POLITECNICO DI TORINO

Engineering Area

Corso di Laurea Magistrale in Ingegneria Mechatronica

Path tracking developmet for a mobile robot

Thesis for Masters Degree

Supervisor

Prof. Giuseppe Quaglia

Candidate

Ajay Chaudhary

March 2018

Special thanks to my family for their support

and continuous encouragement.

A deep gratitude to my friends and to all the people

who have accompanied me in these years.

Summary

The path planning of a mobile robot in this paper is a result of a recent work based on previous use of LIDAR sensors. Lidar sensors have been in use for many years now in various fields for numerous purposes. Path planning, environment mapping, obstacle avoidance, weather mapping, terrain mapping and etc. have been the major application of a Lidar sensor.

The work in this thesis consists of the use of RPLIDAR A1 sensor from Robo Peak for the development of the algorithm for environment mapping, localization and trajectory planning with obstacle avoidance in indoor environment.

At first it is about learning the limitations and calibrating the sensor for optimal use in closed environment. This is followed by the development of algorithm for mapping the said environment. Then the second stage of development requires us to develop algorithm for trajectory planning and localization or SLAM.

The central part of the work consists of the development of efficient and productive algorithm for mapping and path planning in a simulated environment.

Finally it produces the possibility of reading of the local environment and trajectory planning in closed environment. This information can be useful in developing in-house robots for service industry that can be used in restaurants, offices and house and be a part of our daily lives while making them easier and more productive. Use of 3D sensors and cameras can vastly improve the performance and usability of the mobile robot.

Index

List of symbols

Nomenclature used

Chapter 1

Introduction

1.1 Introduction of mobile robotics

Mobile robot is defined as a machine able to move in the environment that surrounds it, thus distinguishing himself from robotics sets used for holding tasks on site.

Until now we have built robots able to move and work in any environment, from those able to move in water or flying over for space applications, up to the most common robot of movement on the ground, the type that the work here is based on.

Important applications for this last type of robots are in the field of security, such as for reconnaissance and surveillance, and in the military or in general a hazardous place for humans, who may be browsing and making interventions in areas which are potentially harmful to people.

In order to operate effectively in different environments that may require the use of a mobile robot, various modes of locomotion, each with its strengths and weaknesses were studied upon.

There are three categories of so-called ' pure ' locomotion:

- on wheels (wheeled – W)
- dozers (tracked – T)
- with more or less articulated (legged – L)

Locomotion wheels (category W) is the easiest and most efficient. The robots equipped with such a system can reach the highest speed on flat surfaces and require a simpler system of control than the other two categories, but these qualities often contrasts with a limited ability to overcoming obstacles and therefore are not adaptable to different types of environments that may be found to act.

Figure 1.1 a *-Example of wheeled robot*

The presence of tracks (category T) instead, due to increased area of contact with the ground, allows the robot a better grip and a better movement in uneven terrain than on wheels and a decent ability to overcome obstacles, but this result in greater resistance resulting in increased consumption of energy and a limited top speed.

Figure 1.1 b *-Example of robot fitted with Caterpillar tracks*

The legged (category L) movement allows the robot mobility in environments with very irregular terrain, but more number of limbs, is a problem for higher energy consumption and slow movements which are more or less depending on the degree of irregularity of the soil and the complexity of the architecture of the limbs. The control of the various actuators can be also very complex.

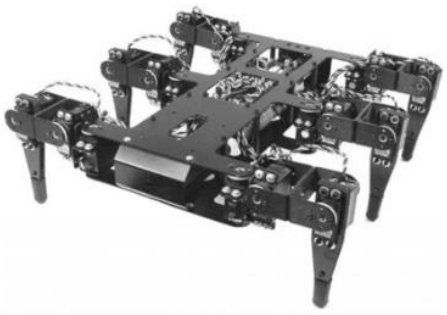

Figure 1.1 c *-Example of robot with legs*

Such strategies of motion can be joined together, thus defining the categories of '*hybrid*' locomotion (**Figure 1.2**), designed to meet the most diverse and special needs for movement and efficiency by the combination of the merits of the categories as well from which they are derived, namely:

- Legs-Wheels (LW)
- Legs-Tracks (LT)
- Wheels-Tracks (WT)
- Legs-Wheels-Tracks (LWT)

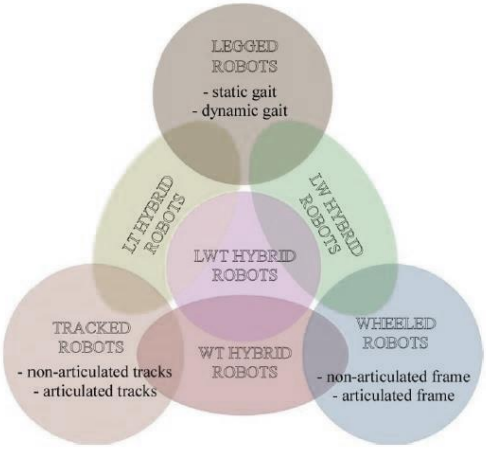

Figure 1.2 *-Categories of ground mobile robots*

An interesting example of robot LW is shown in **Figure 1.3**. The unit is the second version of Mantis hybrid robot, designed for surveillance purposes. It is equipped with two front drive wheels and a couple of rotating legs, which are the characteristic that distinguishes this unit. The locomotion on a flat surface is done purely on wheels while the limbs are used in the presence of an obstacle to be overcome, situation visible in the figure.

Figure 1.3 *-Robot with LW hybrid mobility*

Mobile robot provided with legs and tracks (**LT**) is the category that allows to obtain the best performance in rough environments that present the most difficulty moving, but as for the two categories as well from which they derive, the problem is efficiency and speed. Clear example of this approach is the mobile robot visible in **Figure 1.4.**

Figure 1.4 *– Example of LT hybrid robot*

The combination of tyres and tracks (**WT**) is ultimately best for environments not overly uneven, for example for open spaces, as you reach the objectives of efficiency, speed and adaptability to the lay of the land.

The Daegu Gyeongbuk Institute of Science & Technology has developed a robot in this direction (**Figure 1.5**). The obstacle is tackled with the crawler tracks, while the ability to fold them allows the wheel to move on to better performance and efficiency on flat surfaces.

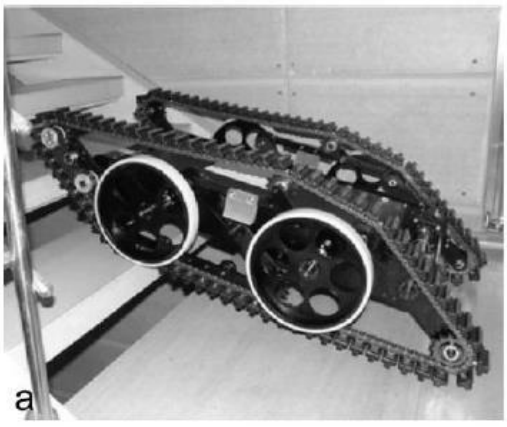

Figure 1.5 – *Example of WT hybrid robot*

When the three categories are joined together at the same time it defines the solution **LWT** which has the possibility to deal with the most varied soil conditions ensuring alternating high adaptability and speed, but can lead to excessive energy consumption due to the mass of the various accessories and locomotor groups that are to be made of the unit and high mechanical complexity which can make it difficult to maintenance or a shift in mechanical architecture.

Azimuth (**Figure 1.6**) is a good example of a medium that combines all three technologies introduced; the unit consists of four joints leg-track-wheel completely independent of each other which can provide the most varied possibilities for overcoming obstacles and handling.

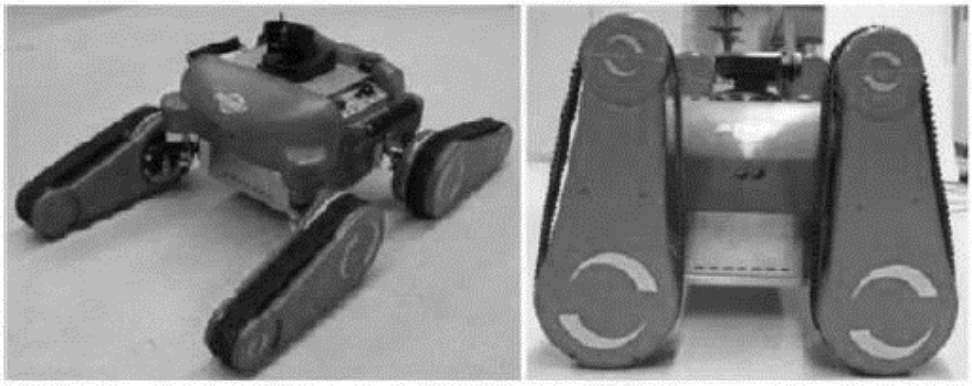

Figure 1.6 *-Robot (LWT) Azimuth*

When selecting the best robot unit for different requirements, it is good to pay attention to mechanical complexity in particular because it has a significant weight on its reliability and can cause considerable time for any repair work in case of failure or Simply for modifications to the actuating or disposition organs of the constituent group.

