TECHNICAL UNIVERSITY OF CRETE

Technical University of Crete
School of Electrical and Computer Engineering
Intelligent Systems Laboratory

Design and Implementation of an Autonomous Robotic Vehicle for Mapping

Diploma Thesis by

Sotirios S. Dimitras

Submitted in partial fulfillment of the requirements for the Diploma of Electrical and Computer Engineer at the Technical University of Crete

February 2019

To be defended publicly
against the thesis committee consisting of

Associate Professor Michail G. Lagoudakis
School of Electrical and Computer Engineering

Associate Professor Georgios Chalkiadakis
School of Electrical and Computer Engineering

Professor Apostolos Dollas
School of Electrical and Computer Engineering

Σχεδίαση και Υλοποίηση Αυτόνομου Ρομποτικού Οχήματος για Χαρτογράφηση

Acknowledgements

A few colleagues have been participated and helped in the completion of this thesis document, and every one of them with their unique way. It feels appropriate to dedicate this part of my thesis to them in order to express my acknowledgements and gratitude.

- Firstly, I wouldd like to thank my supervisor Assoc. Professor Michail G. Lagoudakis for his support, his advice and guidance to the completion of this thesis document.
- Prof. Georgios Chalkiadakis, for being a member of the thesis committee and accepting to evaluate my work.
- Prof. Apostolos Dollas, for the flawless cooperation, his guidance on the hardware design and his support in every aspect of this thesis.
- Richard Pine, a leading writer and lecturer on modern Irish culture and literature, for his support and his help in writing this thesis document in English.
- And finally my family and friends, who supported me during this thesis and backed up my choice.

List of Figures

Contents

Abstract

Robot mapping is the process of building a representation (a map) of an unknown environment using a mobile robot. Robot Localization is the process of estimating the pose of a robot within a map. In real-world robot applications, these two tasks are typically performed in parallel. In this thesis, we present a low-cost and open-source mobile autonomous robot to perform Simultaneous Localization and Mapping (SLAM).Our robot has to have the ability to navigate freely in a room, with no outside guidance while avoiding obstacles and covering as much floor space as possible. The robot is being controlled using an Arduino micro-controller. It uses two ultrasonic sensors for distance measuring and one MPU (Motion Processing Unit) for sensing changes in the robot's position and orientation. The data collected by the sensors are transmitted wirelessly to a remote PC (work station) via a Bluetooth module. A developed software application on the PC processes the data and builds a probabilistic map of the room, while displaying the robot's position and trajectory. As a result, an intelligent and fully autonomous SLAM robot was designed with integrated systems for its control and navigation. The robot is able to cover almost all space of an unknown room without colliding with any object or wall, producing the desired map of the room at the same time.

Περίληψη

Η ρομποτική χαρτογράφηση είναι η διαδικασία δημιουργίας μιας αναπαράστασης (χάρτη) ενός άγνωστουπεριβάλλοντος χρησιμοποιώντας ένα κινητό ρομπότ. Ο ρομποτικός εντοπισμός είναι η διαδικασία εκτίμησηςτης θέσης ενός ρομπότ μέσα σε έναν χάρτη. Στις ρομποτικές εφαρμογές του πραγματικού κόσμου, αυτές οιδύο εργασίες εκτελούνται συνήθως παράλληλα. Στην παρούσα διπλωματική εργασία, παρουσιάζουμε ένακινητό αυτόνομο ρομπότ χαμηλού κόστους και ανοικτού κώδικα για την εκτέλεση ταυτόχρονου εντοπισμούκαι χαρτογράφησης (Simultaneous Localization and Mapping – SLAM). Το ρομπότ μας έχει τη δυνατότητα ναπεριηγείται ελεύθερα μέσα σε ένα δωμάτιο, χωρίς εξωτερική καθοδήγηση, αποφεύγοντας τα εμπόδια καικαλύπτοντας όσο το δυνατόν περισσότερο χώρο δαπέδου. Το ρομπότ ελέγχεται χρησιμοποιώνταςμικροελεγκτή Arduino. Χρησιμοποιεί δύο αισθητήρες υπερήχων για τη μέτρηση αποστάσεων και ένα MPU(Motion Processing Unit) για την ανίχνευση αλλαγών στη θέση και τον προσανατολισμό του ρομπότ. Ταδεδομένα που συλλέγονται από τους αισθητήρες μεταδίδονται ασύρματα σε απομακρυσμένο υπολογιστή(σταθμό εργασίας) μέσω μονάδας Bluetooth. Μια αναπτυγμένη εφαρμογή λογισμικού στον υπολογιστήεπεξεργάζεται τα δεδομένα και δημιουργεί έναν πιθανοτικό χάρτη του χώρου, ενώ εμφανίζει τη θέση και τηντροχιά του ρομπότ. Ως αποτέλεσμα, σχεδιάστηκε και υλοποιήθηκε ένα ευφυές και πλήρως αυτόνομο ρομπότγια SLAM με ενσωματωμένα συστήματα ελέγχου και πλοήγησης. Το ρομπότ μπορεί να καλύψει σχεδόν όλοτο χώρο ενός άγνωστου χώρου, χωρίς να συγκρουστεί με οποιοδήποτε αντικείμενο ή τοίχο, παράγονταςταυτόχρονα τον επιθυμητό χάρτη του χώρου.

Chapter 1

Introduction

In robotic mapping and navigation, simultaneous localization and mapping (SLAM) is the computational process of acquiring a map of an unknown environment with a moving robot, whilst simultaneously keeping track of the robot's location relative to this map. A SLAM problem was first introduced by R.C. Smith and P. Cheeseman on the representation and estimation of spatial uncertainty in the 1980s. Since then, this problem has received very considerable attention from the scientific community, and a flurry of new algorithms and techniques have emerged.

The SLAM problem was initially compared to a chicken and egg problem, as a good map is needed for localization while an accurate pose estimation is needed to build a map. Those two problems can not be solved independently of each other. Before a robot can tell what the environment looks like given a set of observations, it is crucial to know from which locations these observations have been made. Meanwhile it is hard to estimate the current position of the vehicle without the environment's map. Hence SLAM robots have to deal with situations where they lack global positioning.

In order to face the difficulties of their location unawareness, robots have to rely on sensors in order to estimate their position, relatively to the environment, (e.g. odometry, landmarks, inertial navigation). Such sensors accumulate error over time, making the problem of constructing a map a challenging one.

This thesis details the system design process and the final design of one such fully autonomous robot. This approach will allow the educational process of autonomous agents to become accessible and inexpensive, providing students with a hands-on experience on a low cost platform.

The system was extensively tested on various mazes and has been proven reliable. The results were somewhat surprising, as a simple implementation of motion succeeds in driving the robot, covering the mapping area astonishing well. At this

point, it is crucial to mention that this project was not aiming to provide any novelty or new findings in the SLAM domain, rather than gaining a hands on experience in robotics and autonomous agents.

The rest of this thesis is organized as follows. In the second chapter, the concept of Occupancy Grid Mapping is explained in detail and basic background theory about the components used is presented. In the third chapter, related techniques in the concept of robotic mapping are listed and briefly explained. The fourth chapter briefly describes our approach. The fifth chapter specifies the system’s design in means of the hardware. The sixth chapter describes the system’s design concerning the software development. The seventh chapter refers to the experiments conducted for the proper functionality of the platform. The final chapter concludes the present thesis and proposes future improvements.

Chapter 2

Background

Bayesian theory is a branch of mathematical probability theory and statistics that describes the uncertainty of an event by incorporating prior knowledge of conditions, that might be related to the event, and observational evidence. One of the many applications of Bayes' theorem is the *Bayesian inference*, a practical approach to *statistical inference*.

In *Bayesian inference*, all of the uncertainties (including parameters and states) are treated as *random variables*. The objective of Bayesian inference is to infer the conditional probability, using prior knowledge of a given set of finite observations.

Bayesian filtering aims to apply Bayesian statistics and Bayes' rule to probabilistic inference problems. In our approach, we use this type of filtering to estimate our robot's state[1].

2.1 Introduction to Bayesian Filtering

In robotics, the Bayes filter algorithm is generally used for estimating the probabilities of multiple beliefs[2]. This algorithm calculates the belief distribution from observations and control data, in a manner to allow a robot to infer its position and orientation. The Bayes filter is a recursive algorithm, that is because the robot's belief at time t is being calculated from the belief the robot had at time t-1. Essentially, Bayes filter allows robots to continuously update their beliefs by recursively calculating them.

The *Bayes filter algorithm* consists of two essential steps. The first step refers to the algorithm's update step using control data (u). This update step is also called *control update* or *prediction*. The second step of the Bayes filter is called the

1. State estimation addresses the problem of inferring knowledge about quantities from sensor data that are indirectly observable, but can be inferred.
2. The *belief* is the robot's estimation of its current state, a probability density function distributed over the state space.

measurement update, or *innovation*. In this step, the algorithm multiplies the current belief by the probability that the measurement (z) may have been observed.

2.1.1 Binary Bayes Filters with Static State

The Binary Bayes filter constitutes a very special case of the optimal Bayesian filter. The state X to be estimated is *static*, and state space[3] is discrete and binary. This type of filter is suitable for occupancy grid mapping algorithms where each grid's cell can be either occupied or free. Since the state X is static, the belief is a function of the measurements:

$$bel_t(X) \;=\; p(X \mid z_{1:t}, u_{1:t}) \;=\; p(X \mid z_{1:t}) \tag{2.1}$$

Under the Markov assumption and by applying the Bayes theorem twice in equation 2.1, we obtain

$$\begin{aligned} p(X \mid z_{1:t-1}) &= \frac{p(z_t \mid X, z_{1:t-1})\, p(X \mid z_{1:t-1})}{p(z_t \mid z_{1:t-1})} = \frac{p(z_t \mid X)\, p(X \mid z_{1:t-1})}{p(z_t \mid z_{1:t-1})} = \\ &= \frac{p(X \mid z_t)\, p(z_t)\, p(X \mid z_{1:t-1})}{p(X)\, p(z_t \mid z_{1:t-1})} \end{aligned} \tag{2.2}$$

Similarly, for the negate event we get

$$bel_t(\neg X) \;=\; p(\neg X \mid z_{1:t-1}) = \frac{p(\neg X \mid z_t)\, p(z_t)\, p(\neg X \mid z_{1:t-1})}{p(\neg X)\, p(z_t \mid z_{1:t-1})} \tag{2.3}$$

Generally, the belief estimation problem is represented as a *log-odds ratio*. That is to avoid truncation problems which arise from probabilities close to 0 or 1. The *log-odds* ratio of state X is defined as the logarithm of the probability of the event divided by the probability of its negate.

$$l_t(X) \;=\; \log \frac{p(X \mid z_1{:}t)}{p(\neg X \mid z_{1:t})} = \log \frac{p(X \mid z_1{:}t)}{1 - p(X \mid z_1{:}t)} \tag{2.3}$$

By using the equations 2.2 and 2.3 in 2.4, we get

$$\begin{aligned} l_t(X) \;&=\; \log\Big(\frac{p(X \mid z_t)}{p(\neg X \mid z_t)} \frac{p(X \mid z_{1:t-1})}{p(\neg X \mid z_{1:t-1})} \frac{p(\neg X)}{p(X)}\Big) \\ &=\; \log\Big(\frac{p(X \mid z_t)}{1 - p(X \mid z_t)} \frac{p(X \mid z_{1:t-1})}{1 - p(X \mid z_{1:t-1})} \frac{1 - p(X)}{p(X)}\Big) \\ &=\; \log \frac{p(X \mid z_t)}{1 - p(X \mid z_t)} + \log \frac{p(X \mid z_{1:t-1})}{1 - p(X \mid z_{1:t-1})} + \log \frac{1 - p(X)}{p(X)} \\ &=\; \overbrace{\log \frac{p(X \mid z_t)}{1 - p(X \mid z_t)}}^{Inverse\ \ Sensor\ \ Model} + l_{t-1}(X) - \overbrace{\log \frac{p(X)}{1 - p(X)}}^{Prior} \end{aligned} \tag{2.5}$$

3. The **state space** of a dynamical system is the set of all possible states of the system. Each coordinate is a **state** variable, and the values of all the **state** variables in total describes the **state** of the system.

1: **Algorithm binary_Bayes_filter(l_{t-1}, z_t)**:

2: $$l_t = l_{t-1} + \log\frac{p(X|z_t)}{1-p(X|z_t)} - \log\frac{p(x)}{1-p(x)}$$

3: return l_t

Table 2.1 The binary Bayes filter in log-odds form with an inverse measurement model. Here l_t is the log odds of the posterior belief over a binary state variable that does not change over time (source: Probabilistic Robotics)

As we observe the final form of the log-odds ratio representation in equation 2.5, we reach the conclusion that the update algorithm is additive. Moreover, this particular binary Bayes filter uses an *inverse sensor model* $p(X|z_t)$ instead of the most commonly used forward model $p(z_t|X)$. Note that the *inverse sensor model* will be discussed in the following subsection. Table 2.1 provides a basic example of the update algorithm. In order for the updating algorithm to start the recursion, the constant l_0 is needed, which is specified with the help of the prior probability according to

$$l_0(X) = \log\frac{p(X)}{1-p(X)} \tag{2.6}$$

If there is no available knowledge concerning the prior state, complete ignorance can be expressed by setting the prior $p(X)=0.5$.

2.2 Occupancy Grid Mapping

“A map is a visual representation of an area - a symbolic depiction highlighting relationships between elements of that space such as objects, regions, and themes”[4].

Mapping is the processes in which selected features of an area are extracted, through sensors measurements, and stored in a data structure. This data structure is called a map. Such a map can represent the environment either three- or two-dimensionally.

In this Thesis we are going to use two-dimensional map representations, and a specific category of maps, called *occupancy grid maps*.

4. https://en.wikipedia.org/wiki/Map

Occupancy grid maps address the problem of generating consistent maps from noisy and uncertain measurement data under the assumption that the robot 's pose is known. Occupancy grid is a type of map that represents the environment as a rasterized structure where each cell corresponds to a binary random variable[5]. This value holds the probability of an obstacle's presence at that location in the environment. Each cell's likelihood of occupation ranges from zero to one, as is the case with all probabilities. The zero value corresponds to the obstacle 's absence, while the value one is for its presence.

The goal of any occupancy grid mapping algorithm is to compute approximate the posterior probability over maps based on accessible data, as illustrated in equation 2.7.

$$p(m|z_{1:t}, x_{1:t}) \tag{2.7}$$

The convention we use to explain the mathematical foundations is as follows. The *m* stands for the map,while $z_{1:t}$ reflects the set of all measurements and $x_{1:t}$ the set of all the robot's poses up to time t. Due to the assumption that the robot's pose is known, the control data $u_{1:t}$, which are responsible for the path, play no role in occupancy grid maps and are therefore omitted.

Occupancy grid maps represent the map as a finitely, fine-grained grid over the continuous space of location in the environment. Let us assume m_i denotes the grid cell with index i.

$$m = \{m_i\} \quad \text{or} \quad m = \sum_i m_i \tag{2.8}$$

Each m_i has a binary occupational probability value $p(m_i)$ attached to it, which specifies whether the cell is occupied or free. The computational problem with estimating the posterior in equation 2.7 is the dimensionality of the problem. A detailed occupancy grid map may consist of several thousand of individual cells. Assuming we need tens of thousands of cells to represent our environment space, while taking into consideration that each cell holds a binary value (0 or 1), the number of different maps defined in this space equals to $2^{10.000}$. Thus, calculating a posterior probability for all such maps is an infeasible approach.

The standard occupancy grid approach is to break down the problem into smaller problems estimating the occupational probability of each individual cell.

$$p(m_i|z_{1:t}, x_{1:t}) \tag{2.9}$$

5. **Random Variable** is a variable whose possible values are outcomes of a random phenomenon. A random variable is defined as a function that maps the outcomes of unpredictable processes to numerical quantities (labels), typically real numbers.

