Occupancy Grid Mapping. The goal of algorithm is to obtain a 2D map of the environment. The space is divided into square cells. Instead of the probabilistic approach of a cell being occupied or not, a binary determination is

assumed: the cell is either occupied or not. A partially occupied cell is considered to be an occupied cell and a cell needs to be completely empty to be considered unoccupied. The cells are therefore either null (unexplored), 1 (occupied) or 0 (unoccupied).

4.2.1 Sonar data and uncertainty modeling

The sonar was experimented on from a range of 3 to 100 cm to calculate mean percentage uncertainty (2.975%). This helps in evaluating sonar measurements to be able to select which grid is occupied with the uncertainty in mind. This avoids problems in allocating grids as measured as occupied around the borders when the uncertainty at border helps us reconsider and not overestimate occupancy.

Distance (cm)	Sonar Measurement (cm)	% Uncertainty	Right exit (degree)	Left exit (degree)
3	3.07	2.33	-	-
5	4.89	2.2	-	-
10	10.7	7	20	33
20	19.8	1	27	32
30	29.2	2.67	27	31
40	38.5	3.75	27	28
50	48.5	3	19	28
60	58.4	2.67	27	28
70	68.1	2.71	27	25
80	78	2.5	21	25
90	87.6	2.67	19	30
100	96.8	3.2	21	27

Table 3. Sonar uncertainty

4.2.2 Occupancy grid map method

Given the robot location and pose, the robot having a sense of its body dimensions provides its measurements from its 8 sensors. These measurements update the occupancy grid map through a transform applied between the robots location and the location of the occupied cell. The right and left exit degrees are known to us and the cells that hold the arc that falls within the occupancy grid as considered occupied.

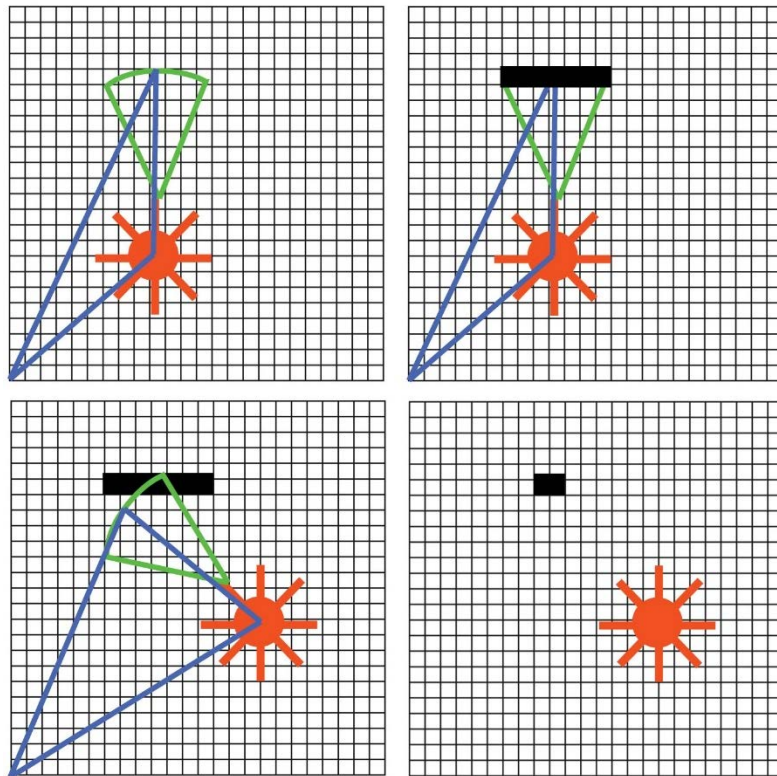


Figure 15. Applying transform over robot position and sonar measurement to recognize occupied cells

The robot may also change its position and update a reading of the same area from a different position, this give us another arc, the union of the cells that are shown as occupied gives us a better guess on which cells are actually occupied. This updates the occupancy of the cells.