There is no better strategy for locomotion, but the class to be chosen from time to time is based on priorities and mobility goals and efficiency.

1.2 Introduction of LIDAR sensors

LIDAR also called as Lidar or LiDAR is a surveying method that measures distance to a target by illuminating that target with a pulsed laser light, and measuring the reflected pulses with a sensor. Differences in laser return times and wavelengths can then be used to make digital 2D or 3D-representations of the target. The name *lidar*, sometimes considered an acronym of *Light Detection and Ranging.*

Lidar is popularly used to make high-resolution maps, with applications in geodesy, geomatics, archeology, geography, geology, geomorphology, seismology, forestry, atmospheric physics, laser guidance, airborne laser swath mappling (ALSM), and laser

saltimetry. The technology is also used for control and navigation for some autonomous cars. Lidar sometimes is called *laser scanning* and *3D scanning*, with terrestrial, airborne, and mobile applications.

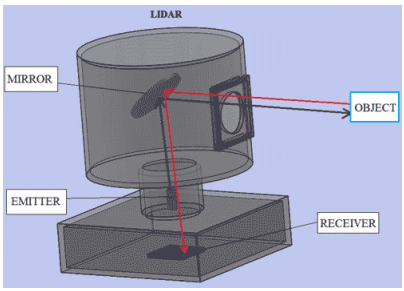

Figure 1.7 *– LIDAR sensor working kayout*

The LiDAR instrument fires rapid pulses of laser light at a surface, some at up to 150,000 pulses per second. A sensor on the instrument measures the amount of time it takes for each pulse to bounce back. Light moves at a constant and known speed so the LiDAR instrument can calculate the distance between itself and the target with high accuracy. By repeating this in quick succession the insturment builds up a complex 'map' of the surface it is measuring. With airborne LiDAR other data must be collected to ensure accuracy. As the sensor is moving height, location and orientation of the instrument must be included to determine the position of the laser pulse at the time of sending and the time of return. This extra information is crucial to the data's integrity. With ground based LiDAR a single GPS location can be added for each location where the instrument is set up.

Generally there are two types of LiDAR detection methods. Direct energy detection, also known as incoherent, and Coherent detection. Coherent systems are best for Doppler or phase sensitive measurements and generally use Optical heterodyne detection. This allows them to operate at much lower power but has the expense of more complex transceiver requirements. In both types of LiDAR there are two main pulse models: micropulse and high-energy systems. Micropulse systems have developed as a result of more powerful computers with greater computational capabilities. These lasers are lower powered and are classed as 'eye-safe' allowing them to be used with little safety precautions. High energy systems are more commonly used for atmospheric research where they are often used for measuring a variety of atmospheric parameters such as the height, layering and density of clouds, cloud particles properties, temperature, pressure, wind, humidity and trace gas concentration.

Chapter 2

State of Art

- **Types of Sensors (For Environment Mapping)**
- **Types of Algorithm**
- **Path Planning Approach**

2.1 Types of Sensors

There are many types of sensors that can be used for Environment Mapping, Obstacle avoidance, Path planning, Localisation and etc. So we will only read about Distance sensor.

Most proximity sensors can also be used as distance sensors, or commonly known as Range sensors.

- **Lidar-** Lidar sensors are also called LIDAR, LiDAR and LADAR. Lidar stands for Light Detection and Ranging. This type of sensor works with the use of a laser in the setup which will emit laser pulse on the object and when the laser pulse comes back it is intercepted by a receiver in the setup which will then analyse the data of the objects distance and the angle of the object with respect to the Lidar sensor or a robot on which the sensor has been mounted. This is the most commonly used sensor for mapping the environment. These sensors are also the cheapest sensors with basic function capabilities and good output feedback available in the market for beginner. There are two types of Lidar sensors 2D sensors and 3D sensors. In this paper we are using a 2D scanning sensor.

- **Infrared Distance Sensor-** IR circuits are designed on triangulation principle for distance measurement. A transmitter sends a pulse i=of IR signals which is detected by the receiver if there is an obstacle and based on the angle the signal is received and the distance is calculated. SHARP has a family of IR transceivers which are very useful for distance measurements. A simple transmit and receive using a couple of transmitters and receivers will still do the job of distance measurements, but if you require precision, then prefer the triangulation method.

- **Ultrasonic Distance Sensors-** The sensor emits an ultrasonic pulse and is captured by a receiver. Since the speed of sound is almost constant in air, which is 344m/s, the time between send and receive is calculated to give the distance between your robot and the obstacle. Ultrasonic distance sensors are especially useful for underwater robots.

- **Stereo Camera-** Two cameras placed adjacent to each other can provide depth information using its stereo vision. Processing the data received from a camera is difficult for a robot with minimal processing power and memory. If opted for, they make a valuable addition to your robot.

- **Encoders-** Encoders are not actually sensors, but a combination of different components which convert angular position of a shaft or wheel into an analog or digital code. The most popular encoder is an optical encoder which includes a rotational disk, light source and a light detector. The rotational disk has transparent and opaque pattern or sometimes its black and white pattern painted or printed over it. When the disk rotates along with the wheel the emitted light is interrupted generating a signal output. The number of times the interruption happens and the diameter of the wheel can together give the distance travelled by the robot.

We can also include Navigation/Positioning sensors in the list but they are more useful in an open environment, but as this experiment is conducted in a closed environment we have no use of such sensors here.

2.2 Types of Algorithm

There are many types of algorithm for SLAM (Simultaneous localization and mapping) which are listed below:

- EKF SLAM
- FastSLAM
- L-SLAM
- GraphSLAM
- Occupancy Gird SLAM
- DP-SLAM
- Parallel Tracking and Mapping (PTAM)
- LSD-SLAM
- S-PTAM
- ORB-SLAM
- ORB-SLAM2
- MonoSLAM
- CoSLAM
- SeqSLAM
- iSAM (Incremental Smoothing and Mapping)

Now we will talk about a few methods only, one of which is EKF SLAM which we are using for our experiment.

1. **EKF SLAM-** In robotics, EKF SLAM is a class of algorithms which utilizes the extended Kalman filter (EKF) for simultaneous localization and mapping (SLAM). Typically, EKF SLAM algorithms are feature based, and use the maximum likelihood algorithm for data association. EKF SLAM is based on Kalman filtering, also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe. The filter is named after Rudolf E. Kálmán, one of the primary developers of its theory.

   The Kalman filter has numerous applications in technology. A common application is for guidance, navigation, and control of vehicles, particularly aircraft and spacecraft. Furthermore, the Kalman filter is a widely applied concept in time series analysis used in fields such as signal processing and econometrics. Kalman filters also are one of the main topics in the field of robotic motion planning and control, and they are sometimes included in trajectory optimization. The Kalman filter also works for modeling the central nervous system's control of movement. Due to the time delay between issuing motor commands and receiving sensory feedback, usage of the Kalman filter supports the realistic model for making estimates of the current state of the motor system and issuing updated commands.

   The algorithm works in a two-step process. In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement (necessarily corrupted with some amount of error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty. The algorithm is recursive. It can run in real time, using only the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required.

   Associated with the EKF is the gaussian noise assumption, which significantly impairs EKF SLAM's ability to deal with uncertainty. With greater amount of uncertainty in the posterior, the linearization in the EKF fails.

2. **FastSLAM-** The key idea of FastSLAM exploits the fact that knowledge of the robot's path $s_1, s_2, \ldots, s_t$ renders the individual landmark measurements independent, as originally observed by Murphy. FastSLAM decomposes the SLAM problem into one robot localization problem, and a collection of K landmark estimation problems.

   In FastSLAM, alike in EKF SLAM, poses are assumed to behave according to a probabilistic law named motion model with an underlying density

   $$p(s_t \mid s_{t-1}).$$

Likewise, the measurements are governed by the (probabilistic) measurement model $p(z_t \mid s_t, \theta, n_t)$ with z_t measurement, $\theta = \{\theta_1, \ldots, \theta_K\}$ the set of landmarks, and $n_t \in \{1, \ldots, K\}$ the index of the observed landmark at time t (only one at a time). The ultimate goal is to estimate the posterior $p(s^t, \theta \mid z^t)$.

Basically, EKF SLAM and FastSLAM solve the same problem while making use of the identical probabilistic motion and measurement models. Furthermore, both use Kalman filtering: EKF SLAM applies the filter once to a high dimensional filtering problem where FastSLAM employes $M \cdot K$ tiny EKFs (K of them in each particle).