Eventually, each of these estimation problems is a binary problem. This decomposition is convenient as it gets rid of the high-dimensional posterior. However, it introduces a new problem. In particular, it does not enable us to model possible dependencies between neighboring cells. Hence, the posterior over a map is approximated as the product of probabilities of all map' s cells.

$$p(m|z_{1:t},x_{1:t})=\prod_i p(m_i|z_{1:t},x_{1:t}) \tag{2.10}$$

Due to this factorization, the estimation of each grid cell's occupancy becomes a binary estimation problem with static state. *Binary Bayes filter with static state* is a suitable algorithm for determining these estimations. The algorithm in Table 2.2 applies this filter to the occupancy grid mapping problem. A noteworthy property of this algorithm is its use of *log-odds* representation ($l_{t,i}$) of occupancy:

$$l_{t,i} = \log\frac{p(m_i|z_{1:t},x_{1:t})}{1-p(m_i|z_{1:t},x_{1:t})} \tag{2.11}$$

The use of *log-odds* representation greatly benefits the algorithm because its numerical advantages in cases of small probabilities. Note that the probabilities can easily be recovered from the log-odds ration

$$p(m_i|z_{1:t},x_{1:t}) = 1-\frac{1}{1+e^{l_{t,i}}} \tag{2.12}$$

The basic functionality of the occupancy_grid_mapping algorithm in Table 2.2 is quite simple: it loops through every single grid cell *i* and determine whether this cell belongs to the perceptual field of the measurement z_t. Cells which falls into the *sensor cone,* have their occupancy probability value updated by virtue of the function *inverse_sensor_model,* while the value of the other cells remain unchanged.

```
1.  Algorithm occupancy_grid_mapping({l_{t-1,i}}, x_t, z_t):
2.      for all cells m_i do
3.          if m_i in perceptual field of z_t then
4.              l_{t,i} = l_{t-1,i} + inverse_sensor_model(m_i, x_t, z_t) - l_0
5.          else
6.              l_{t,i} = l_{t-1,i}
7.          endif
8.      endfor
9.      return{l_{t,i}}
```

Table 2.2 The occupancy grid algorithm, a version of the binary Bayes filter in Table 2.1. (source: Probabilistic Robotics).

The *inverse_sensor_model* function implements the inverse measurement model of the form $p(m_i|z_t, x_t)$. It specifies a distribution over the binary state variable m_i, which relates a certain cell, as a function of the measurement z_t and pose x_t. This is convenient in situations where *measurement space* is much more complex that the *state space*. A basic function for a range finder is given in Table 2.3 and illustrated in Figure 2.1 a & b.

1: **Algorithm inverse_range_sensor_model**(i, x_t, z_t)**:**
2: Let x_i, y_i be the center-of-mass of $\mathbf{m}_i$
3: $r = \sqrt{(x_i - x)^2 + (y_i - y)^2}$
4: $\phi = \text{atan2}(y_i - y, x_i - x) - \theta$
5: $k = \text{argmin}_j \, |\phi - \theta_{j,\text{sens}}|$
6: if $r > \min(z_{\max}, z_t^k + \alpha/2)$ or $|\phi - \theta_{k,\text{sens}}| > \beta/2$ then
7: return l_0
8: if $z_t^k < z_{\max}$ and $|r - z_{\max}| < \alpha/2$
9: return l_{occ}
10: if $r \leq z_t^k$
11: return l_{free}
12: endif

Table 2.3 A simple inverse measurement model for robots equipped with range finders. Here α is the thickness of obstacles, and β the width of the sensor beam. The values l_{occ} and l_{free} denote the amount of evidence a reading carries for the two different cases. (source: Probabilistic Robotics)

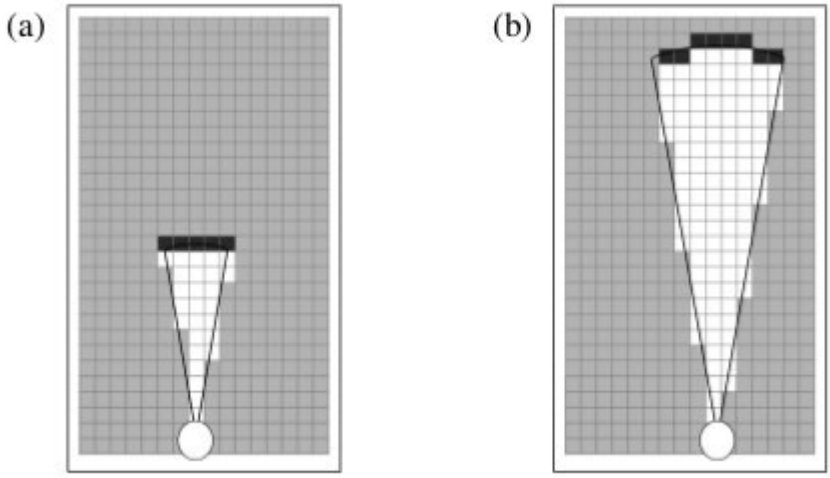

Figure 2.1 Two examples of our inverse measurement model for two different measurements ranges. The darkness of each cell corresponds to the likelihood of occupancy. (source: Probabilistic Robotics)

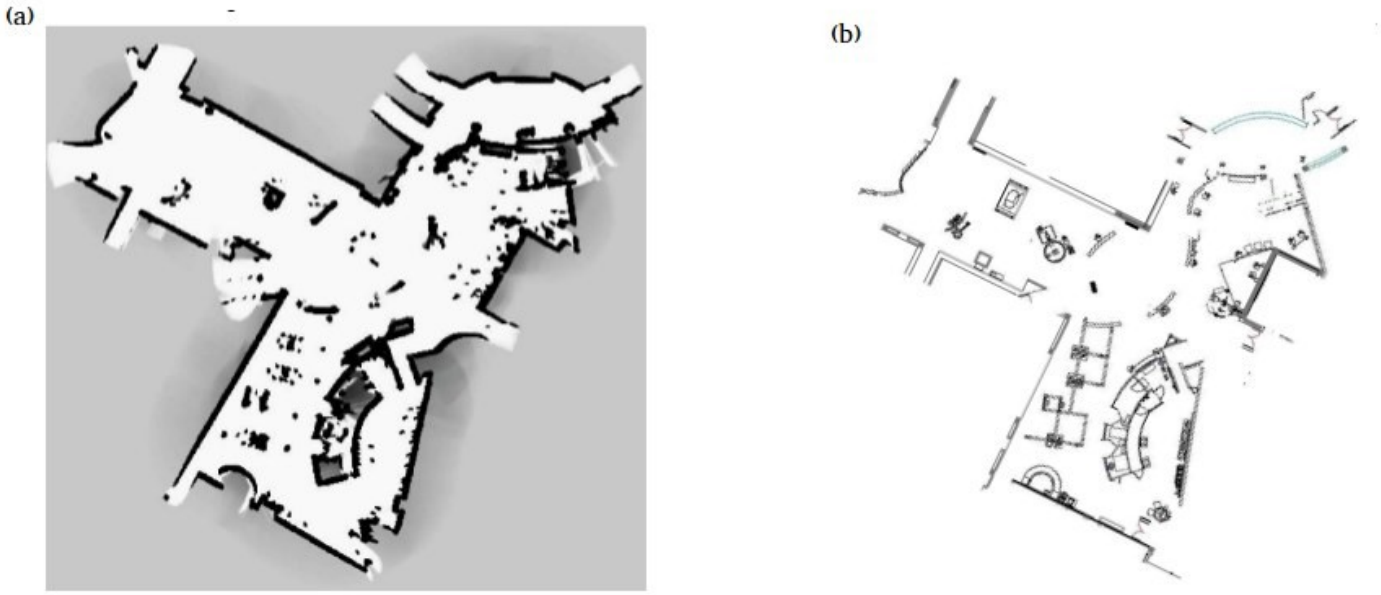

Figure 2.2 (a) Occupancy grid map and (b) architectural blue-print of a large open exhibit space. Notice that the blue-print is inaccurate in certain places. (source: Probabilistic Robotics).

Figure 2.2 shows an example of an occupancy grid map next to the architectural blue-print of the environment space. The map was constructed using a robot's measurements which were acquired while the robot was performing SLAM. The gray-scale indicates the posterior probability: Black corresponds with high probability to an occupied cell, while White corresponds with high probability to free cell. The gray background color represents the prior probability.

As it appears from Figure 2.2, an occupancy grid map shows all structural elements, as well as obstacles which are observable by the sensor. These features make occupancy grid maps an appropriate way of mapping while dealing with SLAM problems.

2.3 Microcontrollers

A microcontroller (MCU for microcontroller Unit) is a compact integrated circuit designed for embedded systems. MCU is similar to, but less sophisticated than, a *system on chip (SoC)*[6]. A typical microcontroller includes one or more processor

6. ***A system on a chip (SoC)*** combines the required electronic circuits of various computer components onto a single, integrated chip (IC). SoC is a complete electronic substrate system that may contain analog, digital, mixed-signal or radio frequency functions. Its components usually include a graphical processing unit (GPU), a central processing unit (CPU) that may be multi-core, and system memory (RAM).

cores (CPUs) along with memory and programmable input/output (I/O) peripherals on a single chip. Microcontrollers are used in autonomous controllers products and devices, such as automobile engine control systems, remote controls, vehicles, office machines and medical devices among other embedded systems. The most important advantage of a microcontroller is they can keep the cost of an integrated system at very low levels.

A microcontroller's processor will vary by application. Options range from the simple 4-bit, 8-bit or 16-bit processors to more complex 32-bit or 64-bit processors. In terms of memory, microcontrollers can use random access memory (RAM), flash memory, EPROM or EEPROM. Note that MCUs are dedicated to one task and run one specific program which is stored in read-only memory (ROM). Their architecture design can be based on Harvard architecture[7] or Von Neumann architecture[8], which differ on the methods of exchanging data between the processor and the memory.

MCUs feature input and output (I/O) pins to implement peripheral functions such as *real-time clock (RTC), sychronous/asychronous receiver transmitter (USART), analog-to-digital converters and universal serial bus (USB) connectivity*. Sensors and other modules/boards can also be attached to microcontrollers through I/O pins.

In our thesis, we use an open-sourse platform, called Arduino, to control our robot. Further information about this board will be discussed on the chapter 4.1.

Figure 2.3 Types of AVR Microcontrollers

7. In **Harvard architecture**, the data bus and the instruction set are separate, allowing for simultaneous transfers.
8. In **Von Neumann architecture**, one bus is used for both data and instruction set.

2.4 Sensors

Sensors are important in Robotics for a number of reasons. To begin with, sensors allow the robot to become more autonomous because it can perceive its own environment and through programming it can make decisions based on what it perceives. Sensors are also an important part of robots for remote operation. That is because they make decisions on what operation the robot should do next.

Two main sensor categories are used in our project: the perception sensors and the navigation sensors. In the following subsections, we will discuss some important information about their basic working principles.

2.4.1 Ultrasonic Sensors

Ultrasonic sensors are type of sensors which are mostly used for measuring distance and detecting an object. The sensor operates by emitting an ultrasonic wave (high frequency audio signal), which will reflect any object in front of the sensor. This reflected signal is detected by the sensor. Then, using the time between emission and reception of the audio signal, we can calculate the distance of any object.

Ultrasonic wave or *Ultrasound* is sound waves with frequencies higher than the upper audible limit of human hearing. A human has the ability to hear sounds with frequencies varying in range from 20 Hz to 20 KHz, while the sound waves emitted by the sensor have 40 KHz of frequency.

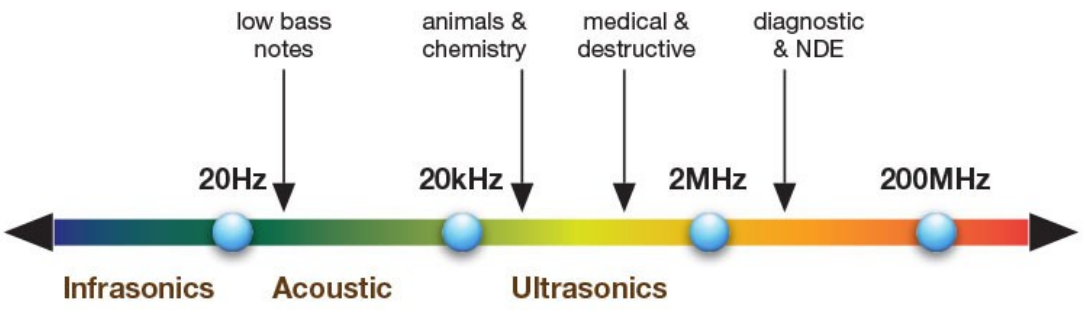

Figure 2.4 Ultrasonic Range Diagram

Despite the fact that ultrasonics are capable sensors with great fit for many applications, we understand that they have limitations which make them not suited for every application. Since ultrasonics operate using audio an signal, they are completely nonfunctional in a vacuum as there is no air for the sound to travel through. These sensors are also not designed for underwater applications. Due to the way they function, their sensing accuracy could be affected by soft materials and changes in temperature of 5-10 degrees or more. Soft fabric absorbs sound waves making it hard for the sensor to receive any reflected signal. Finally, the last sensor's flaw is the limited detection range.

Ultrasonic sensors are active; they only require power so to generate and transmit the ultrasound waves for performing their tasks. This would mean that they could also pick up signals from previous scan measurements, or even from other ultrasonic sensors.

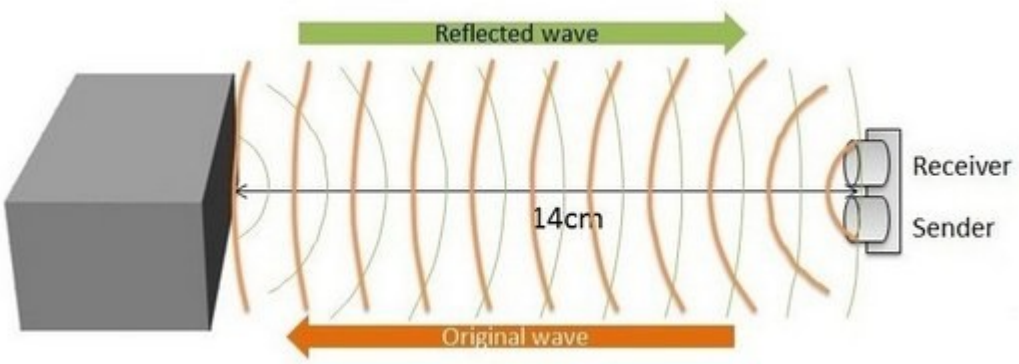

Figure 2.5 Ultrasonic sensor emitting and receiving sound waves

2.4.2 Inertial Measurement Unit (IMU)

An Inertial Measurement Unit, commonly knows as IMU, is an electronic device that measures and reports orientation, velocity and gravitational forces, using a combination of accelerometers, gyroscopes and magnetometers. It is a self-contained system that measures linear and angular motion usually with a triad of gyroscopes and triad of accelerometers. IMUs are a main component of the inertial navigation systems used in aircraft, unmanned aerial vehicles (UAVs) and other unmanned systems, as well as missiles and even satellites. In a navigation system,

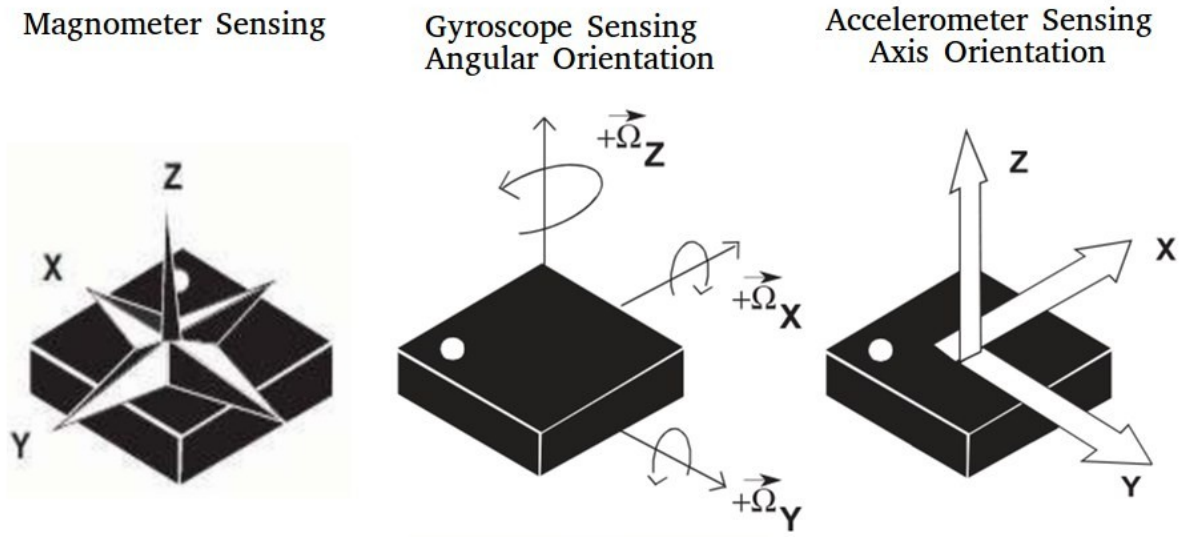

Figure 2.6 The nine Degrees of Freedom of an IMU (Source: Sparkfun)

the data reported by the IMU are fed into a processor which calculates the instantaneous position, velocity, orientation, and direction of movement.