4.3 Navigation and Exploration

4.3.1 Motor controller operation

The motor is slowed down using pulse width modulation or PWM and changing the duty cycle because the motors run very fast without PWM and the so readings would have been largely inaccurate. The robot can be turned right by turning the right wheel forward and left wheel backward and vice versa using HIGH or LOW values for the motor controller using its connected GPIO pin.

4.3.2 Exploration Strategy

The exploration would have to ensure that all accessible cells are either visited or sensed to be inaccessible. In the grid, a neighbour is said to be a cell that is immediately adjacent to each other. Therefore, the robot would have 8 possible neighbours, i.e., the neighbours of the cell (i,j) are $(i,j - 1)$, $(i,j + 1)$, $(i - 1,j)$, $(i + 1,j)$, $(i+1, j+1)$, $(i+1, j-1)$, $(i-1, j+1)$ and $(i-1, j-1)$. The choice of the neighbour is made using a clockwise priority. Hence, the movement of the robot are 45 degree angle turns and linear movements equal to the resolution. An important aspect of Occupancy Grid Mapping is defining the resolution, or minimum size of each cell. The smaller the size of the cell, the better is the mapping. However, defining a very small

resolution would require a lot of memory space and would be more complex. Therefore, the resolution is determined to be the diameter of the robot in addition to the turning radius of the robot.

```
1.    function identify_init_cell()
2.    function check_neighbours(cell)
3.    function move_sensing(occupancy, cell, move_direction)
4.    function backtracking(thread, cell)
5.    function mark_track(thread, cell, move_direction)
6.    allocate n by m grid array and store resolution
7.    initialize grid array cell values as NULL
8.    initial_cell = identify_init_cell()
9.    while (present_cell != initial_cell or exists_neighbour(initial_cell)!=visited)
10.        move_direction = check_neighbours(present_cell)
11.        if move_direction is BACKWARD:
12.            present_cell = move_sensing(occupancy, present_cell, move_direction)
13.            mark_track(thread, present_cell, move_direction)
14.        else:
15.            backtracking(thread, present_cell)
```

Figure 16. Pseudocode i for navigation [16]

```

1.    function check_neighbours(cell):
2.        for each (direction: dir)
3.            if occupancy(getcell(dir,cell) == 0) and sensed(neighbours(getcell(dirm
5.                                                    cell))) == NO
4.                return dir
5.    return BACKTRACK

```

Figure 17. Pseudocode ii for navigation [16]

The algorithm makes sure that each available cell is visited, provided the cells are not unreachable which could arise because of obstacles. Therefore, all the available space will be mapped (pseudocode i). This algorithm requires that the cell directly in front of the robot is unoccupied and accessible. The algorithm also optimizes the selection of the neighboring cell in the next iteration (pseudocode ii). This would mean the robot would not visit cells it has already visited before and so time is optimized. The algorithm will only finish if the robot returns to its source cell and has no explorable neighbor, or if every single cell of the grid has been sensed. The robot would not, however, update a cell that it has found to be empty before but is occupied now. This is because the robot has previously passed through this very cell without colliding with anything.

4.4 Graphical Representation

The map is plotted using Matplotlib which turns the n by m matrix into a plot of different colored cells depending on value of the cells. We are plotting the occupied cells as black, the unexplored as grey, and the unoccupied as white.