3. **GraphSLAM-** In robotics, GraphSLAM is a Simultaneous localization and mapping algorithm which uses sparse information matrices produced by generating a factor graph of observation interdependencies (two observations are related if they contain data about the same landmark). Where a factor graph is a bipartite graph representing the factorization of a function. In probability theory and its applications, factor graphs are used to represent factorization of a probability distribution function, enabling efficient computations, such as the computation of marginal distributions through the sum-product algorithm. One of the important success stories of factor graphs and the sum-product algorithm is the decoding of capacity-approaching error-correcting codes, such as LDPC and turbo codes.

   Factor graphs generalize constraint graphs. A factor whose value is either 0 or 1 is called a constraint. A constraint graph is a factor graph where all factors are constraints. The max-product algorithm for factor graphs can be viewed as a generalization of the arc-consistency algorithm for constraint processing.

4. **Parallel Tracking and Mapping (PTAM)-** PTAM is a monocular SLAM (Simultaneous Localization and Mapping) system useful for real-time 6-DOF camera tracking in small scenes. It requires no markers, pre-made maps, known templates, or inertial sensors. It was originally developed as a research system in the Active Vision Laboratory of the University of Oxford.

5. **LSD-SLAM-** LSD-SLAM is a novel, direct monocular SLAM technique. Instead of using keypoints, it directly operates on image intensities both for tracking and mapping. The camera is tracked using direct image alignment**,** while geometry is estimated in the form of semi-dense depth maps, obtained by filtering over many pixelwise stereo comparisons. We then build a Sim pose-graph of keyframes, which allows to build scale-drift corrected, large-scale maps including loop-closures. LSD-SLAM runs in real-time on a CPU, and even on a modern smartphone.

2.3 Path Planning Approach

The path planning approach for the experiment is based on EKF-SLAM (Extended Kalman Filter). We will be working in a closed environment and trying to determine the

shortest route for the robot to reach its final destination while avoiding the objects in the environment. So we are going to use a LIDAR sensor in ROS to perform our experiment and test the algorithm proposed to achieve the maximum efficiency while path planning and obstacle avoidance in a closed environment.

In the experiment we will use a simple robot mule or do the experiment on ROS on virtual machine and Matlab. But in order to test the algorithm and incure lower cost of working we are going to use ROS Indigo on Virtual machine for our experiment.

Our approach of the algorithm is based on EKF-SLAM for mapping. EKF SLAM is a class of algorithms which utilizes the extended Kalman filter (EKF) for simultaneous localization and mapping (SLAM). Typically, EKF SLAM algorithms are feature based, and use the maximum likelihood algorithm for data association. EKF SLAM is based on Kalman filtering, also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe.

For the past decade, the EKF SLAM has been the de facto method for SLAM, until the introduction of FastSLAM. But we are not going to use FastSLAM for our experiment.

Chapter 3

RPLIDAR A1

3.1 Introduction to RPLIDAR A1

RPLIDAR A1 is a low cost 360 degree 2D laser scanner (LIDAR) solution developed by SLAMTEC. The system can perform 360 degree scan within 6 meter range. The produced 2D point cloud data can be used in mapping localization and object/environment modeling.

Until RPLIDAR A1's scanning frequency is 5.5 hz when sampling 360 points each round. And it can be configured up to 10 hz maximum.

RPLIDAR A1 is basically a laser triangulation measurement system. It can work excellent in all kinds of indoor environment and outdoor environment without sunlight.

3.1.1 System Connection

RPLIDAR A1 contains a range scanner system and a motor system. After power on each sub-system, RPLIDAR A1 starts rotating and scanning clockwise. User can get range scan data through the communication interface (Serial port/USB).

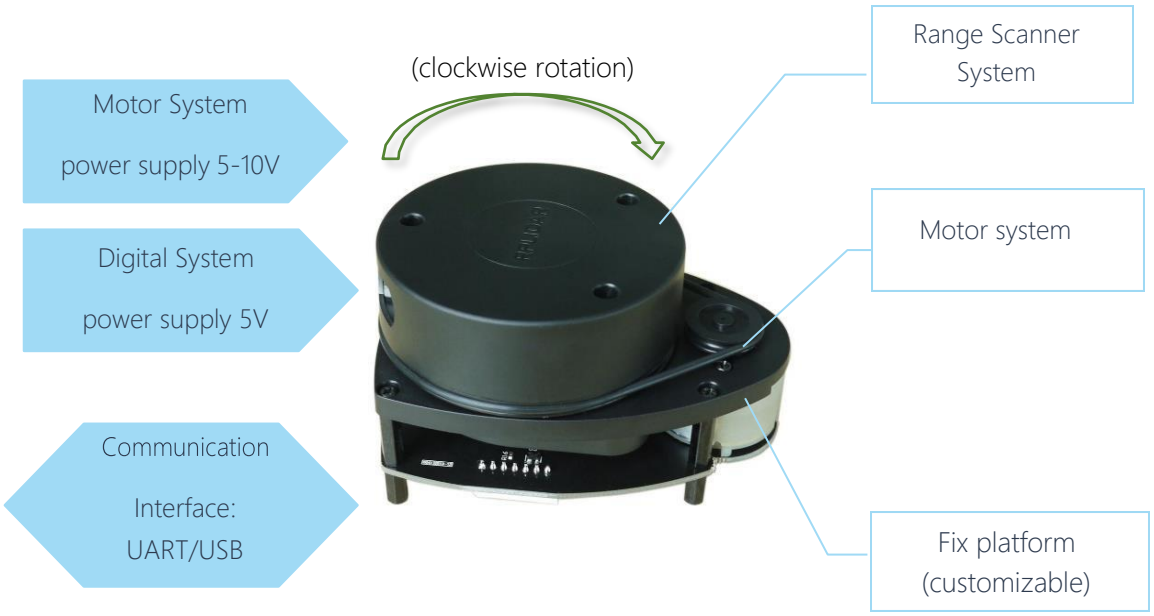

Figure 3.1 *- RPLIDAR A1 System Composition*

RPLIDAR A1 comes with a speed detection and adaptive system. The system will adjust frequency of laser scanner automatically according to motor speed. And host system can get RPLIDAR A1's real speed through communication interface.

The simple power supply scheme saves LIDAR system's BOM cost and makes RPLIDAR A1 much easier to use.

3.1.2 Mechanism

RPLIDAR A1 is based on laser triangulation ranging principle and uses high-speed vision acquisition and processing hardware developed by SLAMTEC. The system measures distance data in more than 2000 times per second and high resolution distance output (<1% of the distance).

RPLIDAR emits modulated infrared laser signal and the laser signal is then reflected by the object to be detected. The returning signal is sampled by vision acquisition system in RPLIDAR A1 and the DSP embedded in RPLIDAR A1 start processing the sample data and output distance value and angle value between and RPLIDAR A1 through communication interface.

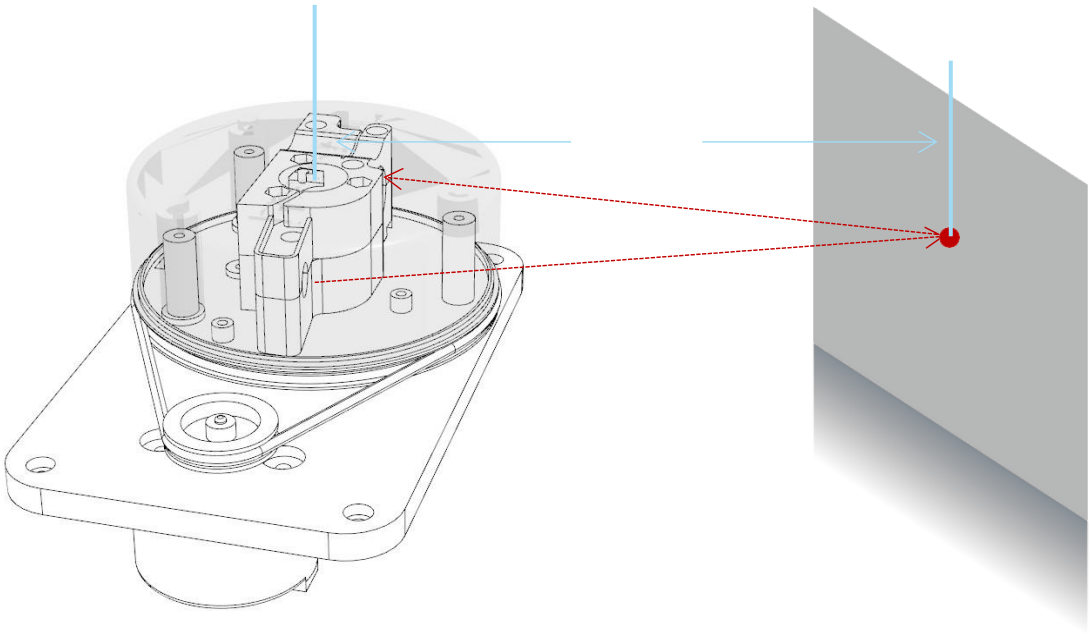

Figure 3.2 *– The RPLIDAR A1 Working Schematic*

The high-speed ranging scanner system is mounted on a spinning rotator with a build-in angular encoding system. During rotating, a 360 degree scan of the current environment will be performed.

