DESIGN AND IMPLEMENTATION OF AN EKF-BASED SLAM IN A SYNCHRO-DRIVE MOBILE ROBOT USING A LASER SCANNER

by

Maria Marinela Mariano Gutierrez

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Mechatronics

Examination Committee: Prof. Manukid Parnichkun (Chairperson)
Dr. Mongkol Ekpanyapong
Dr. Pisut Koomsap

Nationality: Filipino
Previous Degree: Bachelor of Science in Physics
University of the Philippines Baguio, Philippines

Scholarship Donor: RG Battery & Parts Supply, Philippines

Asian Institute of Technology
School of Engineering and Technology
Thailand
July 2021

AUTHOR'S DECLARATION

I, Maria Marinela Mariano Gutierrez, declare that the research work carried out for this thesis was in accordance with the regulations of the Asian Institute of Technology. The work presented in it are my own and has been generated by me as the result of my own original research, and if external sources were used, such sources have been cited. It is original and has not been submitted to any other institution to obtain another degree or qualification. This is a true copy of the thesis, including final revisions.

Date: 21/07/2021

Name (in printed letters): MARIA MARINELA GUTIERREZ

Signature: 

ACKNOWLEDGMENTS

It is a genuine pleasure to express my gratitude to my research supervisor, Prof. Manukid Parnichkun, for the opportunity he had given me for studying in the Mechatronics course. His dedication and keen interest in helping students not only develops the students' knowledge in the field but also prepares them to the future ahead. I really admire his scholarly advice and meticulous scrutiny that contributed to achieve my task.

I would also like to take this opportunity to thank my research committees, Dr. Mongkol Ekpanyapong and Dr. Pisut Koomsap, for their meaningful assistance and constructive criticisms throughout this research. Their visions and suggestions have enabled me to finish this research.

My research and Mechatronics journey would not be complete without the help from the ISE faculty members, administrative officers – Ms. Chowaret Sudsawaeng and Ms. Saowaluck Maneerat, and technical staffs – Mr. Hoang Hung Manh and Mr. Thanit Pattana. Thank you for their meaningful assistance in making my robot come together. I also thank my fellow labmates in the Advance Robotics Laboratory: Pornchanok Vanich, Piyawat Apiwattanadej, Jirapod Jintasornrom, and Rattapan Pitaksongkram for their support during my study in Thailand.

Another person to thank is my friend, John Isaiah Lejano, for his help and advices when I have problems with my program. Also, I would like to thank, John Benedict Bernardo, for supporting me, making an effort, and motivating me to finish my thesis. I won't be able to finish my research without your assistance and help.

Finally, I want to thank my family and relatives for their unending support both financially and emotionally. Being apart from them is one of the main struggles for me while doing my research. Without their help and support, I won't be able to achieve this far.

To all those people involved in my research and my study, Maraming Salamat po!

ABSTRACT

Indoor mobile robots are robotic systems that have a certain level of autonomy. These vehicles have been studied thorough out the years for applications in the mapping and localization. Simultaneous localization and mapping (SLAM) is one of the underlying problems that concurrently estimates the map while identifying the pose estimate of the robot. This study proposed to design a synchronous drive mobile robot with independent steering and driving mechanisms guided by a chain/belt transmission Furthermore, the implementation of EKF-SLAM is addressed in this study using the information from the odometry and laser scanner. The EKF filter has two steps, the 1) prediction step and the 2) correction step. The predicted measurement is used to initially determine the state of the robot of the robot and the correction step uses the actual measurement for comparison with the prediction. In detail, the landmarks are extracted in the environment using point clustering. Clustered points less than 15 points reject these clusters which aren't included in the observed landmarks. Also, a middleware called Robotic Operating System (ROS) is used to communicate the robot's microcontroller to the computer and employ packages related to SLAM. The EKF SLAM method is evaluated through the calculation of the root mean square error between the predicted measurement (calculated EKF-SLAM data) and the observed measurement (actual measurement in the environment). It was found out that drifting of the chains contributed to the increase in the root mean square error of 0.2981 m (x-axis) and 0.1589 m (y-axis). Also, the mean square error from the overall theoretical distance of 4.48 m gives the error of 2.70%.

CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Page

LIST OF ABBREVIATIONS

DR = Dead Reckoning
EKF = Extended Kalman Filter
et = encoding type
GNSS = Global Navigation Satellite System
GPS = Global Positioning System
GR = Gear Ratio
IF = Information Filtering
IMU = Inertial Measurement Unit
KF = Kalman Filter
MLP = Multilayer Perceptron
MMSE = Minimum Mean-Square Error
OS = Operating System
PF = Particle Filter
ppr = pulse per revolution
pv = process variable
PWM = Pulse Width Modulation
RBF = Radial Basis Function
RBPF = Rao-Blackwellized Particle Filter
ROS = Robotic Operating System
SLAM = Simultaneous Localization and Mapping
SP = Setpoint
T = Teeth
UGV = Unmanned Grounded Vehicle
UKF = Unscented Kalman Filter
WMR = Wheeled Mobile Robots

CHAPTER 1
INTRODUCTION

1.1 Background of the Study

Wheeled Mobile Robots (WMR) are mobile systems have a certain level of autonomy. It is one of the most expanded and invested research because of its numerous applications in the field of surveillance, medical care, rescue operations, and so on (Bruzzone et al., 2021; Takahashi et al., 2010). These versatile systems move in different physical environment and operate more efficiently with the inclusion of position sensors, tilt sensors, and laser scanners for applications such as localization, mapping, and navigation (Chen et al., 2021).