IMU sensors available on the market are of various types and shapes. So, the user can select what type, size and shape. The IMU can be selected from its *degrees of freedom* (DOF) that are being developed by the manufacturer. In our project we chose an IMU which is capable of measuring nine degrees of freedom. This includes the measurement of linear motion over three perpendicular axes (*surge*, *heave*, and *sway*), as well as rotational movement about three perpendicular axes (*roll*, *pitch*, and *yaw*) and magnetic field strength over the same three axes of rotation. This yields nine independent measurements that together define the movement of our robot. In Figure 2.6 we can see the 9 Degrees of Freedom of the IMU sensor we use.

Chapter 3

Related work

Since 1986, when Peter Cheeseman, Jim Crowley and Hugh-Durrant Whyte talked about about the topic of simultaneous localization and mapping applying probability, it has been a very active field in robotic research. The creation of SLAM resulted in a huge number of works devoted to finding suitable techniques able to deal with robots performing exploration in unknown environments. Multiple mapping techniques have been developed since then, both for indoor and outdoor environments. These techniques can be roughly classified according to the map representation and the estimation technique.

There are two main map representation methods. The most popular is the *Occupancy Grid*, which is also used in our approach. As we have already discussed, grid based approaches are computationally expensive and require lots of memory. The second map representation is the *Feature based*, in which the map model is expressed by means of landmarks in the environment. This method gained its popularity due to its compactness, which is an advantage in terms of memory consumption and processing speed. On the other hand, such systems rely on predefined knowledge about structures in the environment. This clearly limits the robot's field of action.

Since robotic mapping constitutes an ongoing research area, a number of state estimation methods have been developed and are still developing. These algorithms basically differ in the sensors they use to collect their observations and how they make use of the observed information. They may also differ on the filter they are based on (e.g. the *Kalman filter* or the *Particle filter)*. Among many approaches, we chose to present some of the major and most widespread solutions available for a SLAM state estimation problem.

The ***Gaussian Filters*** constitute the earliest tractable implementations of the *Bayes Filter* for continuous spaces. Despite the limitations, Gaussian Filters are a popular family of techniques to date. The idea behind the use of the Gaussian Filters is that the beliefs are represented by multivariate normal distributions:

$$p(x) = det(2\pi\Sigma)^{\frac{-1}{2}} \exp\frac{-1}{2}(\chi-\lambda)^T \Sigma^{-1}(\chi-\lambda) \tag{3.1}$$

where λ is the mean and Σ is the covariance.

Two of the most popular and best studied methods for implementing Bayes filter are the *Kalman Filter* and the *Extended Kalman Filter.* The ***Kalman filter (KF)***, also known as *Linear Quadratic Estimation* (LQE), is a recursive algorithm that tries to estimate the state $x \in \mathbb{R}^N$ of a discrete-time controlled process that is governed by a linear stochastic difference equation. The KF makes use of a series of measurements observed over time, containing statistical noise and other inaccuracies, to produce estimations for random variables by calculating a joint probability distribution over the variables for each time-frame. In the SLAM context, the variable values to be estimated consist of the robot's position and landmarks' locations. The filter is named after its creator, Rudolf E. Kalman.

The ***Extended Kalman filter (EKF)*** is a non linear version of the *Kalman Filter,* which overcomes the linearity assumption about an estimate of the current mean and covariance. This model constitutes a generalization of the linear *Gaussian model* underlying *Kalman filters.* However, the distribution ceases to be Gaussian when non linear functions are used. Hence, the distribution update step does not possess a closed form solution. Therefore, the EKF calculates an approximation of the true distribution. Thus, the EKF inherits from Kalman filters the basic belief representation, but it differs from it as this belief is only approximate. It is crucial to say that EKF based algorithms are typically feature based and use maximum likelihood algorithm for data association. The effectiveness of the EKF approaches results from the fact that they estimate a fully correlated posterior about landmark maps and robot poses. On the other hand, such methods pose strong assumptions about the robot's motion model and sensor's noise, which make the approaches vulnerable to misconceptions. In addition, it has to be noted that each and every landmark has to be uniquely identifiable.

An optimized version of Extended Kalman Filter was proposed by Jose Guivant and Eduardo Net. The introduces the ***Compressed Extended Kalman Filter (CEKF)***. This approach reduces the computational complexity by dividing the system vector into two parts: the active local state vector and the others. Thus, the EKF fully updates only the local state vector at each step, while the necessary information for updating the other states is compressed into some auxiliary coefficient matrices. This increases the algorithm's efficiency without diminishing the accuracy that characterizes full SLAM algorithms.

Alternative to Kalman techniques, there are the Particle Filters. ***Particle filters (PF)*** or ***Sequential Monte Carlo (CMS)*** are a set of Monte Carlo algorithms used to solve filtering problems in Bayesian statistical inference. The Particle filter is designed for random variables connected in a Markov Chain, where the system consists of hidden and observable variables. The observation variables are connected to the hidden variables (states) by some function form that is known. A Probability Distribution Function (PDF) is used to represent tha state information.

DP-SLAM constitutes a particle filter based, on-line algorithm that generates grid maps. Its purpose is to achieve accurate SLAM without landmarks, while reducing the extensive use of computer resources by avoiding the successive copy of maps per each particle generated. It works by maintaining a joint probability over maps and robot poses. This allows the algorithm to maintain uncertainty about the map over multiple time steps until ambiguities can be resolved, while preventing errors in the map from accumulating over time.

The ***Fast SLAM*** is an alternative method for stochastic mapping and localization. *Fast SLAM* constitutes an interesting method as it is a hybrid algorithm; by this means it uses both Particle and Kalman filters. The basic idea of the algorithm is to exploit the condition independence properties of the SLAM model in order to break up the problem into many, smaller in size, localization and mapping problems.

The previously discussed algorithms use distance sensors and odometry for their explorations. However, due to algorithm research and SLAM 's problem popularity, new techniques in combination with visual sensors have emerged. Such algorithms are often referred to as *Visual-SLAM.* The ***EKFM-SLAM*** algorithm is among them. Despite the fact that the *EKFM-SLAM* algorithm is totally based on *EKF-SLAM*, it

uses a camera as a single sensor instead of a distance sensor. The *Random Sample Consensus (RANSAC)*[9] method is used to integrate the camera's information; that is to estimate the camera's motion (Visual Odometry).

9. The RANSAC algorithm is a learning technique to estimate parameters of a model by random sampling of observed data

Chapter 4

Our Approach

The aforementioned, and many other, approaches have all proven their abilities and robustness in SLAM research. We can point out here that SLAM algorithms can be rounded up into three classes: by the sensors they use, by which method they use or by structure. Algorithms classified by sensors may use range measurement devices such as laser sensors or ultrasonic sensors, artificial vision and odometry. Algorithms distinguished by calculation method may make use of the *Kalman Filter* or the *Particle Filter.* Finally, algorithms categorized by structure are either *Online-SLAM,* which stores only the necessary landmarks, or *fullSLAM,* which stores each landmark through exploration.

Choosing which algorithm to use is an essential step. The decision is based on the robot's abilities, by means of sensors. Some sensors, like LIDARS, are accurate to bits in their measurements. They also have a larger sensing range while they are able to cover a considerably bigger part of the environment with one scan needed. However, such sensors are expensive; both in pricing cost and battery consumption. On the other hand, sensors such as ultrasonics are low cost, able to perform accurate measurements regardless their measurement range limitation.

Our goal was to design a low cost, automated vehicle able to successfully perform *simultaneous localization and mapping*. Understanding the needs of our project, we decided to use low-cost, but reliable sensors and parts for the robot's construction, while focusing on implementing a capable application responsible for processing the robot's information. In particular, our agent focuses on navigating autonomously in the environment while avoiding any obstacles. Moreover, it collects information regarding its orientation, the distance it traveled since its previous measuring step and distances from objects around it, if there are any. Lastly, the robot sends the collected information wirelessly into a computer running application for further processing. In this way, the robot doesn't bear the

responsibility for the computationally expensive processes of mapping. It is only crucial to remember the last measurements.

On the other side of our systems, we developed an application which computes SLAM. The algorithm we make use of is the online *Occupancy Grid Mapping* technique based on an inverse sensor model. The application works as follows. At first, it makes sure that there is a wireless connection with our robot. Then, it creates a grid map with absolute uncertainty of its structure. When done so, the application is waiting for any message containing the robot's data. If there is any, it starts processing it by distinguishing the received information. After the robot's observations about its environment have been pinpointed, it creates a partial map, representing its belief of the space's structure at that time point. It is then that the update step takes place, revising the whole map and presenting us with the robot's progress. It is important to point out that the robot sends messages, containing information about its state, each time after a measurement occurs. This process continues for as long as it takes the robot to cover the whole area.

In the following two chapters, we will describe major details concerning to hardware's architectural design, the robot's *Finite State Machine (FSM)* and the software's implementation.

Chapter 5

Design of the System - Hardware

In this section, we describe our robot's hardware architecture in detail. A variety of figures accompany the report, assisting readers to a better understanding. Firstly, we describe the various electronic and electromechanic components that were used, and afterwards the robot's assembly.

5.1 Components Used

A variety of electronic components and motors were necessary for building our robotic platform. The design is based on a differential driven robotic platform, collecting its observations from ultrasonic sensors. The belief 0f the robot's heading comes from an Inertial Measuring Unit. In addition, stepper motors are used to navigate into the environment 's space. Below in this section we describe each of the parts we use, along with their utilities in our robotic platform.

5.1.1 Arduino Uno Board

Arduino Uno R3 constitutes the "brains" of our robot. It is an open-source board which includes a microcontroller, and this microcontroller is responsible for executing the instructions of our program. The board is based on *ATmega328* micro-controller. This MCU comes from the AVR family; it is an 8-bit device, which means the data-bus architecture and internal registers are designed for 8 parallel data signals. The *ATmega328* has three types of memory: a 32 KB nonvolatile *Flash memory* for storing an application, a 2 KB volatile *SRAM memory* for storing variables used by the application while it's running and a 1 KB *EEPROM memory.*

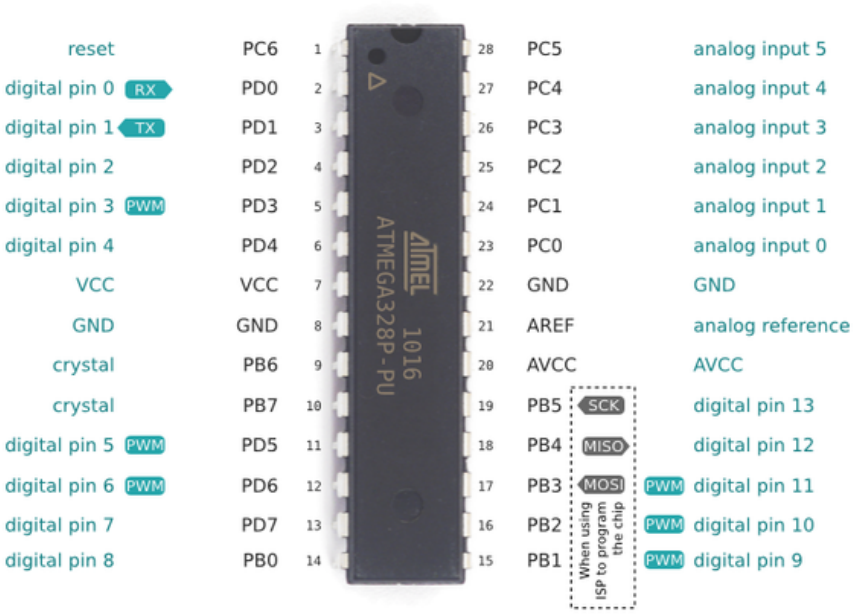

Figure 5.1 ATmega328 Pin mapping

The board is equipped with sets of I/O pins to interface with electronic components and circuits. In particular, the board has 14 digital I/O pins, of which 6 can be used as *PWM*[10] outputs, and 6 analog inputs. Each of the aforementioned pins belongs to one of the three ports; *PORTC*, *PORTHB* and *PORTD*. Each port is controlled by three registers allowing lower-lever and faster manipulation of the I/O pin of the microcontroller. The board also contains a 16 MHz quartz crystal, a type B USB connection, a power jack, *ISCP* header and a reset button.

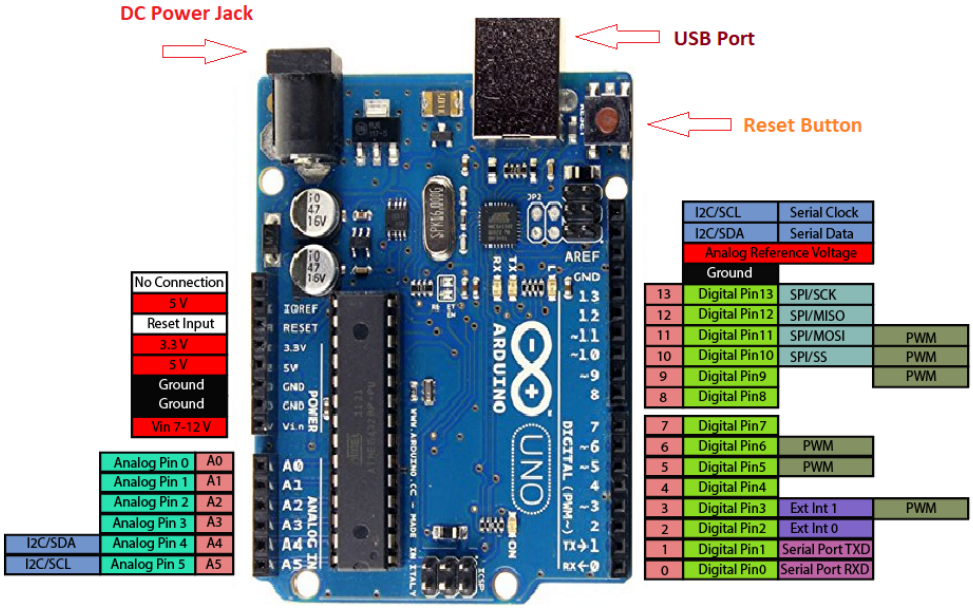

Figure 5.2 Arduino Uno Pinout (source: www.TheEngineeringProjects.com)

10. **PWM** stands for Pulse-Width Modulation. It is a way of describing a digital (binary/discrete) signal that was created through a modulation technique, which involves encoding a message into a pulsing signal.

For power source, Arduino offers the options of using the USB connection or a DC jack. Note that its operating voltage is 5 Volts, while the recommended input voltage is 7-12 Volts. Moreover, the DC current per I/O pin is 20 mA.

5.1.2 MPU 6050

An *Inertial Measuring Unit (IMU)* module is used for navigation. The MPU-6050 is a 6 Degree-of-Freedom sensor, able to measure the absolute orientation, angular velocity, angular motion and acceleration. Absolute orientation, i.e. the current heading of the robot relative to the magnetic north, is a significant value for our application, as it is the key value of calculating the robot's direction and coordinates on the environment (*odometry*).

The MPU-6050 is a sensor based on *MEMS (micro electro mechanical systems)* technology. It is equipped with a 3-axis gyroscope and a 3-axis digital accelerometer on a single silicon die, in tandem with an onboard *Motion Processor™ (DMP™)* capable of processing complex 6-axis *MotionFusion* algorithms. Any device can communicate with the sensor through an auxiliary master *I^2C (Inter-integrated Circuit)* allowing it to gather a full set of the sensor's data, as is shown in Figure 5.4.