3.1.3 Safety and Scope

RPLIDAR A1 system use a low power (<5mW) infrared laser as its light source, and drives it using modulated pulse. The laser emits in a very short time frame which can make sure its safety to human and pet and reach Class I laser safety standard.

The modulatedd laser can effectively prevent ambient light and sunlight during ranging scanning process. This makes RPLIDAR A1 work excellent in all kinds of indoor environment and outdoor environment without sunlight.

3.1.4 Data Output

When RPLIDAR A1 is working, sampling data will output to communication interface. Each sample point contains information. RPLIDAR A1 outputs sampling data continuously. Host systems can configure output format and stop RPLIDAR A1 by sending stop command.

Data Type	Unit	Description
Distance	mm	Current measured distance value between the rotating core of the RPLIDAR A1 and the sampling point
Heading	degree	Current heading angle of the measurement
Quality	level	Quality of the measurement
Start Flag	(Boolean)	Flag of a new scan

Figure 3.3 *– The RPLIDAR A1 Sample Point Data Information*

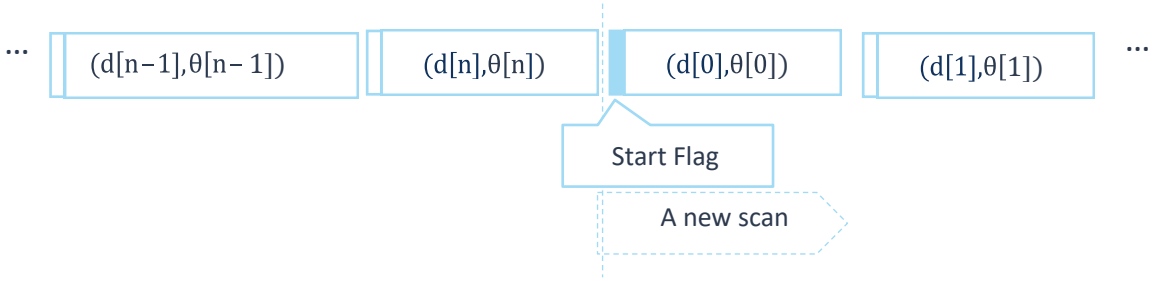

Figure 3.4 *– The RPLIDAR A1 Sample Point Data Frames*

3.1.5 Application Scenarios

The RPLIDAR A1 can be used in the following application scenarios:

- Home service / cleaning robot navigation and localization
- General robot navigation and localization
- Smart toys localization and obstacle avoidance
- Environment scanning and 3D re-modeling
- General simultaneous localization and mapping (SLAM)

Chapter 4

Experimental Test

4.1 Calibration of RPLIDAR A1

In order to calibrate the sensor we need to test it in an undisturbed control working environment. An environment where we can ensure high data accuracy during testing and low chances of corruption.

For this test I performed it in the University Laboratory for better outcome. We will use simple measuring tools to take measurements of the angle, distance and height and compare it to the data string from the sensor outcome.

Equipment's used:

- RPLIDAR A1 sensor
- Measuring tape of 2m length
- Large scale protractor

Experiment (Date- 29/05/2017)

Sensor Data			Experiment Data		
Angle (Degree)	**Distance (mm)**	**Quality**	**Angle (Degree)**	**Distance (mm)**	**Height (mm)**
4.2813	668.3	14	4	663.5	33
5.2344	663.3	14	5	661	34
6.3125	671	11	6	666.2	37
7.1406	722.8	32	7	720.5	43
8.2344	723.5	33	8	721	47
9.1719	725.8	29	9	723.5	40
10.2188	728	29	10	726	42
11.1719	729.8	26	11	728	40
12.1563	732.3	28	12	730.5	43
13.2188	733.5	26	13	732	41
14.1406	736.5	25	14	735	40
15.25	739.8	25	15	738	39
16.125	742.5	25	16	741.5	37
17.1875	746	25	17	744.5	37
18.1719	750.3	24	18	749	35

Figure 4.1 *– The Calibration test 1 data table for RPLIDAR A1*

The experiment was performed in a control environment in a laboratory with fixed objects to be taken as the data points.

We took 15 sample points to compare the data with. These points where fixed points in the real world reference frame which reduced the possibility of high error. These points were of objects in the lab such as, walls, boxes, beams and etc. They were all of made up of different material thus resulting in varying quality of the signal received back by the sensor.

We cannot compare Quality string of data as we cannot measure it in real world.

Here we see that in Experimental data we have measured the height at which the sensor pings the data point in the environment.

Graphs

Now we will see the data comparison in between the data accumulated by the sensor and the data obtained in the experiment results. We will compare the data in reference to the Distance and the corresponding Angle of a point. This will result in two identical lines. The gap between them will be the error between the two sets of data.

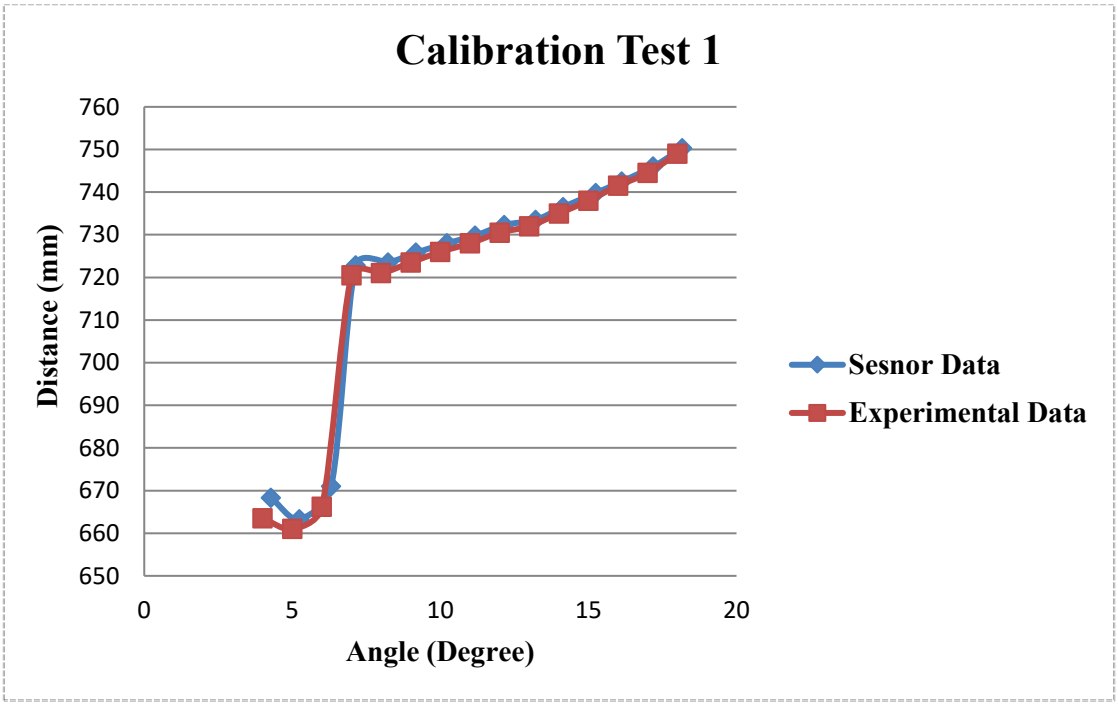

Figure 4.2 *– The Calibration test 1 line graph of RPLIDAR A1*

Below is a line graph plot in MATLAB of the data accumulated from the above experiment. The line graph here is based on sensor data of 246 points it accumulated in 360 Degree field of view. We have Angle (Degree) of the point with respect to the

sensor on X-axis and Distance (mm) of the corresponding point with respect to the sensor on Y-axis on the graph.

Every flat plane in the graph can be read as an object with many consecutive data points forming a sensor facing surface. And every sudden rise or fall in peak can be read as two different data points on two different objects.