The most common type of indoor WMR design is the differential drive wheeled robot which composed of independent driven motors Being a configuration where its wheels directly connected to the motor, synchronization and deviation from the motion of the robot exists (Chung & Iagnemma, 2016). As a result, inaccuracies and inconsistencies with the movement persists using the motor driven wheel approach. One overlooked drive configuration is the synchronous-drive mobile robot which uses a chain/belt transmission and two independent motors to move the system in a certain direction. The introduction of separate steering and driving mechanism allows a simple and stable linear control (Marin-Reyes & Tokhi, 2010). This design also guarantees straight-line motion from dynamically controlling the separated motors.

As synchronous-drive robots are presently operated in indoor environments, Simultaneous Localization and Mapping is one of the well-known problems to discuss. Indoor mobile robots provided the foundation of the SLAM problem because of the line-of-sight problem introduced by the Global Position Devices (GPS) (Chan et al., 2021). Simultaneous localization and mapping is one of the underlying problems that concurrently estimates the map while identifying the pose estimate of the robot. It is like a chicken-or-egg problem that requires a map to be able to localize and a pose estimate for generate the map. Among the approaches developed in SLAM, a state estimator called Extended Kalman Filter (EKF) is introduced. EKF is discussed the seminal paper of Smith and Cheeseman in 1986 focusing on the focuses on the estimation pose and uncertainty associated among the objects (Smith & Cheeseman,

1986). One main advantage of EKF is its significance in solving non-linear models which provides non-linear state update or measurement equations.

1.2 Statement of the Problem

With the emergence of different drive configurations in mobile robots common to academic researches, education, and industry, less attention is given to the synchronous drive design. The differential drive, having an uncomplex design and easily programmable, faces problems in terms of the control due to inconsistencies and motor asynchronization. Divergence to the resulting path is possible and yields to inaccurate results (Chung & Iagnemma, 2016). The car-type (Ackermann Steering) configuration, on the other hand, performs best for straight line motion. However, non-holonomic planning is required and has a minimum turning radius (Agrawal et al., 2021). Additionally, the car-type design has a complicated steering structure making the parking control motion difficult. Alternative drive configuration system should be considered that eliminates the problem of the mechanical control, the synchronous drive mobile robot. The independent steering and driving make the synchro-drive robot control easier and its straight-line motion is mechanically guaranteed.

Another problem to consider is the errors present in indoor localization. Global positioning systems (GPS) doesn't work well in obstructed places from walls without a clear view to the satellite (Maliha Monsur, 2021). To deal with this dilemma, a technique called SLAM must be considered. The information from the environment is processed and utilized to determine the mobile robot's location in the environment.

This research considers a drive system called synchronous mobile robot. A system requires two motors, 1 for steering and 1 for driving. As a result, this guarantees a straight-line motion of the system compared to the differential drive type. The application of a synchronous-drive in mobile robot for SLAM has potential and will be further investigated in this thesis.

1.3 Objectives of the Study

The primary focus of this thesis is to develop a synchronous drive mobile robot and implement the Extended Kalman Filter-SLAM (EKF-SLAM) into the proposed system. In order to achieve this goal, the following tasks are included.

1. To construct a three-wheeled synchronous drive mobile robot with a hexagonal-shaped platform, and
2. To perform EKF-SLAM and determine the estimated state and map of the proposed system using a laser scanner.

1.4 Scope and Limitation

This thesis mainly focuses on designing a three standard wheeled synchronous drive mobile robot with special emphasis in an indoor planar environment and applying the proposed mechanism to explore the problem space. It should be noted that the area is not known and is limited to the presence of static obstacles. Additionally, the robot chassis' geometry will be a regular hexagon with 6 lines of symmetry instead of the conventional three or four-sided design. The landmark detection is also included in the EKF-SLAM operation using point clustering method. Other specifications of the mobile robot are stated as follows.

1. The maximum driving speed of the mobile robot is 0.16 m/s.
2. The maximum elevation of the flat surface has an angle of 5.0 degrees
3. Three uniform standard wheels is incorporated in the robotic system
4. The synchro-drive design always rotates about the center of the robot and being an omnidirectional system, the heading can't be changed when the wheel legs are being steered.
5. The total run time of the robot is 10 minutes.
6. The considered landmarks are limited to shapes such as rectangle, circle, and square. Moreover, colored landmarks can't be detected by the system.
7. The landmark detection rejects the cluster of points of less than 15 points.

An extension of this research is the addition of the obstacle avoidance to illustrate the application of the proposed robot. The experiment uses two obstacles that are placed in the robot environment and is tasked to arrive at the desired location.

It is emphasized that the main contribution of this research is the design of the synchro-drive mobile robot together with the wheel structure design. The EKF-SLAM is then applied to the system using the information from external sensors, laser scanner and encoders, which uses a configured landmark detection.