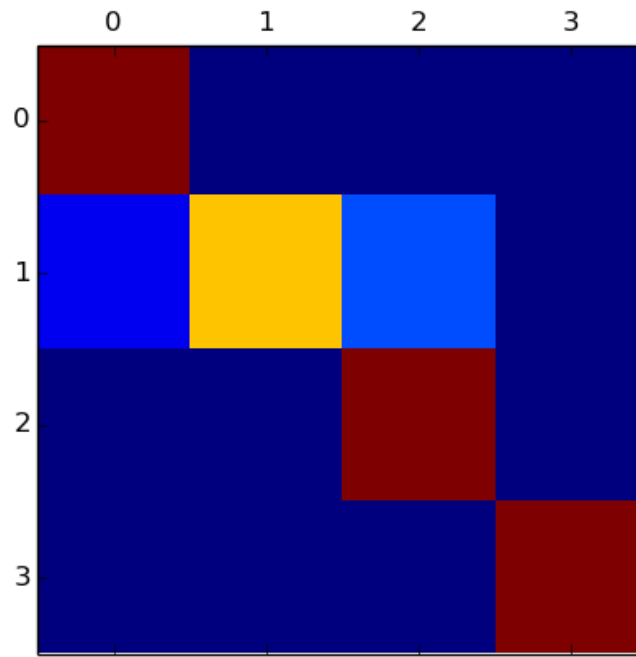


Figure 18. An example of what the Matplotlib library does.

Chapter 5

Experimental Results and Evaluation

The experiment was carried out in a room of size 305 by 459cm, the room was surveyed to make an actual map of the environment that the robot would carry out the SLAM algorithm in.

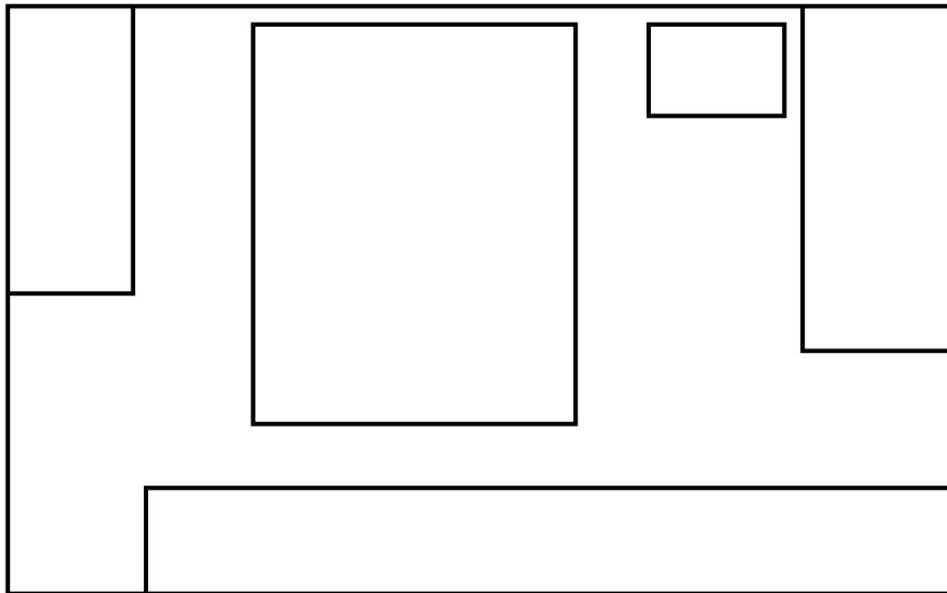


Figure 19. Actual map from survey of room

Using VNC we control the Raspberry Pi remotely using laptop in the same room and wireless network. The main SLAM algorithm runs and terminates to produce the resulting map of the environment. The N by M matrix (in this case 15 by 23 of resolution 20 by 20cm) is plotted using Matplotlib python library.