MATLAB-Plot

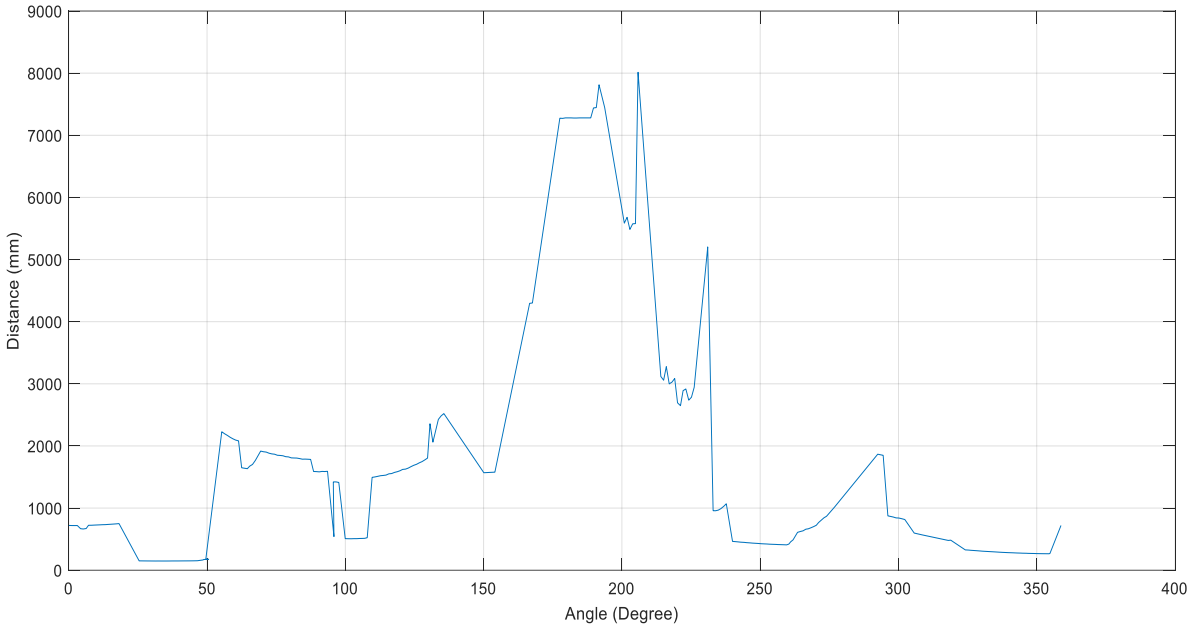

Figure 4.3 *– MATLAb grah for all data points of RPLIDAR A1*

Below is the simulated image of data points that were processed by the SLAMTEC provided SDK tool for RPLIDAR A1 known as Frame Grabber. This image is of the experiment that was performed to gather data and calibrate the sensor.

Here you can see various data points around the sensor with the scaled graph showing the distance and angle reference for each point in the environment. On top left of the screen you can see the mouse pointer's current distance and its corresponding angle in red. Frequency and the rotor speed below it in white.

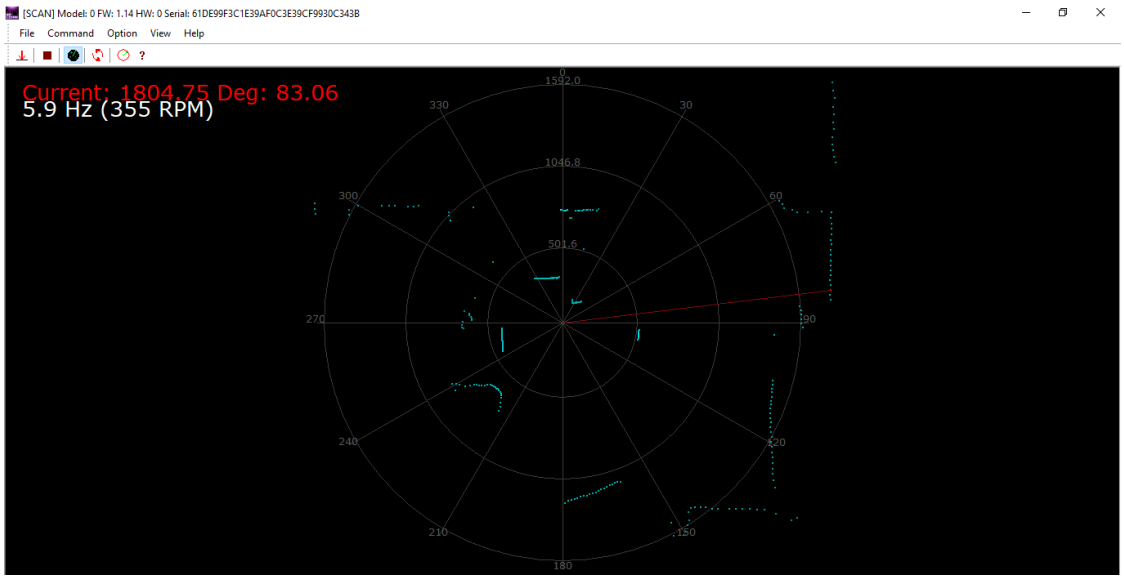

Figure 4.4 *– Screen grab from Frame Grabber tool.*

4.2 Calibration test 2 of RPLIDAR A1

In this calibration test we will check the corresponding range to specific degree angle of the data point. This will allow us to see the variation in change of data with respect to the distance of the object from the sensor.

Experiment (Date- 01/06/2017)

Angle (Degree)	Distance Data 1 (mm)	Distance Data 2 (mm)	Distance Data 3 (mm)
4.2813	668.3	768.3	868.2
5.2344	663.3	763.3	863.2
6.3125	671	771	870.9
7.1406	722.8	822.9	922.9
8.2344	723.5	823.5	923.4
9.1719	725.8	825.8	9259
10.2188	728	828	928
11.1719	729.8	829.7	928.8

Figure 4.5 *– The Calibration test 2 data table for RPLIDAR A1*

The experiment was performed in a control environment in a laboratory with fixed objects to be taken as the data points.

We took 8 sample points of an object to compare as the distance is progressed shifting the object away from the sensor 100mm for every data set. These points where fixed points in the real world reference frame which reduced the possibility of high error.

Here we will see small error that has occurred in data set as the object is moved away from the sensor. The error is very small around 0.1mm. This shows that the sensor is very accurate and has very low chances of error.

Graphs

Now we will see the data comparison in between the different data sets of distance with corresponding angle registered by the sensor. As we plot the graph we will see that the lines are identical and have very little error.

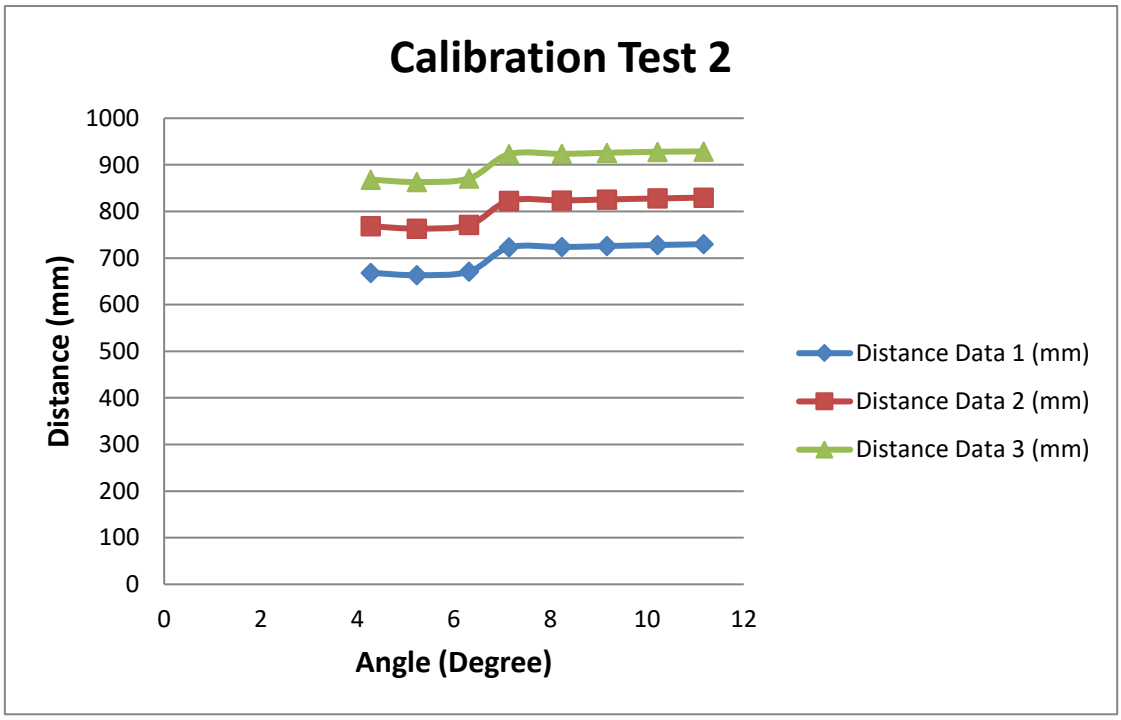

Figure 4.6 *– The Calibration test 2 line graph of RPLIDAR A1*

4.3 Result

We learned from the experiment that the values were within 0.5 degree and up-to 5 mm of tolerance to the data obtained from the sensor. The height measured is also ranging from 33mm-47mm which is within the parameters of the RPLIDAR A1 sensor. This falls well within the parameters of the sensor tolerance. Thus the sensor is very accurate as the tolerance is less than 1% as the company specifies. The quality of the signal is subjective to the material of the object, the angle of attack and the surface reflective quality of the object dependent on material and shape.