1.5 Organization of the Study

This thesis is organized as follows:

- Chapter 2 provides an overview of the wheel mobile robot drive configurations, its recent development and applications, the theoretical concepts of robot localization and mapping, filtering techniques such as Kalman Filter and its variants, and the integration of ROS and SLAM.
- Chapter 3 focuses on the hardware and software implementation on the synchronous-drive robot and discusses the process of implementing Extended Kalman Filter with ROS.
- Chapter 4 presents the results between the actual measurement and the calculated measurement from the EKF-SLAM which includes the discussion of the landmark extraction results.
- Chapter 5 summarizes and concludes the findings from the research. This section also recommends any improvement discovered and observed from the results.

CHAPTER 2
LITERATURE REVIEW

This chapter discusses several works related to the design of wheeled mobile robot (WMR) and its applications such as the synchronous drive robot, the concurrent localization and mapping, and filtering techniques like the Kalman filter (KF) and the approximation of nonlinear functions, the Extended Kalman Filter.

2.1 Wheeled Mobile Robot

Before diverging into the navigational process, the mechanical design of the mobile robot must be defined. The structure of the robotic system needs to be in agreement with the unique constraints from a particular mobile robot application. For instance, the delivery robots have a main objective of transferring products from position A to position B. The design of the robotic system must include a mechanism that allows the robot to climb certain elevation. If not, the goods will not arrive at the destination point. Mobile robots have different configuration depending on the application. Stated below are several drive configurations that are applicable for implementation.

2.1.1 Drive Configuration

Three fundamental issues are considered in designing the mobile robotic system; mobility, control, and balance. Conventionally, three wheels are necessary for satability stability and incorporating suspension design to the wheels resolve the problem on uneven surfaces (Siegwart et al., 2011). In addition, a trade-off between maneuverability and controllability exists. Advantages and disadvantages of several drive configurations are described and stated as follows.

One of the well-known drive mechanisms is the differential drive. Motors are independently driven to produce the robot's trajectory. Additionally, this mechanism can produce a zero turning radius (Chung & Iagnemma, 2016). The problem emerges when the driving motors move at different speeds or not synchronized even with the same applied voltage. As a result, the route of the robotic system will not achieve the desired movement which also applies for an inconsistent terrain (Jones et al., 1998).

Another type is the Ackerman steering (car-type drive) which is usually a common mechanism in automobiles. This configuration consists of two front steered wheels and two driven rear wheels and has an advantage of straight driving motion. However, the mechanism doesn't have the capacity of turning on the spot which requires a certain minimum radius. Moreover, the rear driving wheels experience slippage in curves (Bräunl, 2008).

Last of the configuration types is the synchronous drive mechanism that controls the two independent motors either simultaneously driving or steering the wheels. This gives less control effort and favors the indoor environment. However, the arrangement suffers from an orientation error when backlash or loose coupling takes place (Siegwart et al., 2011).

2.1.2 Synchronous-drive Mobile Robot and its Applications

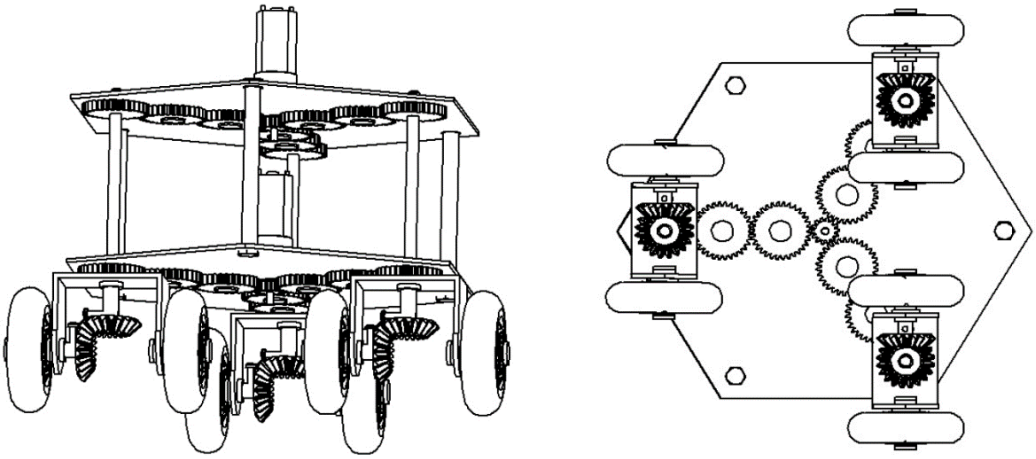
In the endeavor of minimizing the constraints in the design for a certain mobile robot application, several drive configurations are available for implementation. Among those configurations examined from the previous section, the synchro-drive principle was used in this study for the mechanical design of the mobile robot. This new drive system's body maintains a constant orientation which allows the sensor facing towards the direction of travel (Miller, 1986). Short discussions of several synchro-drive designs are reported as follows.

A geared mechanism was introduced by Tatar et al. which uses a three-paired standard wheel design with a hexagonal layered platform. Figure 2.1 illustrates the mechanical design of the omni-directional robot. The upper platform's gears allow the forward or backward driving of the wheels while the lower part is intended for wheel steering (Tătar et al., 2015).