Figure 5.3 MPU-6050. Inertial Measuring Unit module

Figure 5.4 MPU-6050 Communication Level

5.1.3 Ultrasonic Sensor HC-SR04

The HC-SR04 sonar is an ultrasonic ranging sensor. This device provides a range of 2 cm – 400 cm with an accuracy of 3 mm, while an effectual angle of less than 15 degrees is required for the sonar to give proper readings. Every HC-SR04 module includes an ultrasonic transmitter, a receiver and a control circuit. Ultrasonic sensors serve the purpose of detecting objects relative to our robot that pose an immediate threat for collision. At the same time, we use them to measure the distance between the robot and these objects.

The distance of the object or a wall from the robot is measured by noting the time between the emission and reception of the sound waves to the sonar. By multiplying time taken to travel by the sound wave, by the speed of sound we get the total distance traveled by the wave. Due to the fact that the signal takes an equal amount of time to send and receive, this value must be halved.

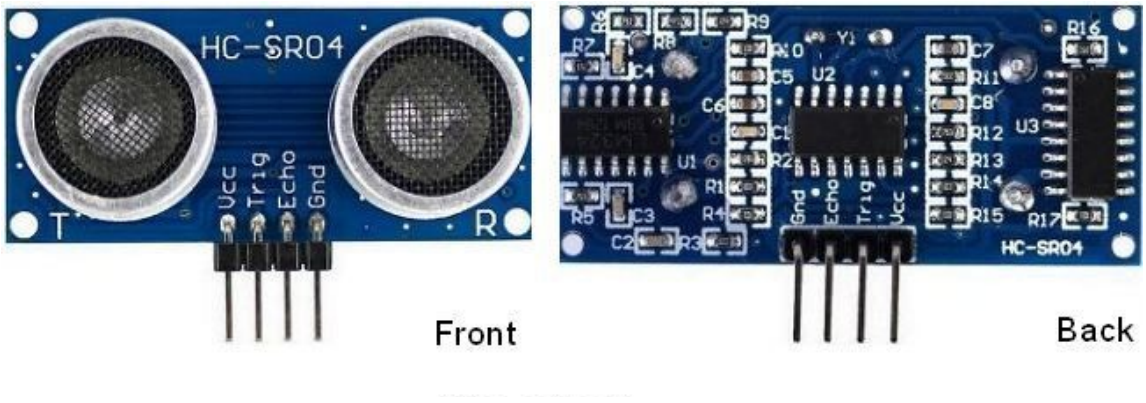

Figure 5.5 HC-SR04 Ultrasonic range sensor

5.1.4 Futaba S3003 Servo Motor

The light weight of Ultrasonic sensors makes them useful for collecting environment information from mobile robots. Many robotic platforms use ultrasonic sensors in a circular formation in surface-moving robots. However, such an approach is not suitable in our case mainly for two reasons. Firstly, due to the computationally expensive method of acquiring multiple measurements without causing errors, and secondly because of the limited I/O pins of the Arduino Uno board. Therefore we created a movable ultrasonic range sensor by combining a small, light weight servomotor and a single ultrasonic range sensor. This sensor is capable of performing 180° measurements of the distance between objects and the robot.

A servo motor is a rotary actuator or motor that allows for a precise control in terms of angular position, accelerations and velocity, capabilities that a regular motor does not have. The servomotor is a closed-loop servomechanism that uses position feedback in order to control its rotational speed and position. The control signal is the input which represents the final position command for the shaft.

The servomotor we use in our design is the *Futaba S3003 Servo*. It's a small, light weight motor, operating with 4.8-6 Volts.

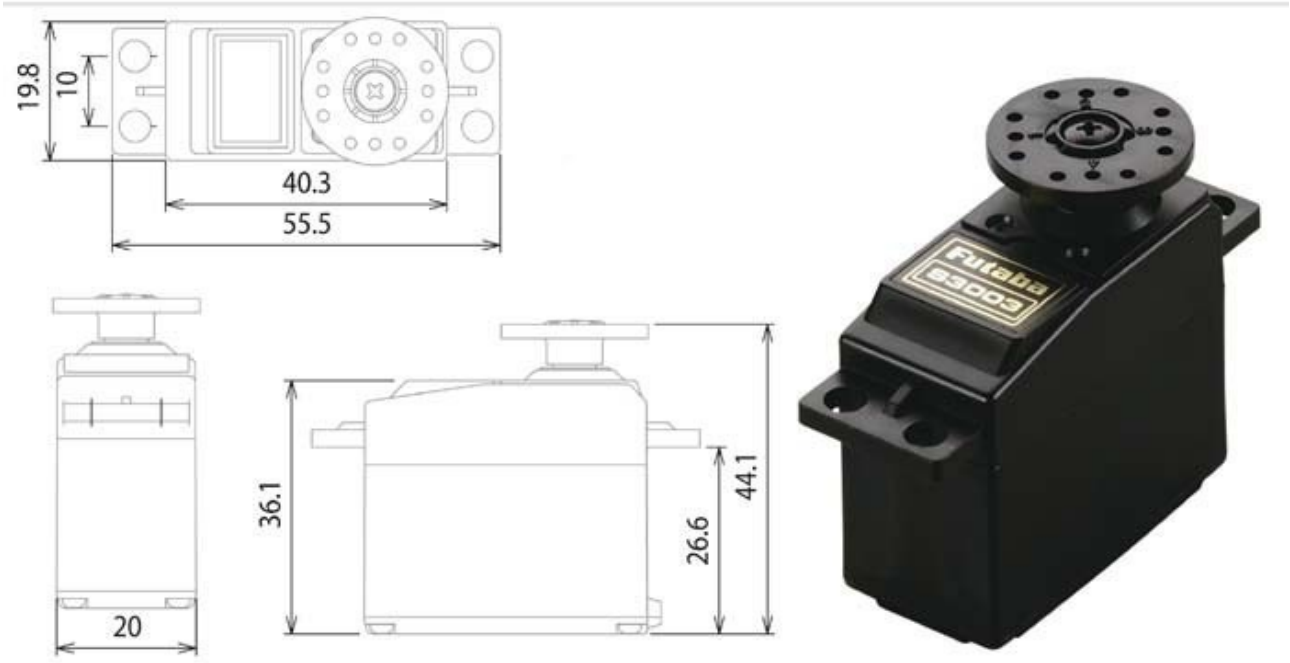

Figure 5.6 Futaba S3003 Servo Motor and its dimensions

5.1.5 28BYJ-48 Stepper Motor and ULN2003 Motor Driver

For our approach we chose to design and implement a robotic platform based on a differential driven chassis. Such robots rely on their two main wheels, each of which is attached to its own motor. Note that a third wheel is placed in the rear to passively roll along while preventing the robot from falling over.

Among the available motors, we chose to use *Stepper motors,* as they are superior to the DC motors for the purpose of control. Stepper motors are electromechanical devices which convert electrical pulses into discrete mechanical movements. One of the most significant advantages of a stepper motor is its ability to be accurately controlled without any position sensor for feedback, as long as the motor is carefully sized to the application in respect to torque and speed. They have multiple coils that are organized in groups called “phases”. By energizing each phase in sequence, the motor will rotate, one step at a time. The stepper motors we use are two 28BYJ-48 steppers.

The 28BYJ-28 is a 5 Volts unipolar geared stepper motor, originally designed for *HVAC*[11] control industry and similar low-demand equipment. It features a 5 mm “Double-D” shaft mounting lags, and a 5-conductor 0.1" pitch connector. The gearing isn’t the beefiest or most precise we ’ve seen, however it is adequate for reasonable loads. The stepper motor’s gearbox is the 64:1 reduction version. This practically means that the motor will have to make 64 steps to complete one full rotation and every step will cover a 5.6250º.

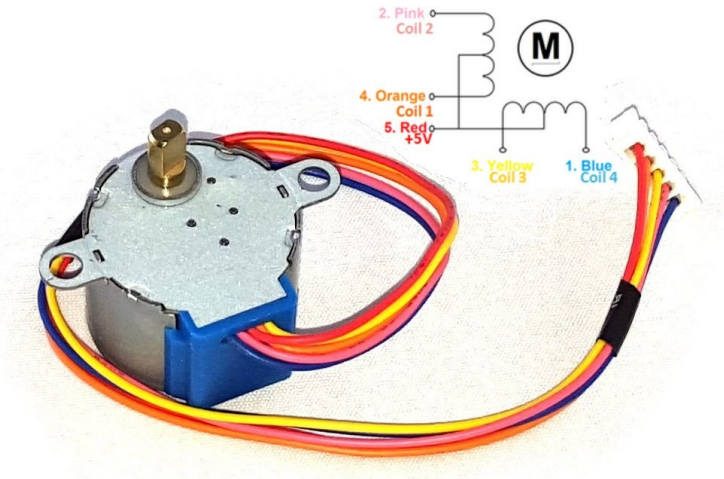

Figure 5.7 28BYJ-48 Stepper Motor

11. **HVAC** stands for Heating,Ventilation and Air Conditioning

The simplest way of interfacing a unipolar stepper motor with an Arduino, is to use a breakout for *ULN2003* transistor array chip. The ULN2003 contains seven Darlington transistor drivers and is somewhat like having seven TIP120 transistors all in one package. The board can pass up to 500 mA per channel and has an internal voltage drop of about 1 Volt when on. It also contains internal clamp diodes to dissipate voltage spikes when driving inductive loads.

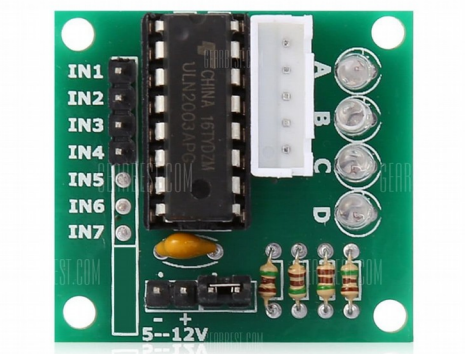

Figure 5.8 ULN2003a motor driver

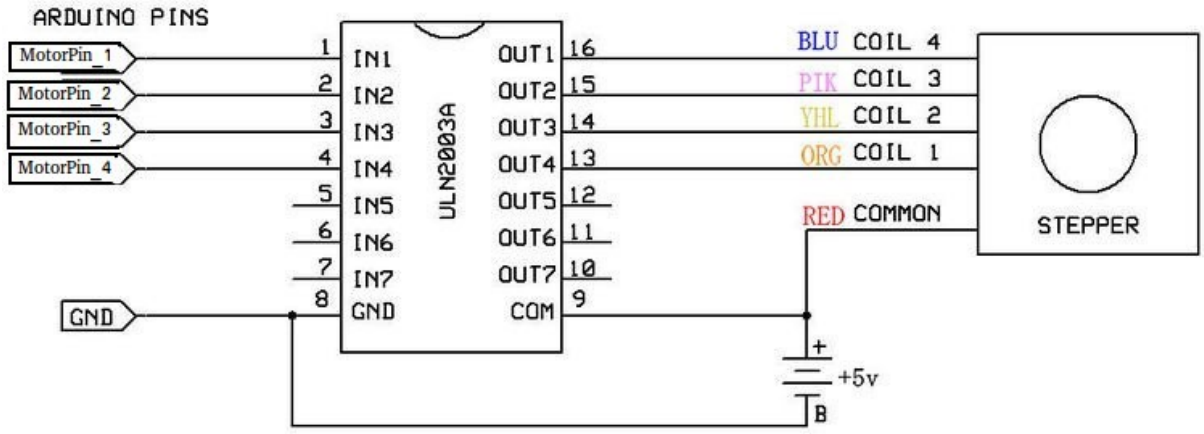

Figure 5.9 Stepper motor and motor driver wiring

In Figure 5.9 we present the electric circuit diagram with the connections between the stepper motor and the ULN2003 motor driver board.

5.1.6 HC-06 Bluetooth Module

The computationally expensive process of mapping takes place in a computer, wirelessly connected with our robot. Each time that the robot gathers a set of observations, it is required to transmit the observed information. The *HC-06*

Bluetooth Module is responsible for this communication between the robot and the computer mapping application.

The HC-06 Bluetooth module is a popular device and very simple to set up with an Arduino board. The module is suitable where wireless data transmission is needed in slave mode. The HC-06 module can reach a range up to 9 meters.

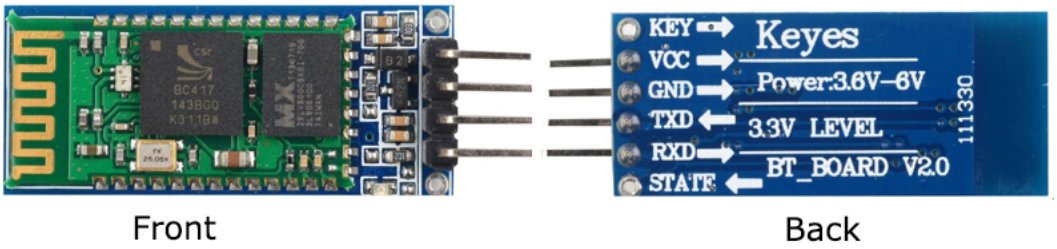

Figure 5.10 HC-06 Bluetooth Module

All the above components and sensors have been properly positioned into the robotic platform to serve the intended purpose in the best possible way. Wiring has been laid out from scratch. All safety measures have been taken in order to avoid any possible damage to the robot. In Figure 5.11 we present an overview of the designed system.

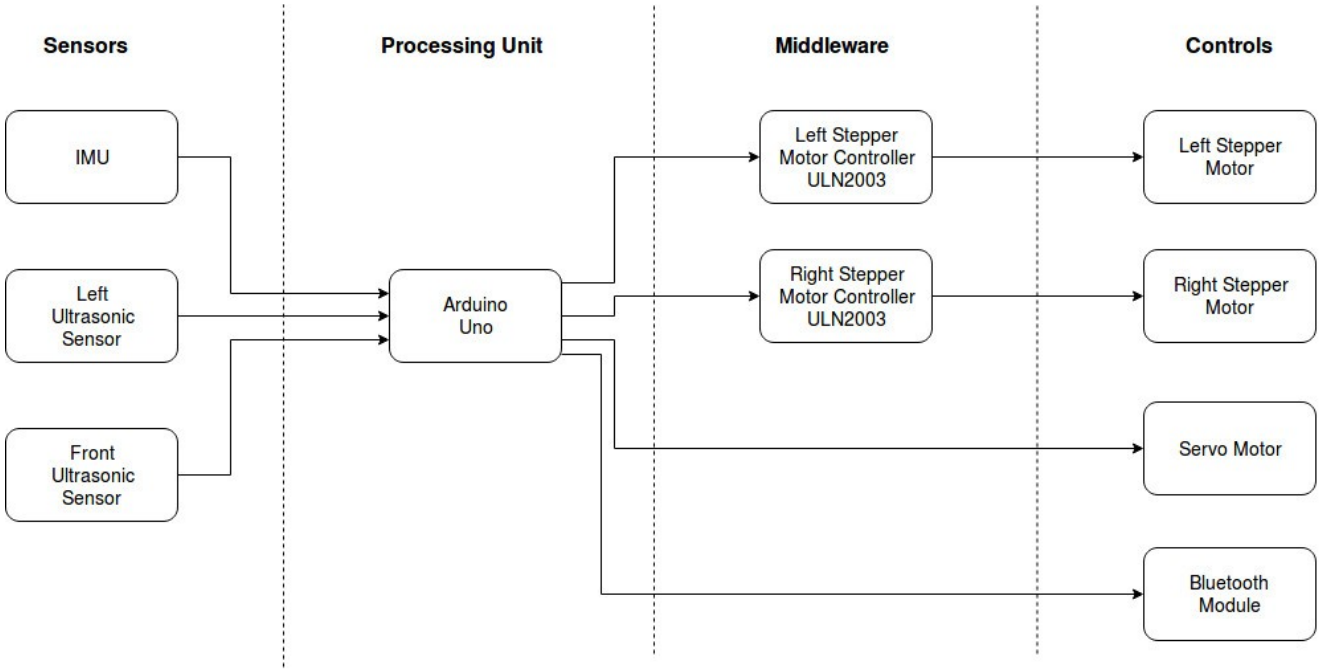

Figure 5.11 System Overview

5.2 Placement and Mounting

5.2.1 Sensor Layout

The placement of the sensors on the robot was such that all directions which the robot moves in are covered, so that obstacles and walls can detected and avoided before any collision occurs. The layout can be seen in Figure 5.12 below. We use three sensors in total; two ultrasonic sensors, of which one is movable, and an IMU. The movable range finder (which is the combination of a servo motor and an ultrasonic sensor) is placed at the front of the robot allowing it to make scans of the area in front of it. The sensor is facing the front while the servo shaft is facing the edge of 90 degrees comparing its initial point. Note that at the initial point, which is 0 degree, the sensor is facing to the right of the robot, while at the ending point, 180 degrees, the sensor is facing to the left.