Chapter 5

Environment Mapping

5.1 SLAM Mapping

Here in this experiment we are going to map the environment in which the experiment was performed for our calibration exercise.

We will use the sensor data and use it to map the environment.

The experiment is based on the RPLIDAR reading points in the environment and then those points being used to form surface of obstacles in the environment. Here all those points which will be too close to each other and consecutive will be formed into line segments of the surface of the objects. We will use an algorithm which will process the images and knit them together closely forming a video which will show us the outcome as mapping being done.

This practice of mapping will be further used in path planning of a robot.

5.2 Environmental Setup

Here we will see how the environment was setup during the experiment.

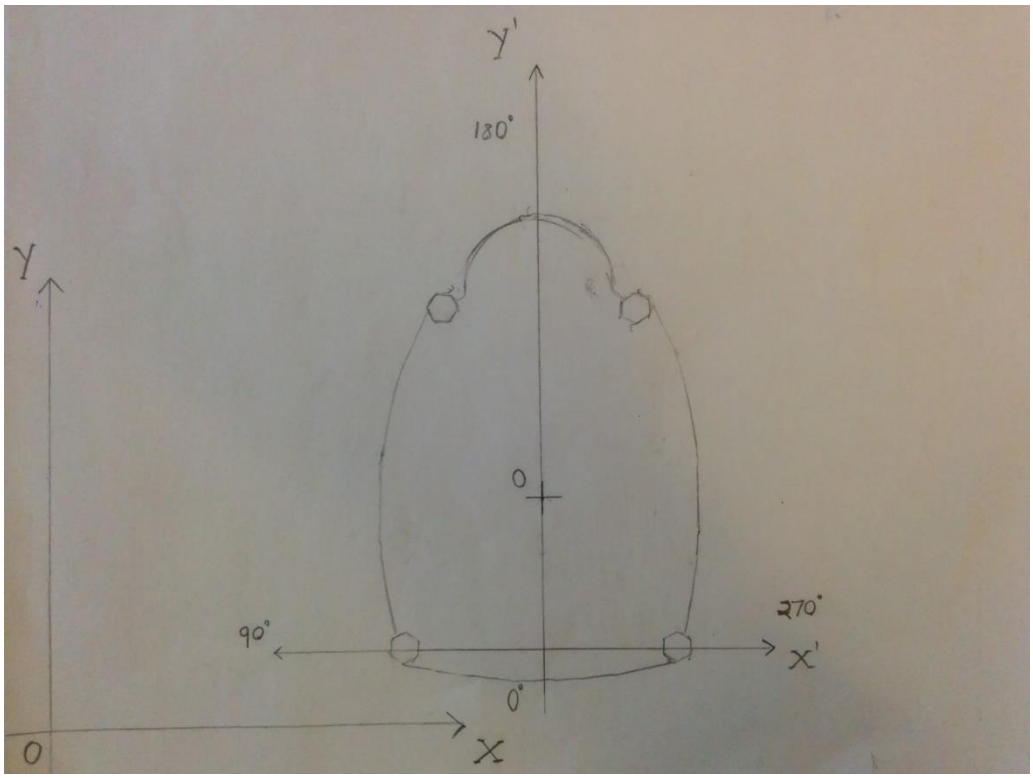

Figure 5.1 *– Scale used for RPLIDAR A1 sensor in experiment.*

This scale was used as a reference and mounting point in the environment for the sensor in order to make sure that the data was not corrupted in repeated experimentation.

In the experiment we had setup 3 fixed objects as to compare data and ensure result reliability.

Two objects were cardboard boxes places at 90 degrees apart with respect to the sensor and at varying distance from the sensor. While the third object was an iron beam near a wall.

These different objects placed with different angles with respect to the sensor resulted in varying quality of the sensor data.

In this experiment we will map the environment and do path planning with the use sensor data processed through coding in MATLAB. This code is based on EKF-SLAM which helps us in using the data from the sensor such as distance and the bearing of the object that has reflected the laser of the sensor. The purpose of doing so is to find an effective functioning code which can process the data from the sensor and be used in acquiring the image of the environment. This can further lead us to read the data in real time and do path planning which can be very useful in automation of the robot or the automotive medium to navigate its path through obstacles and reach its destination in the most efficient way possible.

5.3 MATLAB Code

Functions to perform 2D EKF-SLAM (Extended Kalman Filter – Simultaneous Localization and Mapping) with a range-and-bearing (Distance-and-Angle) sensor are given below. In here we have differentiated between elementary function blocks and the function block SLAM really needs. The SLAM functions are often compositions of elementary functions. All functions are able to return the Jacobian matrices of the output variables with respect to each input variables.

Finally, we give a simple but sufficient code of a full working SLAM algorithm, with simulation, estimation and graphics output.

5.4 Frame transformations

Express a global point in a local frame:

```
function [pf, PF_f, PF_p] = toFrame(F , p)
% TO THE FRAME - Transform point P from global frame to frame F
%
% In:
%   F : reference frame          F = [f_x ; f_y ; f_alpha]
%   p : point in global frame     p = [p_x ; p_y]
% Out:
%   pf: point in frame F
%   PF_f: Jacobian wrt F
%   PF_p: Jacobian wrt p
t = F(1:2);
a = F(3);
R = [cos(a) -sin(a) ; sin(a) cos(a)];
pf = R' * (p - t);
if nargout > 1                                % Jacobians requested
   px = p(1);
   py = p(2);
   x = t(1);
   y = t(2);
   PF_f = [...
      [ -cos(a), -sin(a), cos(a)*(py - y) - sin(a)*(px - x)]
      [ sin(a), -cos(a), - cos(a)*(px - x) - sin(a)*(py - y)]];
   PF_p = R';
end
end
function f()
%% Symbolic code below -- Generation and/or test of Jacobians
% - Enable 'cell mode' to use this section
% - Left?click once on the code below ? the cell should turn yellow
% - Type ctrl+enter (Windows, Linux) or Cmd+enter (MacOSX) to execute
% - Check the Jacobian results in the Command Window.
syms x y a px py real
F = [x y a]';
p = [px py]';
pf = toFrame(F, p);
PF_f = jacobian(pf, F)
End
```

Express a local point in global frame:

```
function [pw, PW_f, PW_pf] = fromFrame(F, pf)
% FROM THE FRAME - Transform a point PF from local frame F to the global frame.
%
% In:
%   F : reference frame     F = [f_x ; f_y ; f_alpha]
%   pf: point in frame F    pf = [pf_x ; pf_y]
% Out:
%   pw: point in global frame
%   PW_f: Jacobian wrt F
%   PW_pf: Jacobian wrt pf
t = F(1:2);
```

```
a = F(3);
R = [cos(a) -sin(a) ; sin(a) cos(a)];
pw = R*pf + repmat(t,1,size(pf,2));     % Allow for multiple points
if nargout > 1                          % Jacobians requested
   px = pf(1);
   py = pf(2);
   PW_f = [...
      [ 1, 0, - py*cos(a) - px*sin(a)]
      [ 0, 1, px*cos(a) - py*sin(a)]];
   PW_pf = R;
end
end
function f()
%% Symbolic code below -- Generation and/or test of Jacobians
% - Enable 'cell mode' to use this section
% - Left-click once on the code below - the cell should turn yellow
% - Type ctrl+enter (Windows, Linux) or Cmd+enter (MacOSX) to execute
% - Check the Jacobian results in the Command Window.
syms x y a px py real
F = [x;y;a];
pf = [px;py];
pw = fromFrame(F,pf);
PW_f = jacobian(pw,F)
PW_pf = jacobian(pw,pf)
end
```

- **Project to sensor**

```
function [y, Y_p] = scan (p)
% SCAN - Perform a range-and-bearing measure of a 2D point.
%
% In:
%   p : point in sensor frame   p = [p_x ; p_y]
% Out:
%   y : measurement           y = [range ; bearing]
%   Y_p: Jacobian wrt p
px = p(1);
py = p(2);
d = sqrt(px^2+py^2);
a = atan2(py,px);
% a = atan(py/px); % use this line if you are in symbolic mode.
y = [d;a];
if nargout > 1                          % Jacobians requested
   Y_p = [...
      px/sqrt(px^2+py^2) , py/sqrt(px^2+py^2)
      -py/(px^2*(py^2/px^2 + 1)), 1/(px*(py^2/px^2 + 1)) ];
end
end
function f()
%% Symbolic code below -- Generation and/or test of Jacobians
% - Enable 'cell mode' to use this section
% - Left-click once on the code below - the cell should turn yellow
% - Type ctrl+enter (Windows, Linux) or Cmd+enter (MacOSX) to execute
% - Check the Jacobian results in the Command Window.
syms px py real
```

```
p = [px;py];
y = scan(p);
Y_p = jacobian(y,p)
[y,Y_p] = scan(p);
simplify(Y_p - jacobian(y,p))
end
```