An alternate design was proposed by Goris in his thesis about the mobile robot mechanical design (Goris, 2005). He used the chain transmission mechanism to transfer the motor's rotary motion to the wheels and pointed out that using gears in the mechanical platform takes a lot of space and is heavy. Moreover, the chain mechanism was implemented because it solves the problem of slippage which was evident in belts.

Figure 2.1

Synchro-drive Design using Geared Transmission

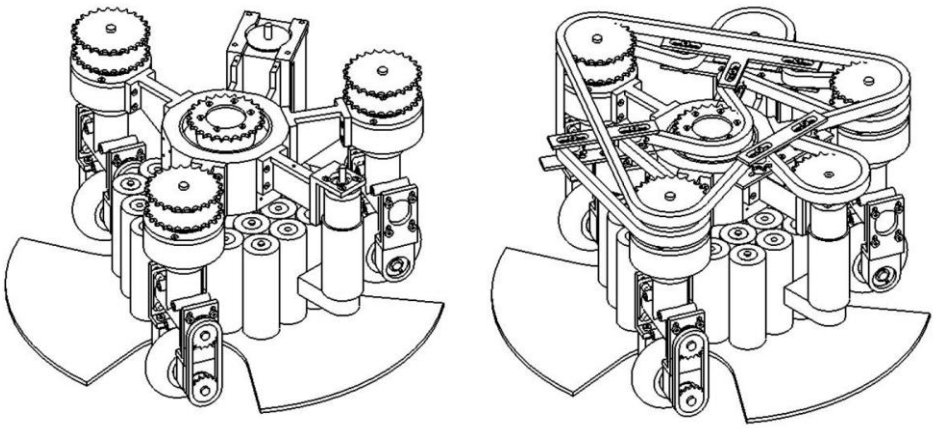


Note. This image was adapted from (Tătar et al., 2015).

Figure 2.2 shows the platform design and the transmission mechanism of an autonomous mobile robot. Similar to the design of Tatar et al (2015), the design uses two DC motors that controls the driving and orientation of the wheels. The design is composed of an upper belt transmission for wheel driving while the lower belt transmission is intended for wheel steering.

Figure 2.2

Mechanical Design of an Autonomous Mobile Robot using Belt Transmission

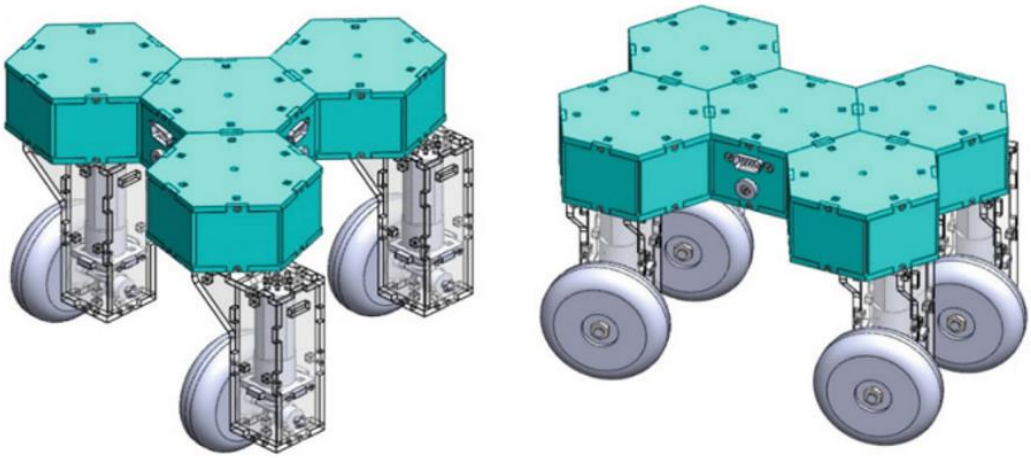


Note. This image was adapted from (Goris, 2005).

Tatar and Cirebea developed a reconfigurable wheeled mobile robot using a combination of multiple active and passive hexagonal modules together with the parallelepipedic module (Tătar & Cirebea, 2018). These active and passive modules are interconnected with the passive controller module at the center to reconfigure the robotic system. In addition, both the active and passive module incorporates a reducer and an encoder in the design. The model of the hexagonal module is illustrated in Figure 2.3.

Figure 2.3

Three-wheeled (Left) and Four-wheeled Hexagonal Modular Robot (Right)

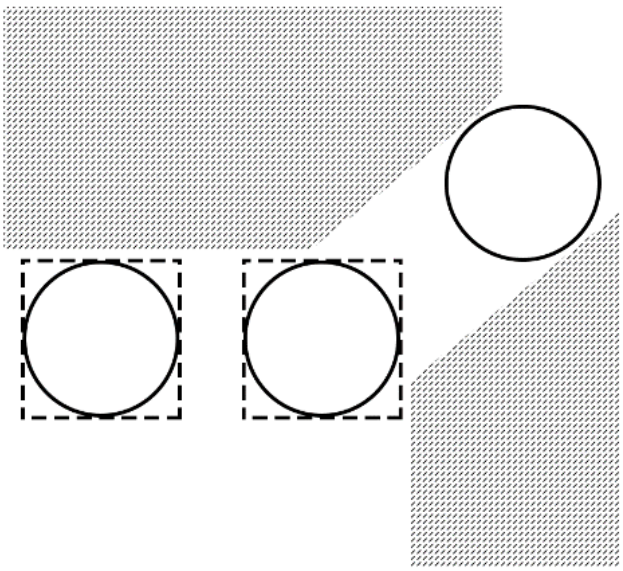


Note. This image was adapted from (Tătar & Cirebea, 2018).