The second ultrasonic sensor is positioned toward the back of the robot facing the left, which in conjunction with the corresponding sensor attached on the servo motor, allows the robot to align the wall on its left. Lastly, the MPU was placed on a platform near the middle of the robot.

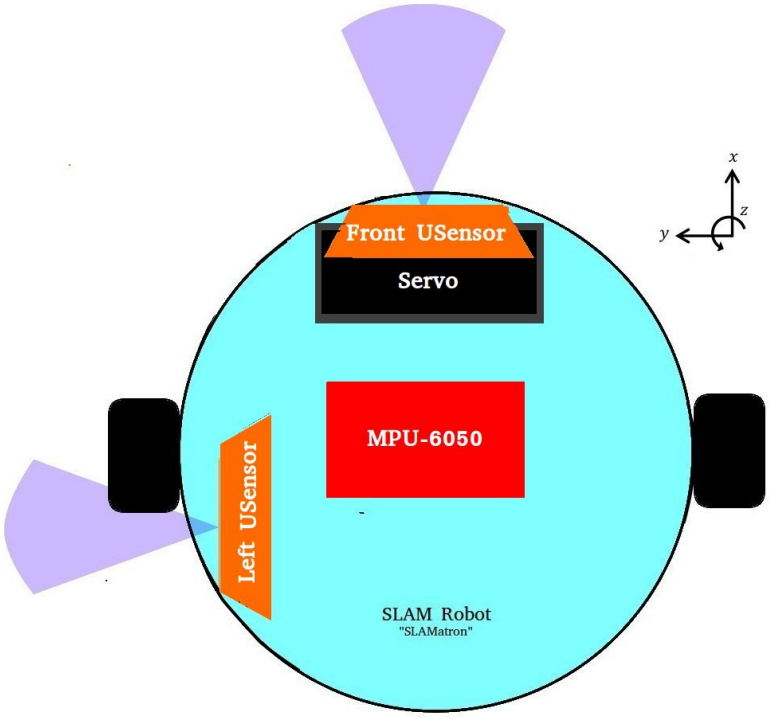

Figure 5.12 Sensor Layout

5.2.2 Mounts

Mounts were designed both for the stepper motors and the ultrasonic sensors to secure them in place on the robot, as shown in Figure 5.13. The mounts were designed in a 3D modeling platform, named SketchUp, based on each component's dimensions referred on its data sheet. The resulted models were 3D printed and properly placed on the robot.

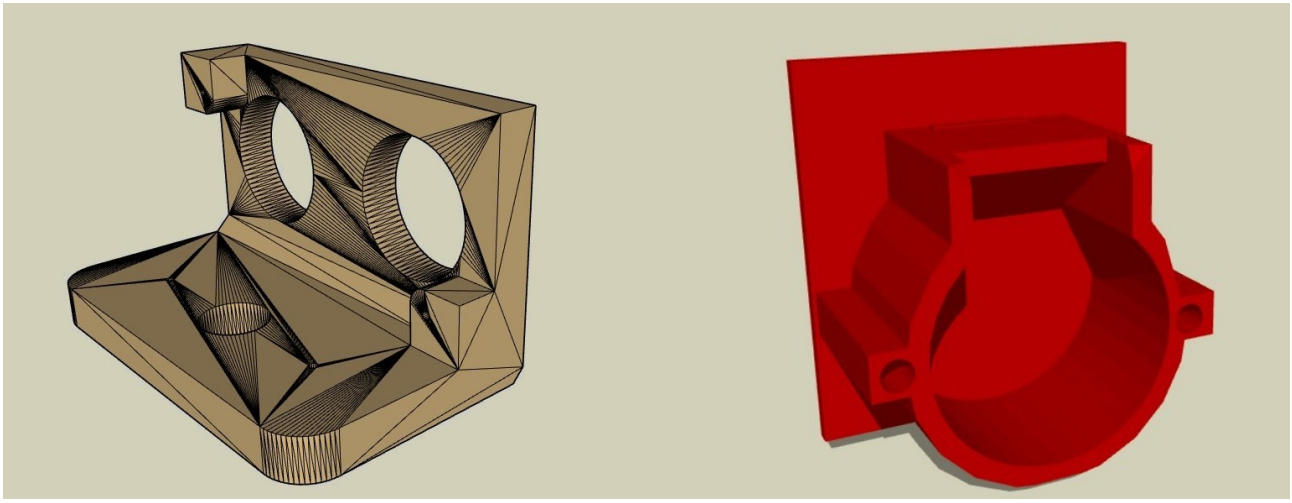

Figure 5.13 Ultrasonic Sensor's Mount (Left), Stepper motor's Mount (Right)

5.2.3 Overall Assembly

The robot's chassis consists of two levels. The sensors along with the Arduino board and the Bluetooth module are placed at the top level, leaving enough space at the bottom level for placing the two motor driver boards and the power sources. It is important to say that our system has three independent power sources. The first one, a 9V rechargeable battery, is the powering supply of the Arduino board. All the sensors and the Bluetooth module are powered by the Arduino's 5V output power supply pin. The second power source, a 9V alkaline battery, is responsible for powering the Servo motor. Lastly, the third power source is a 12V rechargeable battery responsible for powering the two stepper motors attached on the robot's wheels. For better understanding, we have designed a 3D model of our robot without the wiring between components, as shown in Figure 5.14 below. We also provide images showing the robot's figure.

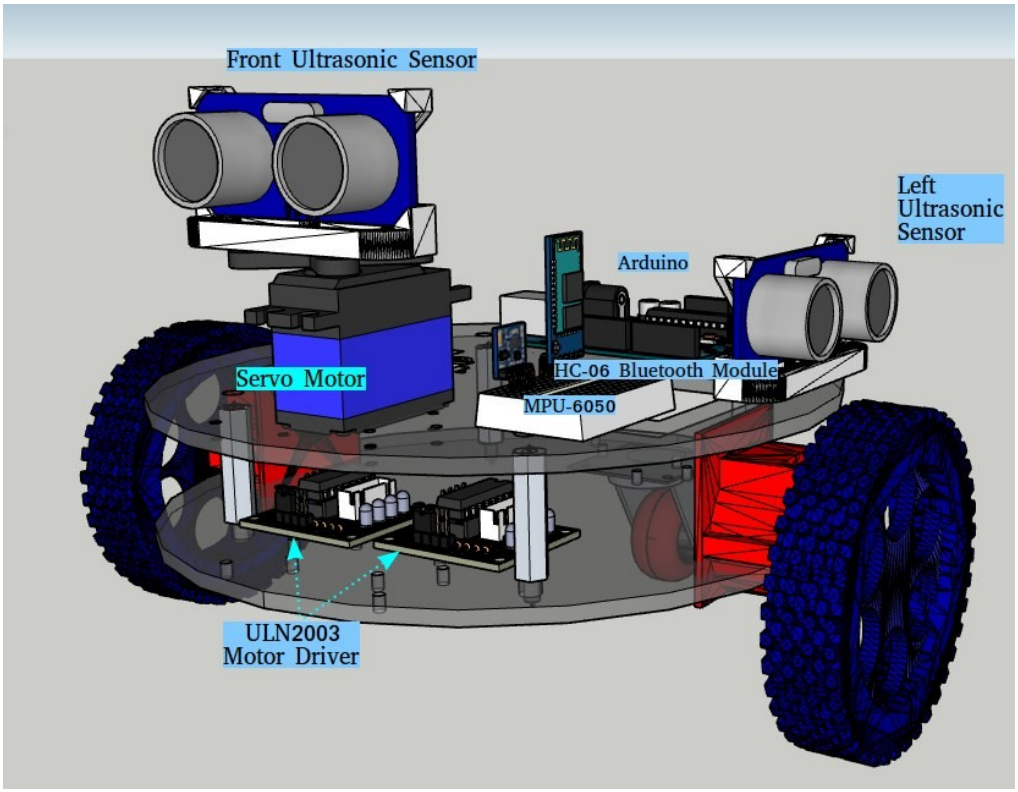

Figure 5.14 Robot's 3D model

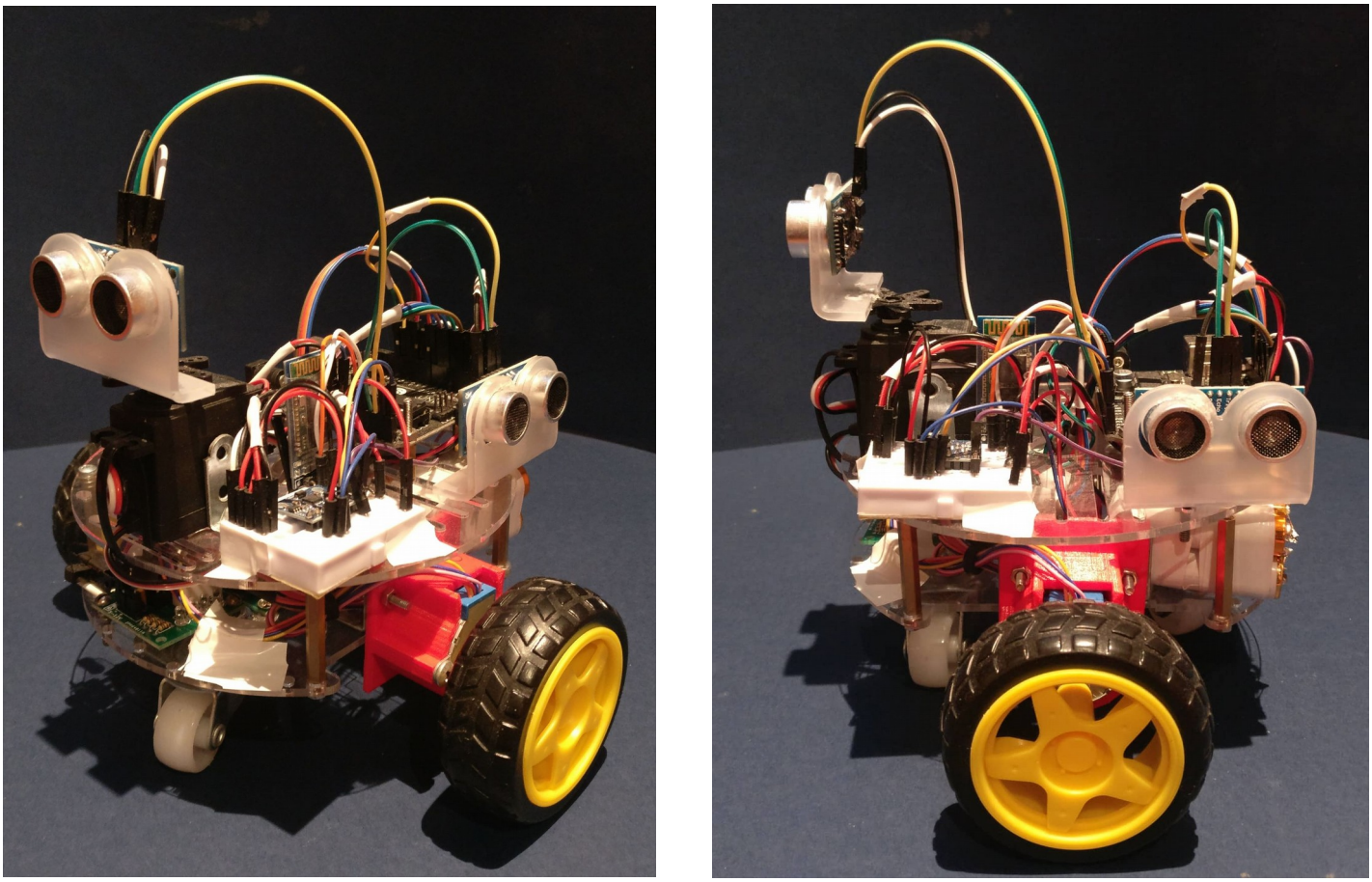

Figure 5.15 SLAM Robot

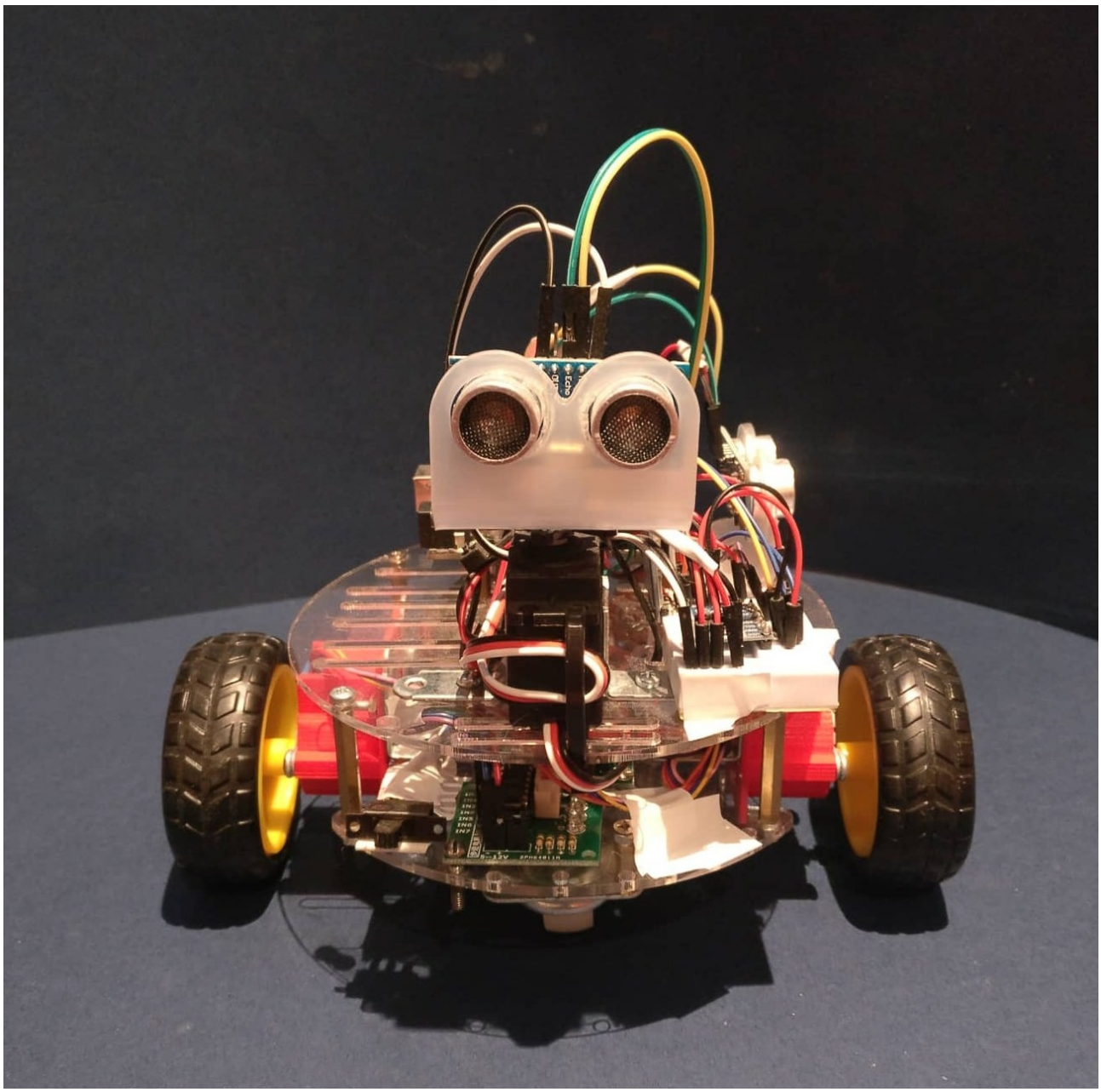

Figure 5.16 SLAM Robot (aka SLAMatron)

Chapter 6

Design of the System - Software

Equally essential with the hardware design and the robot's architecture is the software development for the processing of the sensor's data, the control of the robot and the creation of the desired map. The software development procedure can be classified in two notable classes, the embedded code development and the external code development. The first, deals with development of a program and functions on-board the Arduino Uno. The external code corresponds to the development of a mapping application on a remote device.