- **Back project from sensor**

```
function [p, P_y] = invScan(y)
% INVERSE SCAN - Backproject a range-and-bearing measure into a 2D point.
%
% In:
%   y : range?and?bearing measurement     y = [range ; bearing]
% Out:
%   p : point in sensor frame            p = [p_x ; p_y]
%   P_y: Jacobian wrt y
d = y(1);
a = y(2);
px = d*cos(a);
py = d*sin(a);
p = [px;py];
if nargout > 1                              % Jacobians requested
   P_y = [...
      cos(a) , -d*sin(a)
      sin(a) , d*cos(a)];
end
```

5.5 SLAM level operations

- **Robot motion**

```
function [ro, RO_r, RO_n] = move(r, u, n)
% MOVE - Robot motion, with separated control and perturbation inputs.
%
% In:
%   r: robot pose          r = [x ; y ; alpha]
%   u: control signal      u = [d_x ; d_alpha]
%   n: perturbation, additive to control signal
% Out:
%   ro: updated robot pose
%   RO_r: Jacobian d(ro) / d(r)
%   RO_n: Jacobian d(ro) / d(n)
a = r(3);
dx = u(1) + n(1);
da = u(2) + n(2);
ao = a + da;
if ao > pi
   ao = ao - 2*pi;
end
if ao < -pi
   ao = ao + 2*pi;
end
```

```
% build position increment dp=[dx;dy], from control signal dx
dp = [dx;0];
if nargout == 1                         % No Jacobians requested
   to = fromFrame(r, dp);
else                                    % Jacobians requested
   [to, TO_r, TO_dt] = fromFrame(r, dp);
   AO_a = 1;
   AO_da = 1;
   RO_r = [TO_r ; 0 0 AO_a];
   RO_n = [TO_dt(:,1) zeros(2,1) ; 0 AO_da];
end
ro = [to;ao];
```

- **Direct observation model**

```
function [y, Y_r, Y_p] = observe(r, p)
% OBSERVE - Transform a point P to robot frame and take a range-and-bearing measurement.
% In:
%   r : robot frame          r = [r_x ; r_y ; r_alpha]
%   p : point in global frame   p = [p_x ; p_y]
% Out:
%   y: range?and?bearing measurement
%   Y_r: Jacobian wrt r
%   Y_p: Jacobian wrt p
if nargout == 1                         % No Jacobians requested
   y = scan(toFrame(r,p));
else                                    % Jacobians requested
   [pr, PR_r, PR_p] = toFrame(r, p);
   [y, Y_pr] = scan(pr);
                                        % The chain rule!
   Y_r = Y_pr * PR_r;
   Y_p = Y_pr * PR_p;
end
```

- **Inverse observation model**

```
function [p, P_r, P_y] = invObserve(r, y)
% INVERSE OBSERVE - Backproject a range-and-bearing measurement and transform to map frame.
%
% In:
%   r : robot frame     r = [r_x ; r_y ; r_alpha]
%   y : measurement     y = [range ; bearing]
% Out:
%   p : point in sensor frame
%   P_r: Jacobian wrt r
%   P_y: Jacobian wrt y
if nargout == 1                         % No Jacobians requested
   p = fromFrame(r, invScan(y));
else                                    % Jacobians requested
   [p_r, PR_y] = invScan(y);
   [p, P_r, P_pr] = fromFrame(r, p_r);
                                        % here the chain rule !
   P_y = P_pr * PR_y;
end
end
function f()
```

```
%% Symbolic code below -- Generation and/or test of Jacobians
% - Enable 'cell mode' to use this section
% - Left-click once on the code below - the cell should turn yellow
% - Type ctrl+enter (Windows, Linux) or Cmd+enter (MacOSX) to execute
% - Check the Jacobian results in the Command Window.
syms rx ry ra yd ya real
r = [rx;ry;ra];
y = [yd;ya];
[p, P_r, P_y] = invObserve(r, y);        % We extract also the coded Jacobians P_r and P_y
                                         % We use the symbolic result to test the coded Jacobians
simplify(P_r - jacobian(p,r))  % zero?matrix if coded Jacobian is correct
simplify(P_y - jacobian(p,y)) % zero?matrix if coded Jacobian is correct
end
```

5.6 EKF-SLAM code

This follows a 102-lines-of-code m-file performing SLAM. This code uses all the files above, plus the helper function **cloister.m** (also given below) which is just used to define the set of landmarks for the simulation.

HELP NOTES:

1. The robot state is defined by [xr;yr;ar] with [xr;yr] the position and [ar] the orientation angle in the plane.

2. The landmark states are simply Li=[xi;yi]. There are a number of N landmarks organized in a 2-by-N matrix W=[L1 L2 ... Ln] so that Li = W(:,i).

3. The control signal for the robot is U=[dx;da] where [dx] is a forward motion and [da] is the angle of rotation.

4. The motion perturbation is additive Gaussian noise n=[nx;na] with covariance Q, which adds to the control signal.

5. The measurements are range-and-bearing Yi=[di;ai], with [di] the distance from the robot to landmark Li, and [ai] the bearing angle from the robot's x-axis.

6. The simulated variables are written in capital letters,
   R: robot
   W: set of landmarks or 'world'
   Y: set of landmark measurements Y=[Y1 Y2 ... YN]

7. The true map is [xr;yr;ar;x1;y1;x2;y2;x3;y3; ... ;xN;yN]

8. The estimated map is Gaussian, defined by
   x: mean of the map
   P: covariances matrix of the map

9. The estimated entities (robot and landmarks) are extracted from {x,P} via pointers, denoted in small letters as follows:

r: pointer to robot state. r=[1,2,3]
l: pointer to landmark i. We have for example l=[4,5] if i=1,
l=[6,7] if i=2, and so on.
m: pointers to all used landmarks.
rl: pointers to robot and one landmark.
rm: pointers to robot and all landmarks (the currently used map).

Therefore: x(r) is the robot state,
x(l) is the state of landmark i
P(r,r) is the covariance of the robot
P(l,l) is the covariance of landmark i
P(r,l) is the cross?variance between robot and lmk i
P(rm,rm) is the current full covariance -- the rest is unused.

10. Managing the map space is done through the variable mapspace. mapspace is a logical vector the size of x.
    If mapspace(i) = false, then location i is free.
    Oterwise mapspace(i) = true. Use it as follows:
    * query for n free spaces: s = find(mapspace==false, n);
    * block positions indicated in vector s: mapspace(s) = true;
    * liberate positions indicated in vector s: mapspace(s) = false;

11. Managing the existing landmarks is done through the variable landmarks.
    Landmarks is a 2-by-N matrix of integers. l=landmarks(:,i) are the pointers of landmark i in the state vector x, so that x(l) is the state of landmark i.
    Use it as follows:
    * query 1 free space for a new landmark: i = find(landmarks(1,:)==0,1)
    * associate indices in vector s to landmark i: landmarks(:,i) = s
    * liberate landmark i: landmarks(:,i) = 0;