According to Goris (2005), the cylindrical-shaped robot has a learning edge over the square-shaped platform because of the former's structure when subjected in confined spaces. The square platform will most likely be trapped in a narrow passage. Figure 2.4 further illustrates the concept between the cylindrical and square shape. Once trapped, the square geometry will have to move backwards and rotate before entering the confined space. The cylindrical shaped on the other hand doesn't need to do this process.

Figure 2.4

Obstacle Representation of a Cylindrical and Square Robot Shape



Note. This image was adapted from (Goris, 2005).

2.2 Localization

A mobile robot that estimates its current location with respect to a fixed reference frame (also known as localization) can localize itself through the use of sensorial observation from proprioceptive and exteroceptive sensors. While GPS is widely used in outdoor environments, these radionavigation system show a limitation in indoor or with solid structures. Examples of proprioceptive sensors include wheel encoder and inertial measurement units (IMUs) that acquires motion information from the robot. On the contrary, exteroceptive sensors consist of a laser range scanner or camera that obtains measurement from the external environment. Incorporating only the proprioceptive sensors produces errors from odometry uncertainties due to wheel slippage, drifting of wheels, or uneven floor surface (Borenstein & Feng, 1996). With the inclusion of sensors such as the laser scanners, the robot acquires additional and useful information concerning its environment.

Researchers have provided techniques to adhere to the problem of mobile robot localization which are organized into two main group, relative localization and absolute localization. The discussion of each type is further illustrated in the following sections.

2.2.1 Relative (Local), Absolute (Global), and Combined Localization

In terms of its position information, relative localization uses environmental information such as images or edges of landmarks to distinguish whether the robot has traveled or not (Kim, 2019). Dead reckoning (DR) utilizes the wheel encoder data thru wheel rotation count to estimate the position of the robot with respect to its starting position. For longer distances, errors using this process will accumulate over time.

Absolute localization, on the contrary, utilizes the fixed frame of the Earth to provide vehicle location from measurement sensors such as the Global Navigation Satellite System (GNSS), wheel encoders, and IMU. Compared to the accumulative error from dead reckoning, the error growth in the absolute localization is reduced. The reason about the mitigated error came from the time and location independency of the robot's position. However, absolute localization has disadvantages over small distances when tracking the robot using a GPS (Goel et al., n.d.).

2.2.2 The Simultaneous Localization and Mapping Problem

The introduction of SLAM dates back to the seminal paper of Smith and Cheeseman in 1986 which focuses on the estimation method (position and orientation) and uncertainty associated among the objects (Smith & Cheeseman, 1986).

Well-known mapping algorithms have one thing in common in literature, it has a probabilistic nature. This approach is prominent to robot mapping because it models noise sources and analyze the influence to the measurements (Thrun, 2003). Being a posterior probability problem, Simultaneous Localization and Mapping or SLAM is a joint estimation method that determines the pose estimate of the mobile robot while incrementally constructing the map of its environment (Durrant-Whyte & Bailey, 2006).

The SLAM technique favors indoor applications which solves the problem raised by the inaccuracies of GPS in indoor environments (Cadena et al., 2016). Moreover, the nature of the SLAM problem yields to an improved result as the number of observations is increased and its correspondence to other attributes rises. However, the increase in landmark count is quadratically scaled and results as a drawback. This restricts the real-time application to small and medium scaled domain (Durrant-Whyte & Bailey, 2006).

The SLAM problem can be classified into two categories; online SLAM and full SLAM. For the case of online SLAM, the posterior distribution is estimated with the robot pose, x_t, and the model of the map, M, which yields to

$$p(x_t, M|Z_T, U_T) \tag{2.1}$$

where U_T represents the odometry and Z_T represents the observations (Siegwart et al., 2011). This approach estimates only the current pose.

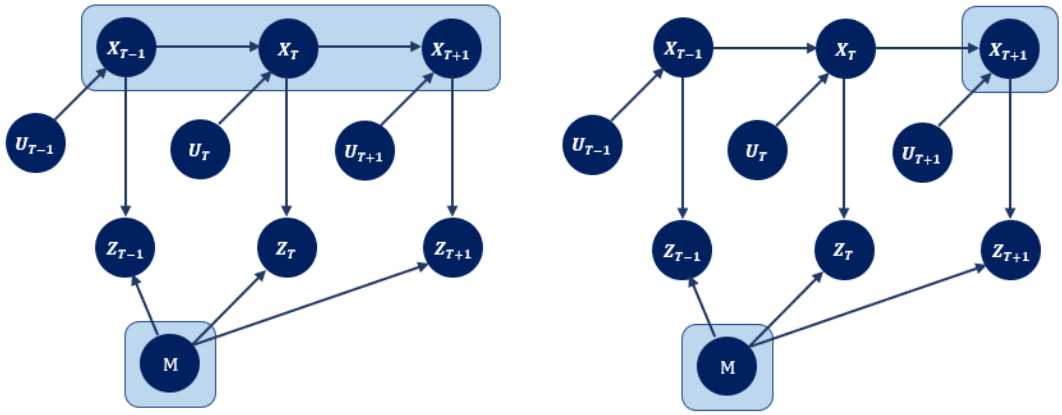
The full SLAM problem, on the contrary, is approximated with the entire robot path, X_T, and M, results to

$$p(X_T, M|Z_T, U_T) \tag{2.2}$$

The graphical model of the aforementioned SLAM problem is illustrated in Figure 2.5.