6.1 Embedded Code Development

The embedded code refers to a program which specifies our robot's actions. In particular, all the main modules for robot control and sensors' logging work under sync under the main program. Furthermore, the embedded code consists of various functions, where each deals with a unique task, such as the functions *send_measurements(), side_Sonar_Read() or driveStraightDistance().* For more information about the implemented functions you may check the Appendix A.

The main program is nothing more than an implementation of the robot's behavior determined by a *finite state machine (FSM)* which consists of six states. These states call of a number of functions and change various parameters in order to determine the next state. It is worth mentioning that state transitions are determined by inputs and the current state, therefore the system functions as a Mealy machine. The overall finite state machine is shown in the figure 6.1, with the entry and exit conditions shown for each state.

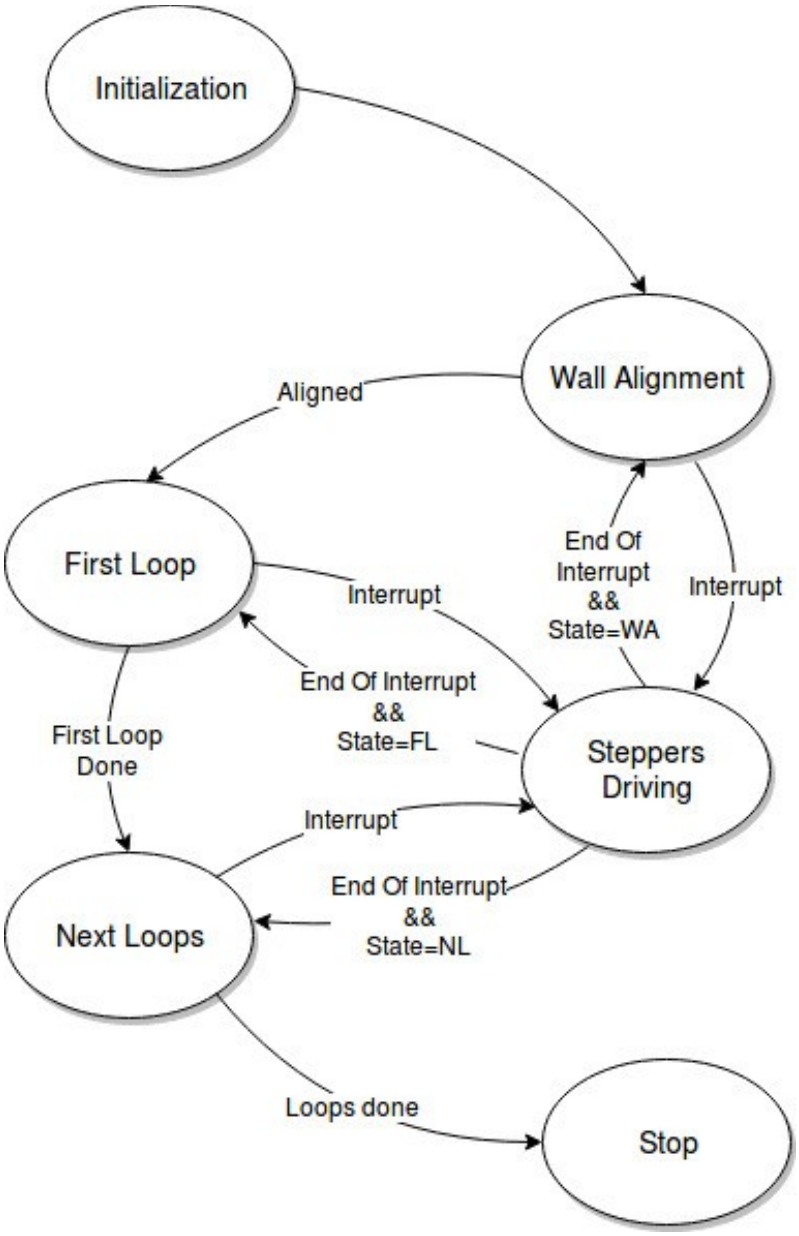

Figure 6.1 Overall Finite State Machine

The starting point is the *Initialization* state, which will remain active until the robot configures and enables the motors, configures and stabilizes the MPU readings, initialize the internal timers and obtains an initial readout of the sensors. When that happens, the current state is transitioned to *Wall Alignment* state, which remains active until the robot is aligned with the wall on its left, assumingly that it is positioned near a wall. *When the robot finally aligns with the wall on its left, the active state becomes the First Loop*. When the robot's first run is completed, namely the exploration of the environment's terrain, the current state is transitioned to *Next Loops* state. The program will progress to the *Stop* state when all loops have been completed.

Let us take a step back and examine the *Steppers Driving* state. In our approach, we designed our stepper motors to be interrupt driven. This benefits us greatly, as we are able to perform any move with our vehicle while executing any other task at the same time. Therefore, the *Stepper Driving* state is accessible by all other states, apart from *Initialization* state, when an interrupt occurs.

The major states of the designed FSM are the *First Loop state* and *Next Loops state*. For a better understanding of these particular states, we introduce the concept of path planning. *Path Planning* is an important prerequisite for autonomous mobile robots as it helps robots to find the optimal path between two points. The chosen path for the robot was to follow the wall on its left, which helps us to gauge the room's terrain. In addition with our robot's capabilities on sensors, we are able to observe the environment around the robot, providing all useful information for mapping. That takes place while the *First Loop* state is active. However, due to the ultrasonic sensor's range limitations, there might be areas of the environment which will remain unexplored in the *First Loop as shown in Figure 6.2. This occasion happens when the dimensions of the track are larger than 8 m. In that case, we need a number of additional loops of our robot exploring the unknown area. The equation that was to be used was:*

$$Number\ of\ Loops\ Needed = \lfloor larger\ side\ of\ track / 2 \rfloor \tag{6.1}$$

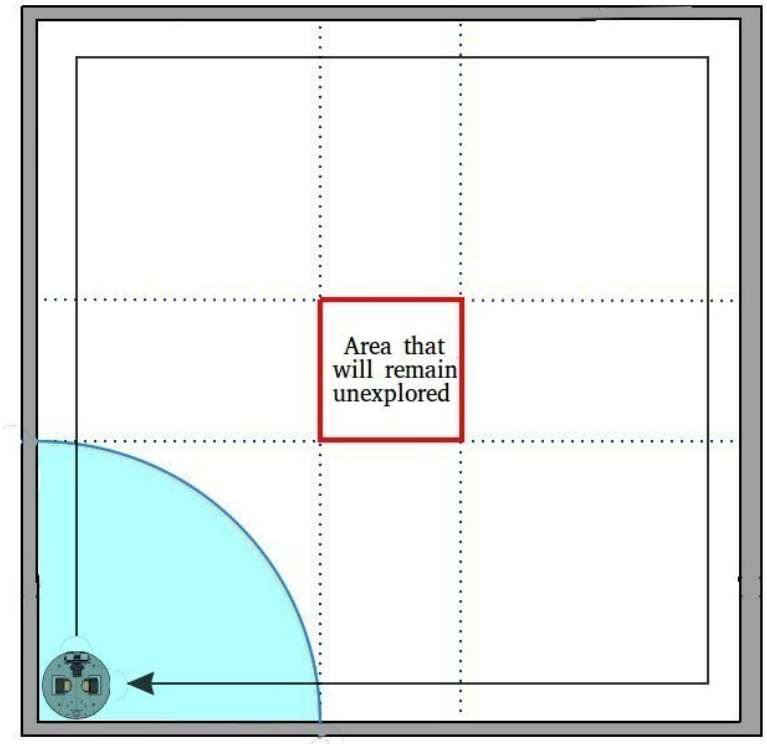

Figure 6.2 Area that will remain unexplored after the First Loop state's scan

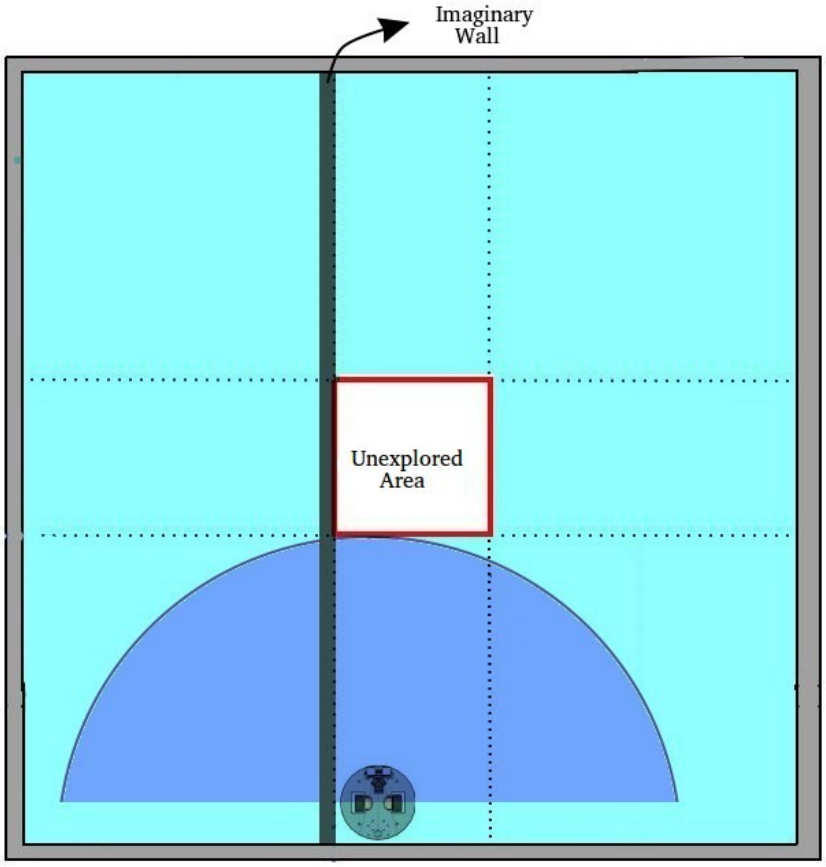

Figure 6.3 Imaginary wall that robot follows in order to cover the unexplored area

After calculating the number of loops needed to completely cover the track, the robot calculates the coordinates of the new imaginary walls to follow as shown in Figure 6.3. The current state of the FSM is then transitioned to *Next Loops* state. At that time, the robot starts moving heading to the calculated coordinates of the first imaginary wall and continues by following it and observing the unexplored area of the environment. It is crucial to mention that the robot functions similarly in both *First Loop* and *Next Loops states*. However, their major difference is that in the *First Loop* state, the robot follows an actual wall where we acquire real information about its alignment with the wall, while on the *Next Loops* state the robot follows imaginary walls where the information about its alignment with the wall it is calculated using the previous and current robot's heading.

For better understanding of how the robot actually functions in both *First Loop state* and *Next Loops state, we created a flowchart presenting all necessary steps for our robot's navigation.*

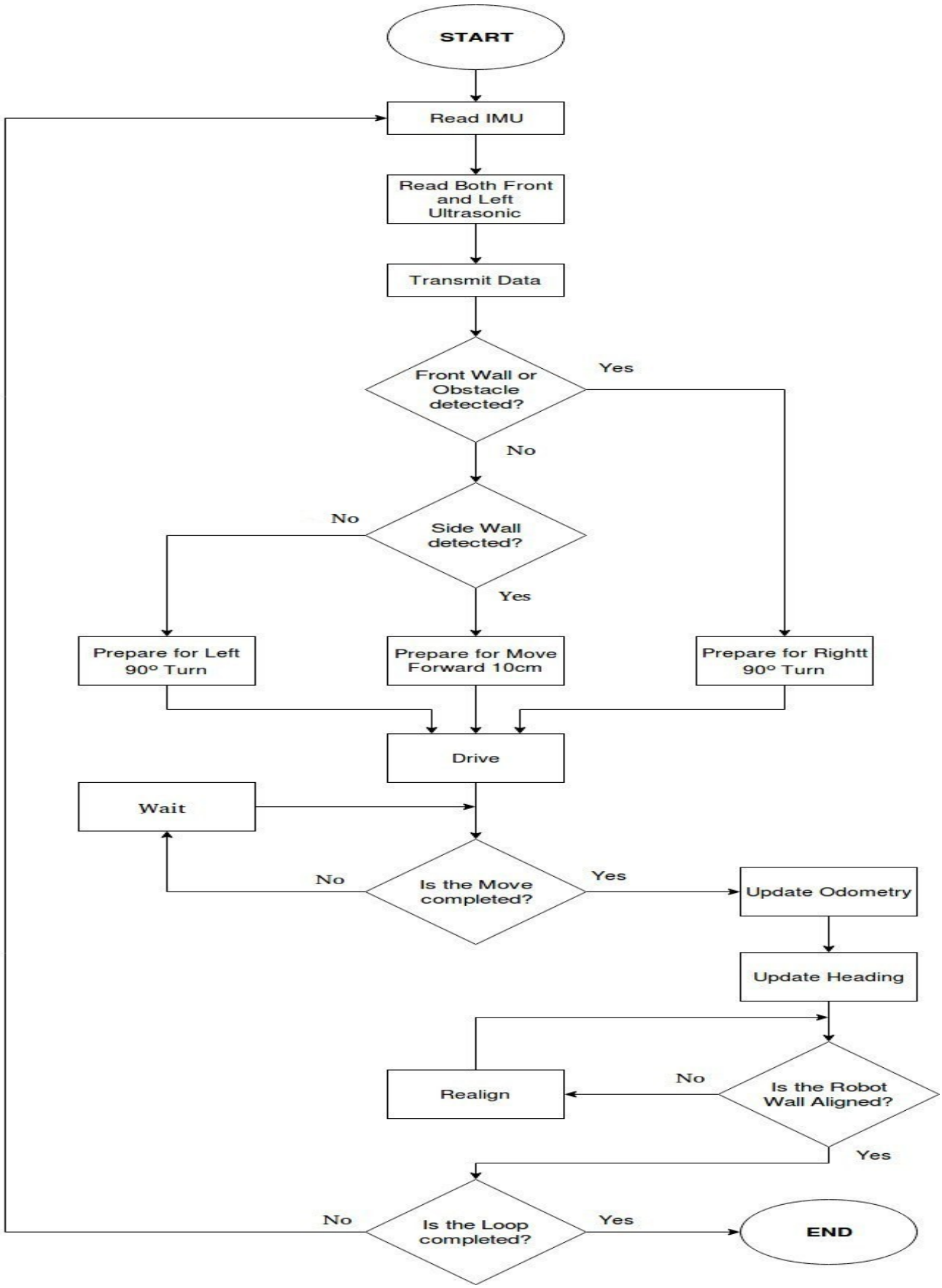

Figure 6.4 *Flowchart*

Figure 6.4 shows the flowchart for mobile robot navigation. When the state is transitioned either to *First Loop* state or *Next Loops* state, the robot calculates the value of its heading degree by fusing MPU's data. Then, it looks for obstacles or walls around it and measures their distance using ultrasonic sensors. At that point, the robot transmits all necessary information for mapping to the base station. As we discussed, in our approach the robot follows a wall to navigate through the environment. The robot has to make a decision based on the aforementioned measurements whether it will move forward or make a left or right turn. If there is no obstacle or wall at the front and a left side wall is present, then the robot moves forward. If a front wall is detected then the robot takes a right turn and when both walls are not detected the robot takes a left turn. While in rhis turning process, the robot uses MPU's value to make an exact 90 degree turn.

In pursuit of the move 's completion, the robot updates its co-ordinates and its heading. It is worth mentioning that initial co-ordinates of the robot are assumed to be [0,0]. At that moment, the robot checks if it is aligned with the wall on its left. In case that is not, the robot re-aligns with the wall and continues. Note that the state will be transitioned to *Next Loops* state and complete the first loop's scans once the robot revisits the initial co-ordinates.