12. Graphics objects are Matlab 'handles'. See Matlab doc for information.

13. Graphic objects include:
    RG: simulated robot
    WG: simulated set of landmarks
    rG: estimated robot
    reG: estimated robot ellipse
    lG: estimated landmarks
    leG: estimated landmark ellipses

```
% SLAM 2D - A 2D EKF-SLAM algorithm with simulation and graphics.
%
% I. INITIALIZE
% I.1 SIMULATOR -- use capital letters for variable names
% W: set of external landmarks
W = cloister(-4,4,-4,4,7);       % Type 'help cloister' for help
% N: number of landmarks
N = size(W,2);
% R: robot pose [x ; y ; alpha]
R = [0;-2;0];
% U: control [d_x ; d_alpha]
U = [0.1 ; 0.05];                          % fixing advance and turn increments creates a circle
```

```
% Y: measurements of all landmarks
Y = zeros(2, N);
% I.2 ESTIMATOR
% Map: Gaussian {x,P}
% x: state vector's mean
x = zeros(numel(R)+numel(W), 1);
% P: state vector's covariances matrix
P = zeros(numel(x),numel(x));
% System noise: Gaussian {0,Q}
q = [.01;.02];                  % amplitude or standard deviation
Q = diag(q.^2);                 % covariances matrix
% Measurement noise: Gaussian {0,S}
s = [.1;1*pi/180];              % amplitude or standard deviation
S = diag(s.^2);                 % covariances matrix
% Map management
mapspace = false(1,numel(x));   % See Help Note #10 above.
% Landmarks management
landmarks = zeros(2, N);        % See Help Note #11 above
% Place robot in map
r = find(mapspace==false, numel(R) ); % set robot pointer
mapspace(r) = true;             % block map positions
x(r) = R;                       % initialize robot states
P(r,r) = 0;                 % initialize robot covariance
% I.3 GRAPHICS -- use the variable names of simulated and estimated variables, followed by a capital G
to indicate 'graphics'.
% NOTE: the graphics code is long but absolutely necessary.
% Set figure and axes for Map
mapFig = figure(1);         % create figure
cla                             % clear axes
axis([-6 6 -6 6])               % set axes limits
axis square                     % set 1:1 aspect ratio
% Simulated World -- set of all landmarks, red crosses
WG = line(...
   'linestyle','none',...
   'marker','+',...
   'color','r',...
   'xdata',W(1,:),...
   'ydata',W(2,:));
% Simulated robot, red triangle
Rshape0 = .2*[...
   2 -1 -1 2; ...
   0 1 -1 0];                   % a triangle at the origin
Rshape = fromFrame(R, Rshape0);  % a triangle at the robot pose
RG = line(...
   'linestyle','-',...
   'marker','none',...
   'color','r',...
   'xdata',Rshape(1,:),...
   'ydata',Rshape(2,:));
% Estimated robot, blue triangle
rG = line(...
   'linestyle','-',...
   'marker','none',...
   'color','b',...
   'xdata',Rshape(1,:),...
   'ydata',Rshape(2,:));
% Estimated robot ellipse, magenta
```

```
reG = line(...
   'linestyle','-',...
   'marker','none',...
   'color','m',...
   'xdata',[ ],...
   'ydata',[ ]);
% Estimated landmark means, blue crosses
lG = line(...
   'linestyle','none',...
   'marker','+',...
   'color','b',...
   'xdata',[ ],...
   'ydata',[ ]);
% Estimated landmark ellipses, green
leG = zeros(1,N);
for i = 1:numel(leG)
leG(i) = line(...
   'linestyle','-',...
   'marker','none',...
   'color','g',...
   'xdata',[ ],...
   'ydata',[ ]);
end
% II. TEMPORAL LOOP
for t = 1:200
   % II.1 SIMULATOR
   % a. motion
   n = q .* randn(2,1);                % perturbation vector
   R = move(R, U, zeros(2,1) );        % we will perturb the estimator
                                       % instead of the simulator
   % b. observations
   for i = 1:N                         % i: landmark index
      v = s .* randn(2,1);             % measurement noise
      Y(:,i) = observe(R, W(:,i)) + v;
   end
   % II.2 ESTIMATOR
   % a. create dynamic map pointers to be used hereafter
   m = landmarks(landmarks~=0)';    % all pointers to landmarks
   rm = [r , m];                       % all used states: robot and landmarks
                                       % ( also OK is rm = find(mapspace); )
   % b. Prediction -- robot motion
   [x(r), R_r, R_n] = move(x(r), U, n); % Estimator perturbed with n
   P(r,m) = R_r * P(r,m);
   P(m,r) = P(r,m)';
   P(r,r) = R_r * P(r,r) * R_r' + R_n * Q * R_n';
   % c. Landmark correction -- known landmarks
   lids = find( landmarks(1,:) );      % returns all indices of existing landmarks
   for i = lids
      % expectation: Gaussian {e,E}
      l = landmarks(:, i)';            % landmark pointer
      [e, E_r, E_l] = observe(x(r), x(l) ); % this is h(x) in EKF
      rl = [r , l];                    % pointers to robot and lmk.
      E_rl = [E_r , E_l];              % expectation Jacobian
      E = E_rl * P(rl, rl) * E_rl';
      % measurement of landmark i
      Yi = Y(:, i);
      % innovation: Gaussian {z,Z}
```

```
   z = Yi - e;                          % this is z = y - h(x) in EKF
   % we need values around zero for angles:
   if z(2) > pi
      z(2) = z(2) - 2*pi;
   end
   if z(2) < -pi
      z(2) = z(2) + 2*pi;
   end
   Z = S + E;
   %
   if z' * Z^-1 * z < 9
      % Kalman gain
      K = P(rm, rl) * E_rl' * Z^-1;      % this is K = P*H'*Z^-1 in EKF
      % map update (use pointer rm)
      x(rm) = x(rm) + K*z;
      P(rm,rm) = P(rm,rm) - K*Z*K';
   end
end
% d. Landmark Initialization -- one new landmark only at each iteration
lids = find(landmarks(1,:)==0);    % all non-initialized landmarks
if ~isempty(lids)                     % there are still landmarks to initialize
   i = lids(randi(numel(lids)));      % pick one landmark randomly, its index is i
   l = find(mapspace==false, 2);      % pointer of the new landmark in the map
   if ~isempty(l)              % there is still space in the map
      mapspace(l) = true;             % block map space
      landmarks(:,i) = l;             % store landmark pointers
      % measurement
      Yi = Y(:,i);
      % initialization
      [x(l), L_r, L_y] = invObserve(x(r), Yi);
      P(l,rm) = L_r * P(r,rm);
      P(rm,l) = P(l,rm)';
      P(l,l) = L_r * P(r,r) * L_r' + L_y * S * L_y';
   end
end
% II.3 GRAPHICS
% Simulated robot
Rshape = fromFrame(R, Rshape0);
set(RG, 'xdata', Rshape(1,:), 'ydata', Rshape(2,:));
% Estimated robot
Rshape = fromFrame(x(r), Rshape0);
set(rG, 'xdata', Rshape(1,:), 'ydata', Rshape(2,:));
% Estimated robot ellipse
re = x(r(1:2));                       % robot position mean
RE = P(r(1:2),r(1:2));         % robot position covariance
[xx,yy] = cov2elli(re,RE,3,16);        % x- and y- coordinates of contour
set(reG, 'xdata', xx, 'ydata', yy);
% Estimated landmarks
lids = find(landmarks(1,:));           % all indices of mapped landmarks
lx = x(landmarks(1,lids));             % all x-coordinates
ly = x(landmarks(2,lids));             % all y-coordinates
set(lG, 'xdata', lx, 'ydata', ly);
% Estimated landmark ellipses -- one per landmark
for i = lids
   l = landmarks(:,i);
   le = x(l);
   LE = P(l,l);
```

```
      [xx,yy] = cov2elli(le,LE,3,16);
      set(leG(i), 'xdata', xx, 'ydata', yy);
   end
   % force Matlab to draw all graphic objects before next iteration
   drawnow
   % pause(1)
end

function f = cloister(xmin,xmax,ymin,ymax,n)
% CLOISTER - Generates features in a 2D cloister shape.
%   CLOISTER (XMIN,XMAX,YMIN,YMAX,N) generates a 2D cloister in the limits indicated as parameters.
%
%   N is the number of rows and columns; it defaults to N = 9.
if nargin < 5
   n = 9;
end
% Center of cloister
x0 = (xmin+xmax)/2;
y0 = (ymin+ymax)/2;
% Size of cloister
hsize = xmax-xmin;
vsize = ymax-ymin;
tsize = diag([hsize vsize]);
% Integer ordinates of points
outer = (-(n-3)/2 : (n-3)/2);
inner = (-(n-3)/2 : (n-5)/2);
% Outer north coordinates
No = [outer; (n-1)/2*ones(1,numel(outer))];
% Inner north
Ni = [inner ; (n-3)/2*ones(1,numel(inner))];
% East (rotate 90 degrees the North points)
E = [0 -1;1 0] * [No Ni];
% South and West are negatives of N and E respectively.
points = [No Ni E -No -Ni -E];
% Rescale
f = tsize*points/(n-1);
% Move
f(1,:) = f(1,:) + x0;
f(2,:) = f(2,:) + y0;
end
```

Chapter 6

Conclusion

The development of an effective algorithm allows the user to communicate and perform movement operations and collect relevant data for real time control and path planning.

The type of controller developed is fairly simple and very effective and offers great possibility to modify and control the behavior of a robot or any other form of automotive medium such as cars, drones and unmanned-vehicles and etc. While in case of automatic mode or trajectory planning there is greater possibility to improve the algorithm. There are still some problems of working in outside environment where the intensity of light creates problem in data acquisition and reduces the performance of the sensor. For this problem can be solved with the use of more powerful sensors which can work more accurately in harsh environments.

As for the robots ability to do path planning is very good and can be improved upon with better algorithm. As currently the algorithm only allows basic obstacle detection, obstacle avoidance, trajectory planning and environment mapping.

The next step could be the use of a GPS sensor, which could improve tracking more efficiently and use 3D-Lidar sensor to read the environment with better efficiency. We could also incorporate use of visual aid such as camera which can further improve the ability to read and detect obstacles and their avoidance. We could also improve the algorithm for better automatic maneuverability.

Chapter 7

Appendix

Chapter 8

Bibliography

Chapter 9

List of figures and Tables