Figure 2.5

Graphical Representation of Full SLAM and Online SLAM



Online SLAM (a) **Full SLAM** (b)

Note. These images are adapted from (Chung & Iagnemma, 2016) .

2.3 Filtering-based Techniques in SLAM

Three major classifications of the SLAM paradigm is introduced in this section which consists of the Kalman Filter, Particle-based techniques, and the Graph-based implementations. The discussion of the mentioned techniques is examined below.

2.3.1 Particle Filter, Graph-based, and Kalman Filter Approach

The particle filters (PFs) apply a sample set of particles that approximates the state's actual configuration. Particle-based SLAM or the sequential Monte-Carlo method is not limited to linear models or Gaussian noise which is a great advantage for nonlinear models compared to the Kalman Filter. However, the negative aspect of this technique is the exponential growth of particles as the space dimension increases resulting to a higher computational requirement (Chung & Iagnemma, 2016). Furthermore, this type of filter displays an inaccurate result because of its inability to omit the past data. One of the examples under this filter is the Rao-Blackwellized particle filter (RBPF) which stems back to the paper by Murphy in 2000 (Doucet et al., 2013; Murphy, 1999).

In 1997, Lu and Milios formulated the graph-based technique in the SLAM problem which involves the graph generation where nodes constitute the state (robot pose or landmarks) and the edge in between two nodes encodes a sensor measurement (Grisetti et al., 2010; Lu & Milios, 1997). One of the advantages of the graph-based SLAM is its ability to constantly update time of the graph and the required memory linearity in the feature. Although, the graph optimization can become computationally expensive if the robot path is long.

A simulation was conducted by Nguyen to analyze the performance of the Bayesian filtering techniques such as Extended Kalman Filter, Unscented Kalman Filter (UKF), and FastSLAM over an identical set of parameters; 150 landmarks, 3° steering control noise, 0.1 m range, and a bearing of 1°. Also, the velocity of the mobile robot was assumed to 3 m/s and the maximum sensor range is 30 m. Results show that EKF technique showed good performance amongst others (H. K. Nguyen, 2014). Additionally, Table 2.1 further demonstrates the edge of one algorithm to the other.

A notable filtering approach in the SLAM problem is the Kalman Filter (KF) which was presented by Rudolf E. Kalman in 1960 (Kalman, 1960). This filter utilizes linear transition functions in a state described by a Gaussian distribution (Montella, 2014). Well-known examples under KF are the Extended Kalman Filter, Unscented Kalman Filter, Information Filtering (IF) or the Extended IF. Advantages of using KF and its variants is the ideal minimum mean-square error (MMSE) approximation of the robot

Table 2.1

Comparison Table for Different Filter-based Techniques in SLAM

	EKFSLAM	**FastSLAM**	**GraphSLAM**
Complexity	$O(n^2)$	$O(k * logn)$	$O(e)$
Distribution	Gaussian	Any	Gaussian + outlier rejection
Linearization	Once	Not needed	Re-linearize
Flexibility	0	+	++
Large Scale	-	+	++
Parallelizability	-	+	++
Pros	Easy to implement well known	Can use negative information	Scales well Robust
Con	Can't handle large maps	Hard to recover, need many particles to be robust	Harder to implement

Note. This table illustrates the comparison of different SLAM techniques such as EKFSLAM, FastSLAM, and GraphSLAM. Variables in the complexity row (n, k, and e) represent the number of landmarks, number of particles, and number of edges, respectively. This table is adapted from (Lindholm & Palsson, 2015).

state and landmark positions. Also, the covariance matrix has a strong convergence (Aulinas et al., 2008).

One solution isn't always guaranteed for solving the concurrent mapping and localization problem because of certain dependencies between specific criteria or variables such as feature count in the environment, computation time, and map resolution. Hence, an optimal solution should be preferred when dealing with the SLAM problem.

2.4 Kalman Filter

Kalman Filter (KF) is a filtering method that approximates the state of the system using Gaussian distribution. This filter has been applied to numerous state estimation process with the addition of noise. In autonomous vehicles, KF is used for predicting succeeding states of the system without requiring any data history.