6.2 External Code Development

The external code has to do with the implementation of a mapping application. In our approach, we designed our system to function using an occupancy grid mapping technique. Therefore, the mapping application implements an Occupancy Grid algorithm as it was described in the subset two of the second chapter (2.2 Occupancy Grid Mapping).

The programming language of our preference was Python. Additional libraries had to be used in order to perform all necessary tasks, two of which are of great importance for the proper running of the program. The first crucial library is the *serial library* which is responsible for the wireless communication between the robot and the remote device. The second one is the *matplotlib library* from which we make use of the *pyplot package* for the purpose of visualizing the robot's belief of the environment's structure.

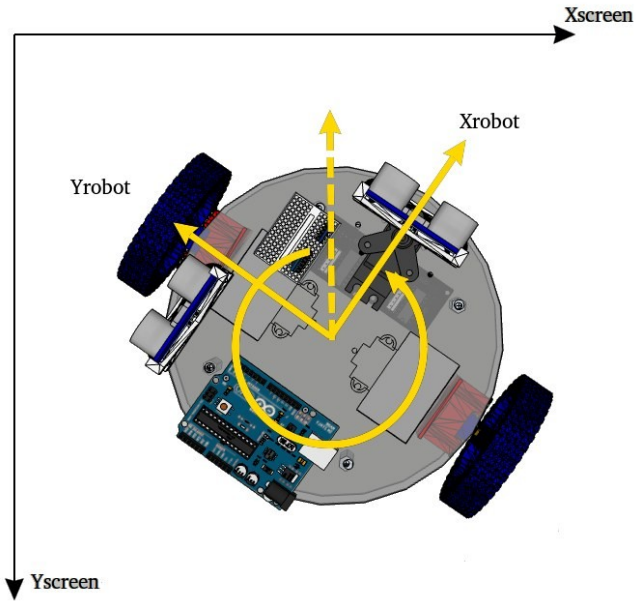

Figure 6.5 Robot and screen coordinate systems

Initially, the application sets the wireless connection between the robot and the remote device. Once the connection has been established the program continues with the initialization phase. At first it creates an array and stores the values of the 19 bearing angles[12] in the format of radians. It then creates a second array, consisting of three cells in which we store the robot's believed coordinates and heading every time the robot sends measurements for processing. The aforementioned data in addition to the observed distance measurements around the robot's position, are used for updating the instances of the map. Lastly, the grid size is specified and the grid map is constructed with the cells having total awareness about their states (free or occupied).

Note that all of the values obtained from the sensors were with respect to the robots coordinate system and thus had to be transformed. This was achieved through the recalculation of the all bearings from the robot. Figure 6.5 shows the coordinate systems of both the robot and the screen. The map on screen was scaled so that 1 mm of the robot's travel equals 1 mm.

Subsequently, the program waits for the robot's messages. Once they are received, the program decodes them and it distinguishes the observed data. Then, the map needs to be updated. In this case our application decides which cells of the

12. The 19 bearing angles corresponds to the 19 measurements that the robot makes at a single point in order to observe his surrounding area

grid map should be altered by calculating the L-norm distance to all cells from the robot. Once the application has figured out which cells are in range of the robot's sensors, it has to calculate the likehood of their being occupied by an obstacle or a wall. It is crucial to mention that the robot's belief of the map is shown through out the process of SLAM. By other words, our system consists of an online application. Note that the application terminates once its receives an analogous *ending* message from the robot.

Chapter 7

Validation of the System

Platform construction and system design for both hardware and software have been described in detail in the previous chapters. Yet, the target of this project is not only the construction of the platform, but also the proper performance of the system. Therefore, thorough testing was necessary for the design and functionality of the SLAM robot and was performed for all aspects of the system.

The first target for the platform in development was primarily for hardware performance as it was necessary to prove proper functionality and robustness. Therefore, the initial experiments were focused on testing the range sensors after calibration, testing the Bluetooth for the wireless communication with the computer, testing the IMU's data as well as testing for any inherent differences in the stepper motors. From these tests we drew some conclusions of great importance for the robot's proper behavior.

To begin with, it was discovered that the ultrasonic range sensors were only capable of reliably reading up to 30 cm rather than the 40 cm specified on the data sheet. Testing the IMU's data revealed that the sensor required calibration to ensure right performance. Moreover, testing the stepper motors revealed that they required absolute synchronization to ensure proper steering.

Later testing of the SLAM robot was aiming to validate the functionality of the embedded code. At first, portions of the code, such as a state of the FSM or a function, were tested individually to ensure they performed as desired. Once the results met the requirements and provided the proper performance, we then tested the embedded code as an integrated part of the system. It is worth mentioning that in order to debug the embedded code, outputs of the FSM's current state and the sensors' reading were sent over Bluetooth to a computer so that a written output of what the robot was doing could be seen. While in this stage of testing, problems such as the insufficient voltage being sent to the Bluetooth module and the

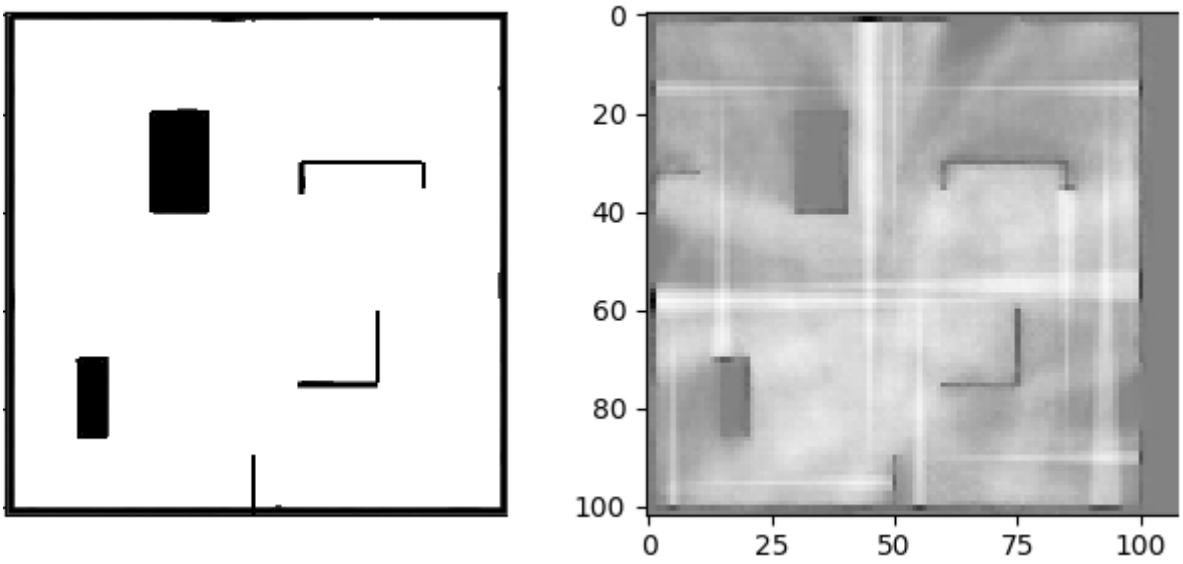

Figure 7.1 Mapping application testing.
The environment's structure is being presented on the image at the left, while at the right image the robot's belief about its environment is being shown.

servo motor were discovered. The results of these tests also helped to alter and improve the robot's design.

Finally, the testing process aimed at evaluating the proper performance of the external code. To do so, we initially tested our application with the sensor readings observed by another system's robot which was successfully tested and it performed well. This test was carried out in order to establish how our application was corresponding with the sensor readings that we know from the map they create. As Figure 7.1 shows, our application met our expectations and successfully produces the map which corresponds to the above sensor readings.

Further experiments were conducted testing our system in total. All the testing runs were successful, providing us with an accurate representation of the environment's structure. While the overall software design of the robot has room for improvement, it is evident that from the results that the robot was able to successfully navigate the room with sufficient coverage avoiding all obstacles. Rotation control and wall following were also executed very well. In figures below, we present the results of several experiments in comparison with the environment's structure.

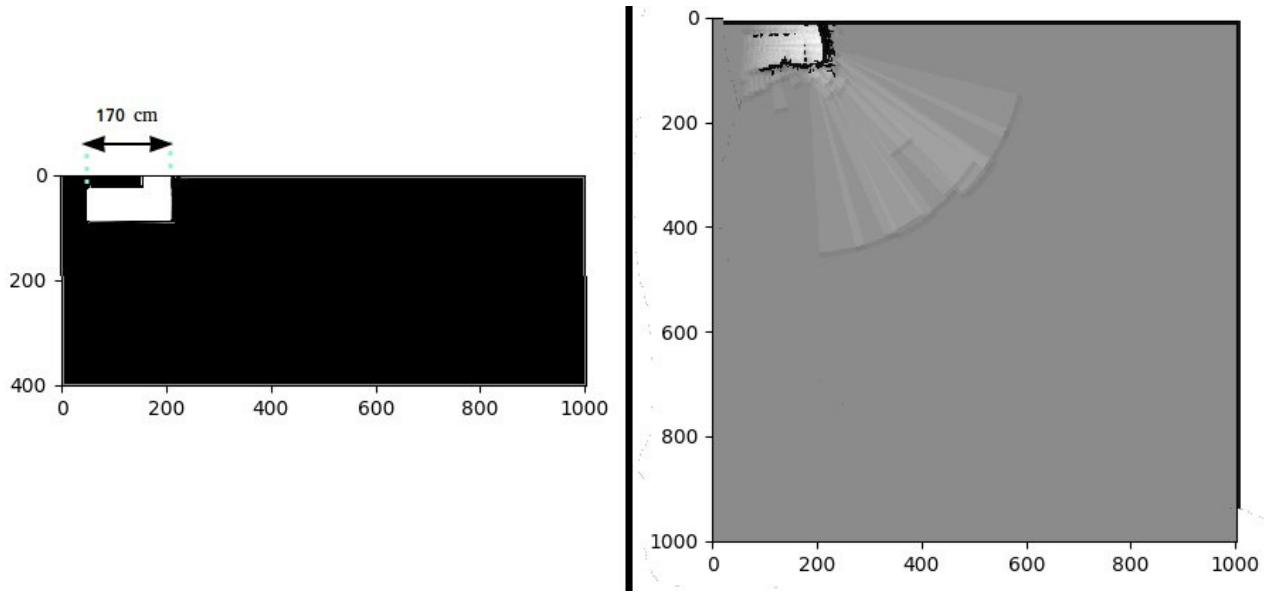

Figure 7.2 First experiment. Robot navigates in a straight path, which is 170 cm of length.
The image on the left represents the environment's path while the image on the right is picturing our system's result..

From the first experiment we conducted and presented on the Figure 7.2, we can understand how the number of scans in a particular area is connected with the robot belief for this area. By increasing the number of scans in a particular area, the robot 's belief gets stronger on whether the respective grid cells are occupied or free.

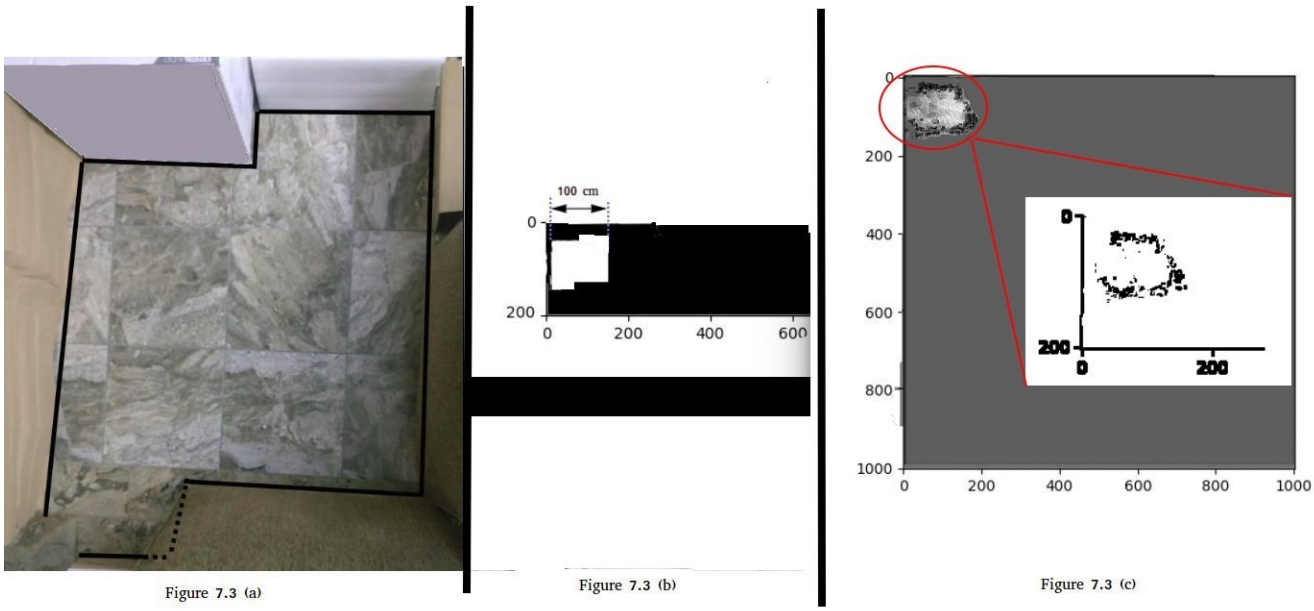

Figure 7.3 Second Experiment. Robot navigates in a constructed environment, which is shaped similar to a square. Note that there is no object in the middle of the structure.
(a) Photograph of the environment, (b) Environment's structural draw, (c) System's result

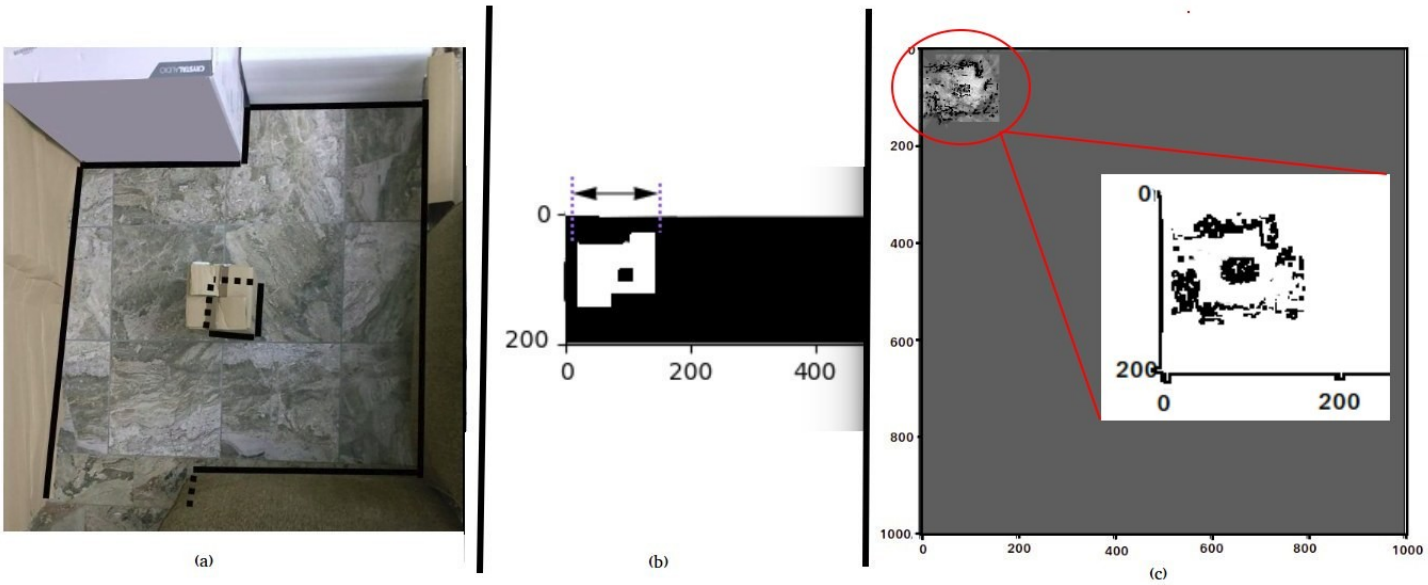

Figure 7.4 Third Experiment. Robot navigates in a constructed environment, which is shaped similar to a square. Note that there is an object in the middle of the structure.
(a) Photograph of the environment, (b) Environment's structural draw, (c) System's result

While observing the Figures 7.3 and 7.4 we come to the conclusion that our system is not fully capable of identifying all the corners on our space. Yet, this observation was expected due to the limitations of the ultrasonic sensors. However, this drawback can be avoided by further processing the resulted image, meaning that we could apply an edge detection filter (e.g. Canny edge detection). Thus, we could produce a clear map representing the environment.