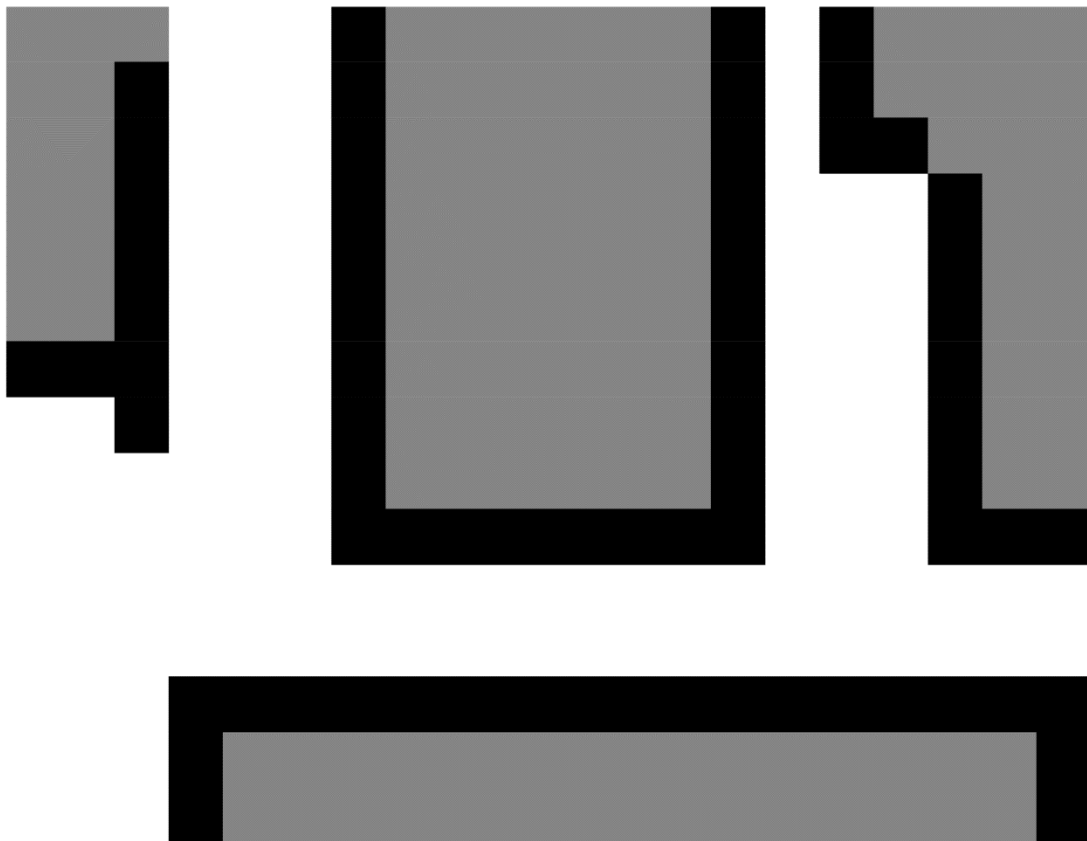


Figure 20. Map obtained from SLAM

The resulting map is approximately the same as the actual map, the black colored cells are occupied as found by the occupancy grid mapping (with cells of value 1), the grey colored cells are unreachable cells (with cell values null), and the white colored cells are unoccupied (with cell values 0). It does have some error in occupancy but the map is within an acceptable range of accuracy given the resolution of the cells.

Chapter 6

Concluding Remarks

6.1 Applications

SLAM is used indoors for various purposes, the map generated could be used for path planning for any applications relating to autonomous mobile robots. Applications for this project relates to any autonomous mobile robots due to the need of a SLAM implementation for a truly autonomous robot. It can be used in smart vacuum cleaners to explore the area and clean it. The map generated would show which areas were cleaned. It can also be used as personal home assistant and for aiding the blind. Search and rescue operations could use SLAM in cases of disasters like chemical fallout in factories and earthquakes to map out an indoor area.

6.2 Limitations and Future Work

Even though using ultrasonic range sensors are inexpensive compared to LiDAR and vision based approaches, this inherently introduces a significant noise into the environment. Another

limitation is the kidnapped robot problem, where the robot comes back to a previously-visited position but with the accumulation of errors, the robot would think that it is a previously unexplored area. This brings us to the point of loop closure, where the robot should recognize a position it has visited before and use the new found information to update the position. The other limitation is to do with a lack of any features within the map, a hybrid approach for both grid based and feature based SLAM can be implemented in the future. Landmark extractions would help in loop closure problem as well.