2.4.1 Mathematical Model of Kalman Filter

The process of Kalman Filter can be divided into two steps, the a) prediction and the b) update step. In the prediction step, the next position of the system at time interval t+1 is based from the previous position and the system's kinematic model. This step also predicts the covariance error. On the contrary, the update step compares the predicted state and the actual measurement from the sensors involved. It also includes a Kalman

gain that varies depending on where it will be biased to. The prediction step equation is given as follows.

$$x' = Fx + B\mu + v \tag{2.3}$$

$$P' = GP'G^T + Q \tag{2.4}$$

where x' is the predicted value or the a priori estimate, P' is the predicted error covariance or uncertainty measure in the estimated state, *F* is the state transition matrix, *B* is the control input matrix, *v* is the process noise present in the system, *G* is the Jacobian matrix, and *Q* is the process noise/motion noise (Singh, 2018). The update step of the KF process yields to the difference of the actual measurement and predicted measurement value given by the Equation (2.5).

$$y = z - H\,x' \tag{2.5}$$

where *z* is the actual measurement and *H* is the state transition matrix. Another step in the KF process is the summation of error from the measurement error and prediction error given by S which later yields to the Kalman gain, K.

$$S = HP'H^T + R \tag{2.6}$$

$$K = P'H^TS^{-1} \tag{2.7}$$

The Kalman gain serves as a bias and heavily decides whether the measurement or the predicted value mainly contributes to the state vector. The behavior of the Kalman gain is observed in Equations (2.6) – (2.8) from variables *R* and P'. When the value of R approaches 0, the gain yields to H^{-1} which shows that the measurement value mostly influences the state vector. As P' goes to a smaller value, Equations (2.6) and (2.7) become 0 and the resulting value in Equation (2.8) is mainly influenced by the a priori estimate. The update equation of KF is illustrated in Equations (2.8) and (2.9).

$$x = x' + Ky \tag{2.8}$$

$$P = (I - KH)P' \tag{2.9}$$

2.4.2 Extended Kalman Filter

As Kalman Filters are defined and addressed for estimating linear system states, a non-linear state estimator should be used to accommodate non-linear systems. Extended Kalman Filter is a filtering process that linearizes nonlinear models about the mean of the current estimate. The basic idea of EKF is that non-linear functions are linear approximated through the help of the first derivative from the Taylor Series Expansion (See Equation 2.10).

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots \quad (2.10)$$

After the linearization process and a linear model is obtained, the Kalman Filter equations mentioned in Section 2.4.1 are applied (Haykin, 2004).

Drawbacks also exist in the EKF filtering process which includes the computational complexity due to the calculation of Jacobian especially requiring numerical differentiation, differentiable model limitation, and highly non-linear systems non-optimality. Other state estimators including Kalman Filter and Extended Kalman Filter are tabulated in Table 2.2.

Given with its advantages and drawbacks, the Extended Kalman Filter is used as a state estimator for the SLAM problem introduced in this research. Current studies related to the EKF-SLAM is described and explained as follows.

Table 2.2

Comparison Table for Different State Estimator Filters

State Estimator	Model	Assumed Distribution	Computational Cost
Kalman Filter (KF)	Linear	Gaussian	Low
Extended Kalman Filter (EKF)	Locally Linear	Gaussian	Low (if Jacobians need to be computed analytically) Medium (if the Jacobians can be computed numerically)
Unscented Kalman Filter (UKF)	Nonlinear	Gaussian	Medium
Particle Filter (PF)	Nonlinear	Non-Gaussian	High

Note. This information was retrieved from https://www.youtube.com/watch?v=Vefia3JMeHE&list=PLn8PRpmsu08pzi6EMiYnR-076Mh-q3tWr&index=5

The actual application of the SLAM involves the utilization of sensors to describe the robot's environment (Chatterjee et al., 2011). Several procedures were implemented under the EKF method using laser beam to detect the object with respect to the reference point. Lasers were used because of its performance and accuracy in detecting the environment (Lei & Li, 2012; Lindholm & Palsson, 2015; Lv et al., 2015). In addition, it is robust in varying of lighting and temperature conditions (Lv et al., 2014).

Nguyen and his colleagues applied the orthogonality principle to define the indoor environment such office rooms or laboratories in the line feature approach in SLAM (V. Nguyen et al., 2006). The linear estimation problem was minimized by applying the Kalman filter and the Relative Map technique. The proposed technique focuses on developing an efficient and lightweight approach in real-time applications. It was verified that the accuracy of the map is comparable to the ground truth map. Another research investigated on a geometrically constrained EKF framework proposed by Choi to accurately estimate the line feature positions. The researchers emphasized that the general indoor setting is orthogonal or parallel to each other so that line features is used. However, laser scanners aren't used in this research because of its objectives to develop a low-cost cleaning mobile robot application. More, the robot has attached seven IR sensors to the differential drive configuration (Choi et al., 2008).

Lv and his associates used the concept of straight-line segments using two laser range finders to detect features in the environment. Moreover, line segments for feature association are more complicated compared to point feature because of the additional consideration of line segment endpoints. The researchers also used the EKF approach to correct the predicted state with the observation and former feature relationship (Lv et al., 2014).