Chapter 8

Conclusions and Lessons Learned

8.1 Conclusions

This thesis has described in detail the assembly, development and testing of a low-cost, autonomous mobile platform developed for the purpose of mapping while decreasing the platform’s cost. It has described the drawbacks and the advantages that our system offers. It has also presented and briefly explained the related work in the domain. Moreover, there has been an exhaustive description of every component used in the SLAM robot, as well as the software development procedure. Through thorough testing and experimentation, the system’s performance proved its ability to serve its purpose.

Whilst the performance of the robot was satisfactory in both mapping and coverage capabilities, the robot has room for further improvements in order to achieve accurate results. Better sensors, such as Sharp IR sensors along with low pass filters to reject high frequency fluctuations in readings or 180 degree LIDAR sensor, could replace the current ultrasonic sensors. Moreover, additional sensors could be used in order to achieve 360 degree detection instead of the current 180 degree detection. These improvements might be slightly expensive in both cost and energy consumption, however the system could have plotted an excellent map and kept track of the obstacles with ease.

8.2 Lessons Learned

As I am heading toward the end and reflecting on the journey thus far, I'm thinking of all the lessons I have learnt about conducting this research and writing

this thesis. There were lessons that were something I had to stumble across on my own, yet others that I got from somebody else.

The first lesson that I learnt during my thesis was the importance of wasting 2–3 hours in learning things, that would have saved days and weeks in writing thesis and more importantly would have relieved lot of mental stress that I under went just because I did not used efficient and smart methods. In particular, I ignored reading the MPU sensor's manual. This resulted in using the sensor without calibrating it. Subsequently the robot was unable to rotate properly due to the faulty measurements (could not properly locate the magnetic north). As a result, I spent quite a lot of time trying to understand why the robotic vehicle did not responded as I wanted. In the end, by having read what the manual had to tell us, I understood that this type of sensors had to be calibrated every time that they are to operate in a different location.

The second lesson I learnt, I got it from someone else. At the very beginning of the robotic platform's implementation, we used DC motors for its steering. This turned out, to be a huge mistake in the design. While trying to drive the robot forward in a straight line, we observed that either the robot 's wheels were drifting or the robot was moving diagonally. We tried to control the DC motors and synchronize them using a plethora of approaches. At first we implemented a "master-slave" logic, meaning that we assumed there was a "master" wheel and a "slave" wheel. The concept was that the slave would have to follow the master, whilst the master would not proceed ant further unless the slave had reached it. However we tried to control both motors in such a way, we did not get the expected results. Then, we thought of implementing a PID controller in oder to successfully control both motors. But even in this approach, we did not have any improvement. The answer to our problem was given by Professor Apostolos Dollas. During a meeting we had, we discussed about the problem we were facing and he suggested changing the motors we use. Therefore, we changed the DC motors into stepper motors, which were easier to be controlled, achieving proper steering of the robot.

Bibliography

[1] Dhiraj Arun Patil, Sakshi Vinod Agiwal, “Design and Implementation of Mapping Robot using Digital Magnetic Compass and Ultrasonic Sensor”, International Journal of Engineering Research & Technology (IJERT), ISSN: 2278-0181, Vol. 4 Issue 06, June-2015.

[2] Neelam Barak, Neha Gaba, Shipra Aggarwal, “Two Dimensional Mapping by using Single Ultrasonic Sensor”, ISSN: 0976-5697, Volume 7, No. 3, May-June 2016.

[3] Sebastian THRUN, Wolfram BURGARD, Dieter FOX, “PROBABILISTIC ROBOTICS”, 1999-2000.

[4] Md. Shadnan Azwad Khan, Shoumik Sharar Chowdhury, Nafis Rafat Niloy, Fatema Tuz Zohra Aurin, “Simultaneous localization and mapping (SLAM) with an autonomous robot”, BRAC University.

[5] R. G. Brown, B. R. Donald, “Mobile Robot Self-Localization without Explicit Landmarks”, Algorithmica (2000) 26: 515–559.

[6] Vassilis Varveropoulos, “Robot Localization and Map Construction Using Sonar Data”, 1999.

[7] Jørund Øvsttun Amsen, “Improving Navigation and Mapping with Arduino robot”, Norwegian University of Science and Technology, June 2017.

[8] Weihua Chen, and Tie Zhang, “An indoor mobile robot navigation technique using odometry and electronic compass”, International Journal of Advanced Robotic Systems, 2017.

[9] Woo Yeon Jeong, Kyoung Mu Lee, “Visual SLAM with Line and Corner Features”,Oct. 2006 .

[10] Josep AULINAS, Yvan PETILLOT, Joaquim SALVI, Xavier LLADÓ, “The SLAM problem: a survey”.

[11] Shoudong Huang, Gamini Dissanayake, “Convergence and Consistency Analysis for Extended Kalman Filter Based SLAM”, IEEE Transactions on Robotics, 08 October 2007.

[12] L. Armesto, J. Torreno, “SLAM based on Kalman filter for multi-rate fusion of laser and encoder measurements”, IEEE/RSJ International Conference on

Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), Oct. 2004.

[13] WooYeon Jeong, Kyoung Mu Lee, “CV-SLAM: a new ceiling vision-based SLAM technique”, IEEE/RSJ International Conference on Intelligent Robots and Systems, Aug. 2005

[14] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, “FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem”,2002.

[15] Michael Montemerlo, Sebastian Thrun, “Simultaneous Localization and Mapping with Unknown Data Association Using FastSLAM”.

[16] Austin Eliazar, Ronald Parr, “DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks”.

[17] F. Caballero, L. Merino, A. Ollero, “A general Gaussian-mixture approach for range-only mapping using multiple hypotheses”, IEEE International Conference on Robotics and Automation, May 2010.

[18] SEBASTIAN THRUN, “Learning Occupancy Grid Maps with Forward Sensor Models”, Autonomous Robots 15, 111–127, 2003.

[19] Thommas Colleens, J.J. Colleens, Conor Ryan, “Occupancy grid mapping: An empirical evaluation”, Mediterranean Conference on Control & Automation, June 2007.

.

Appendices

Appendix A

Embedded Code Development – Functions

In this section we are presenting the functions which we make use of for the robot's autonomous navigation and we are briefly explaining them. Note that these functions are stored in the embedded program.

- **void send_measurements()**: Function that sends measurements of the robot's surroundings. We send AHeading (aka Artificial Heading) and distance that robot traveled in order to compute its coordinates on the grid. We also send all measured distances of the objects around the robot, in order to process them and create the corresponding map. The format that the information is being sent, has to be as the following as it has to match while being read by the python script
- **void end():** Function helps to terminate the python script and show the final mapping when the robot stops moving and collecting new information
- **void Side_Sonar_Read():** This function is being used in order to get the distance between the robot and the wall on its left Thus, we are able to understand if there is a wall so we can follow it. The sensor responsible for this task is located on the back left side of the robot. This distance information, plus the distance information measured from the function below when the servo pose equals to 180, will also help us to align the robot to the wall next to it.
- **void Front_Sonar_Read():** This function helps us measure the distance between the robot and the objects,if there are any. It collects 19 measurements which are being collected from an ultrasonic sensor attached on top of a servo motor. The 19 measurements correspond to the range [0-180](degrees) with a step of 10 degrees. The 0 degree position is on the right side of the robot, the 90 degrees position facing at the front of the robot and the 180 degrees position is at the left side of the robot. It is important to mention that in order to get more accurate measurements, we repeat the procedure twice and store the average of the two results.

- **int CMtoSteps(float cm):** Function that converts the distance the robot shall travel from centimeters into number of steps.
- **void driveStraightDistance(int steps):** Function that drives the robot forward for a specified number of steps.
- **turnAbsolute(float target):** Function that turns the robot in a specified angle. We have an offset within 0.5 degrees of the desired angle
- **void get_heading(float *head):** This function is being called every time we need to calculate the robot's heading. It is being used in order the robot to turn left or right by 90 degrees with an error in +-0.5 degrees.
- **void CheckIMU(int *state):** Check to see if the IMU has settled down and is giving a steady heading. If it hasn't then the robot can not start navigating.

Appendix B

Electric Diagram – Circuit

In this section we presenting the connections between the various electronic components used in the SLAM robot.

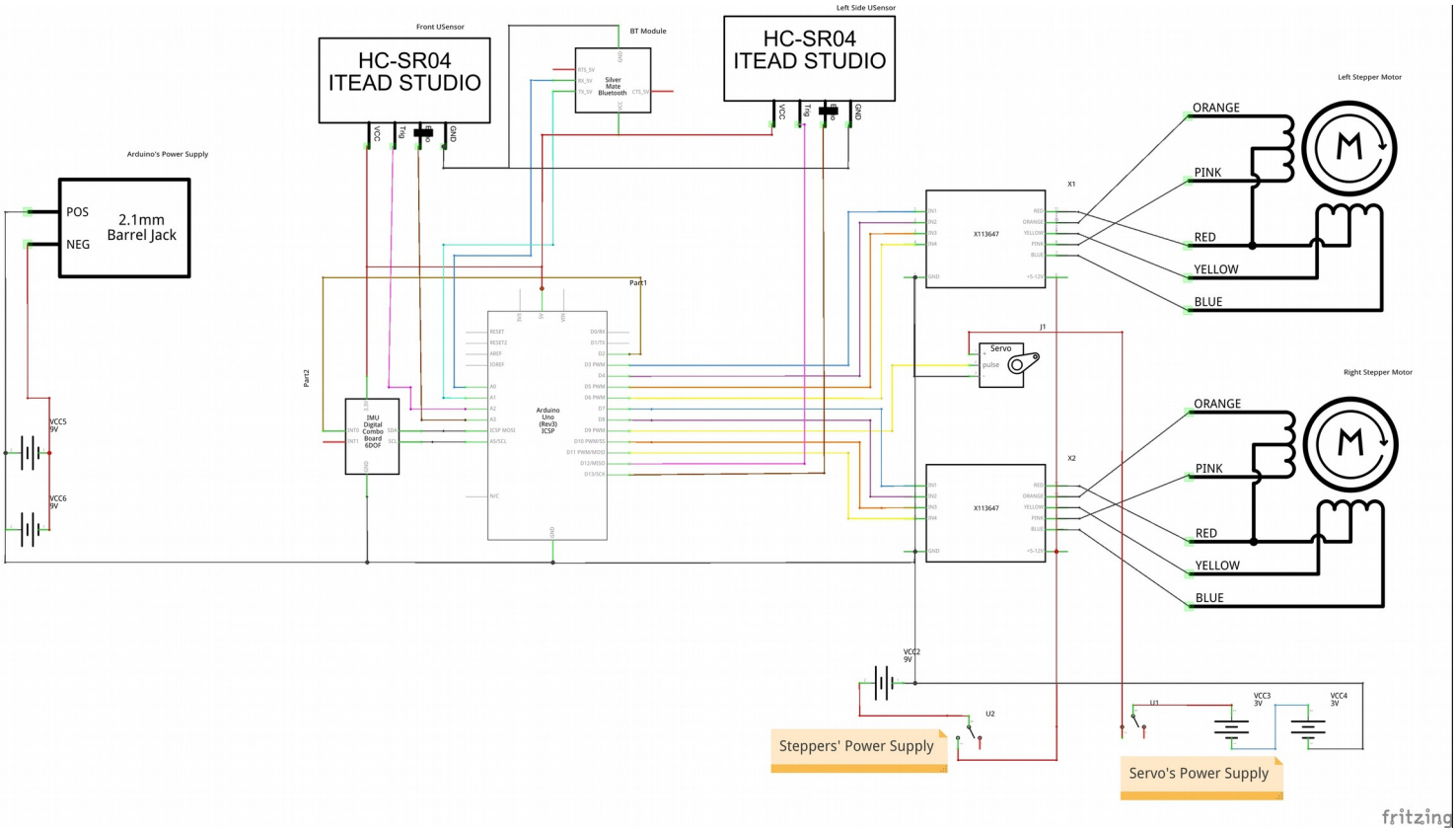

Figure B.1 Electric Diagram

Appendix C

Block Diagram

In this section we present the block diagram of the components used in the robotic platform.

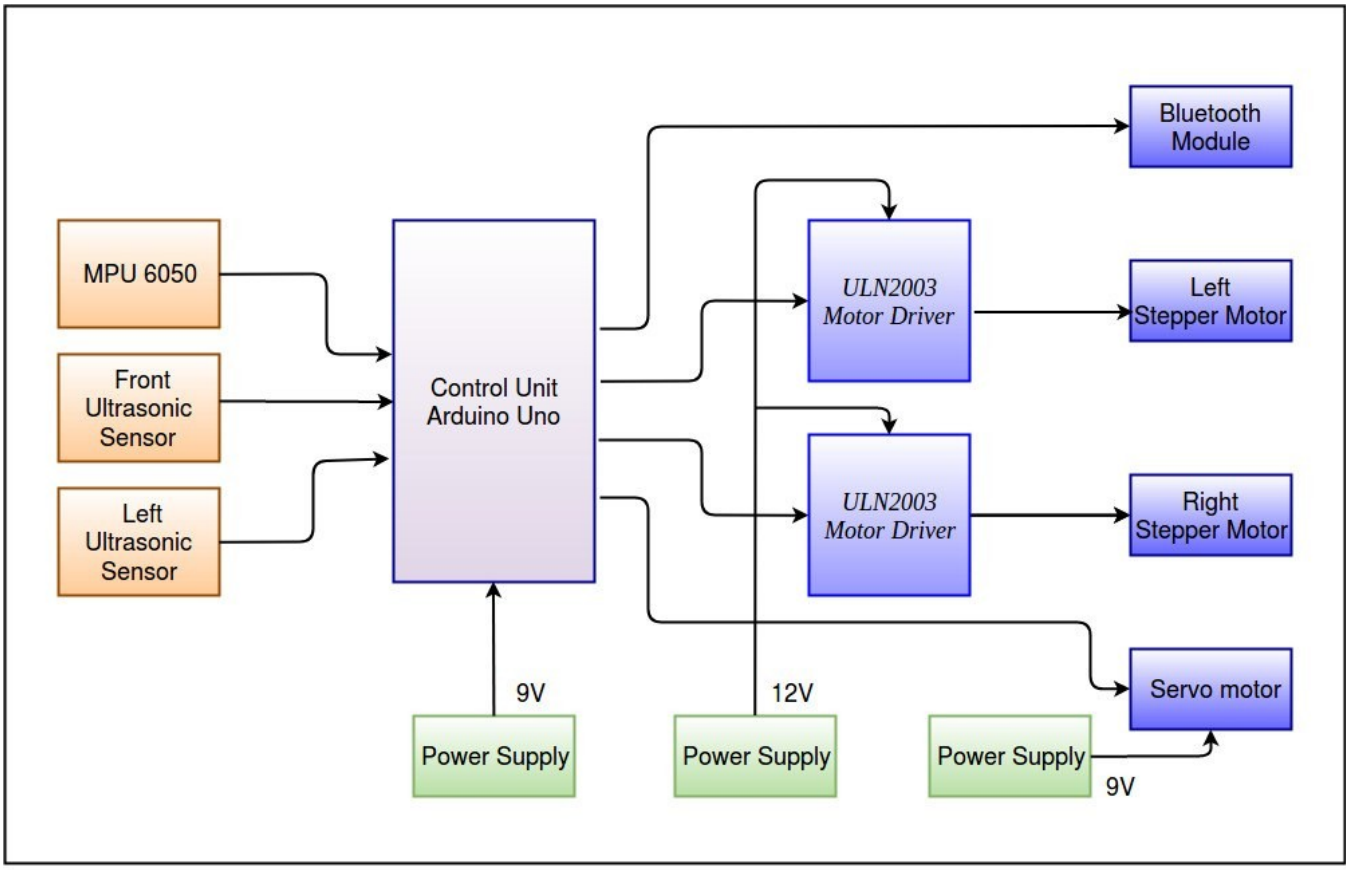

Figure C.1 Block Diagram