The implementation is not designed for outdoor use, and future work which fuses relative positioning data as shown with absolute positioning as can be obtained from GPS can make an outdoor implementation work in a similar way. Partial prior map information can also be extracted from Google Maps API which would help simplify the problem computationally.

We would also like to implement SLAM on a much larger scale and do it on several robots together in the future: a swarm of robots. Robots can communicate with each other wirelessly and move towards a more collaborative approach by not exploring an area that another robot has already explored. This would speed up the mapping process.

6.3 Conclusion

In this paper, we have discussed and implemented the simultaneous localization and mapping (SLAM) problem. The purpose of this paper was to implement SLAM in the most inexpensive way possible. The proposed method uses Kalman filter for localization and occupancy grid mapping using several ultrasonic range sensors, inertial measurement units

and wheel encoders. Localization is done using the IMU (3 axis-accelerometer, 3 axis-gyroscope and 2-axis magnetometer) and wheel encoders whereas ultrasonic range sensors are used exclusively for mapping. The entire process is carried out in discrete steps and along with occupancy grid map makes the navigation much simpler and reduces error. The experiment shows the resulting map of the environment within an acceptable threshold of error.

References

- [1] Shoudong Huang & Gamini Dissanayake. Convergence and Consistency Analysis for Extended Kalman Filter Based SLAM. IEEE Transactions on Robotics (Volume: 23, Issue: 5, Oct. 2007)
- [2] Søren Riisgaard and Morten Rufus Blas. SLAM for Dummies: A Tutorial Approach to Simultaneous Localization and Mapping.
- [3] R. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. Intl. J. of Robotics Research, 5(4):56–68, 1987.
- [4] J. Guivant and E. Nebot. Optimization of the simultaneous localization and map building algorithm for real time implementation. IEEE Trans. Robot. Automat., 17(3):242–257, June 2001.
- [5] Michael Montemerlo, Sebastian Thrun, Daphne Koller and Ben Wegbreit. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem
- [6] R. Smith, M. Self & P. Cheeseman. Estimating Uncertain Spatial Relationships in Robotics.
- [7] C. Estrada, J. Neira, and J.D. Tardos. Hierarchical SLAM: Real-time accurate mapping of large environments. IEEE Trans. Robotics, 21(4):588–596, Aug 2005.
- [8] Butterly, Patrick, and Joshua Daly. "Implementing Odometry And SLAM Algorithms On A Raspberry Pi To Drive A Rover". Academia.edu. N.p., 2017. Web. May 2014.
- [9] Theophil, Sebastian. "Stheophil/Mappingrover". GitHub. N.p., 2017. Web. 19 Apr. 2017.

- [10] "Raspberry Pi 3 Model B - Raspberry Pi", Raspberry Pi, 2017. [Online]. Available: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>. [Accessed: 15- Apr- 2017].
- [11] "Ultrasonic Sensor - HC-SR04 - SEN-13959 - SparkFun Electronics", Sparkfun.com, 2017. [Online]. Available: <https://www.sparkfun.com/products/13959>. [Accessed: 15- Apr- 2017].
- [12] InvenSense, "MPU-9255 Product Specification," September. 2014
[Revision 1.0].
- [13] "Pololu - Encoder for Pololu Wheel 42x19mm", Pololu.com, 2017. [Online]. Available: <https://www.pololu.com/product/1217>. [Accessed: 15- Apr- 2017].
- [14] "Pololu - 30:1 Micro Metal Gearmotor HP 6V", Pololu.com, 2017. [Online]. Available: <https://www.pololu.com/product/1093>. [Accessed: 15- Apr- 2017].
- [15] B.-S. Cho, W. Moon, W.-J. Seo, and K.-R. Baek, "A dead reckoning localization system for mobile robots using inertial sensors and wheel revolution encoding," J. Mech. Sci. Technol., vol. 25, no. 11, pp. 2907–2917, Nov. 2011.
- [16] Gonzalez-Arjona, David et al. "Simplified Occupancy Grid Indoor Mapping Optimized For Low-Cost Robots". N.p., 2013. Print.