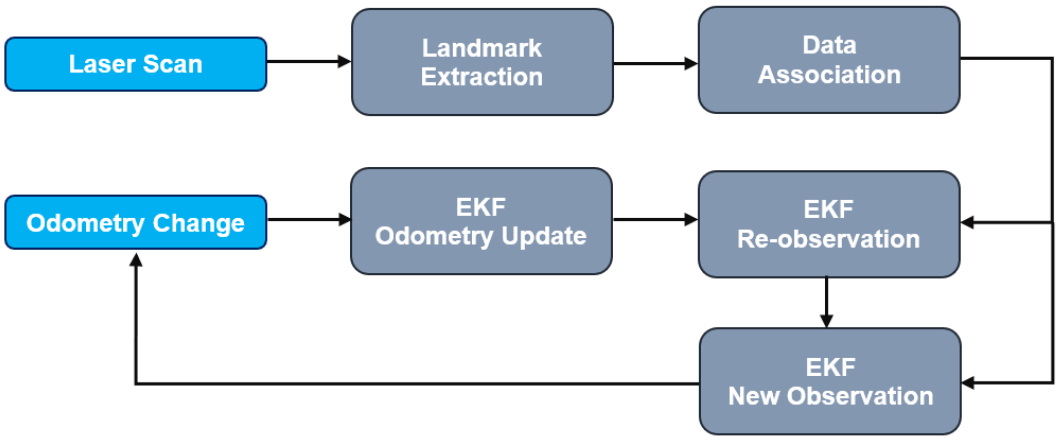
Genevois combined a laser scanner and an odometry system to perceive the environment and estimate the position and heading of the EKF based SLAM. A downfall for this technique is slower when general exploration takes place and limited landmarks are used because of the point estimation used in the landmarks (Thomas Genevois & Zielinska, 2014).

Saputra simulated the EKF-SLAM to determine the state of the system with the comparison of the dead reckoning operation and the GPS system and implemented the method into a real Unmanned Grounded Vehicle (UGV) platform. The author found out that landmark filtering methods are necessary to produce better results of the robot's estimated state (Saputra, 2015).

The main concept of the EKF-SLAM utilizes and extracts the environment's features, commonly termed as landmarks, to update its position of the robot in the environment. Extended Kalman Filter allows keeping track of the uncertainty estimate of the robot's position and landmark uncertainty. An overview of the EKF-SLAM process is illustrated in Figure 2.6.

Figure 2.6

EKF-SLAM Process from Laser Scanner and Mobile Robot Odometry



Note. This image was adapted from the documentation entitled (Riisgaard & Blas, 2003).

The EKF-SLAM starts by gathering information from the environment and extracts the necessary data for landmark association. Then from laser scanner data, the observed landmarks are either new or re-observed. An equivalent signal is sent to the robot for state update.

2.5 Robotic Operating System (ROS)

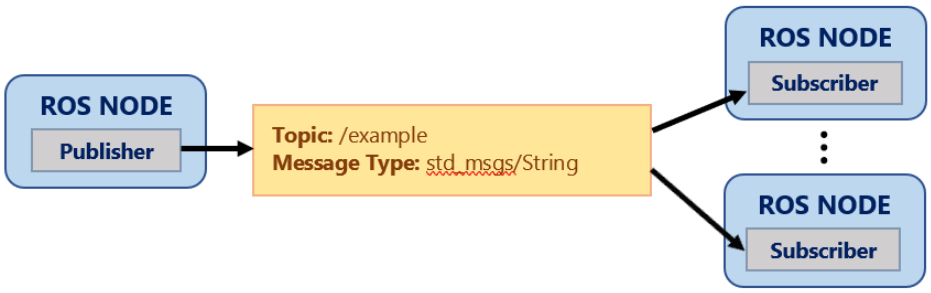
The robotic operating system (ROS) serves as a framework consisting of tools, libraries, or packages essential in writing robot software. This open-source middleware provides flexibility to developers and potentially improve the system introduced by the community.

The concept of ROS revolves around the exchange of data between a publisher node and subscriber node through a message containing a unique topic name and type. The

publisher node broadcasts the information into a topic while the subscriber receives that data from the publisher with the same topic. Many-to-many communication is employed between the subscriber and publisher for a given topic. Thus, the publishing node is allowed to send multiple messages at the same topic while the subscriber is authorized to receive multiple data from a topic. One unique characteristic for a message is that it can be published even without an active subscriber. An illustration of a publisher-to-subscriber communication is illustrated in Figure 2.7.

Figure 2.7

Concept Map of node-to-node communication



Note. This image was adapted from https://www.mathworks.com/help/ros/ug/exchange-data-with-ros-publishers-and-subscribers.html.

ROS works on the operating systems (OS) of Ubuntu Linux and experimentally supports OS X, Gentoo Linux and Windows. However, it is recommended to use the Ubuntu Linux as the operating system since it well documented and developed compared to the other OS. The programming languages used in ROS is either C++ or Python and can be used interchangeably when running nodes. An important part to take note of while working with ROS is the type of message to deliver the information to the other node.

2.5.1 Robotic Operating System and SLAM

The use of SLAM in robotics is well established in ROS since the ROS Electric Emys distribution release in August 30, 2011 including the packages *hector_slam* and *gmapping.* The *hector_slam* package uses an approach not requiring an odometry nor roll/pitch motion from the robotic system. It provides two-dimensional pose estimates using LIDAR systems such as Hokuyo UTM-30LX. However, the closed-loop ability was not provided in the method but provides sufficiently estimate for real-world applications (Kohlbrecher & Meyer, 2020). A sample application using *hector_slam* by handholding the device is illustrated in Figure 2.8.

Figure 2.8

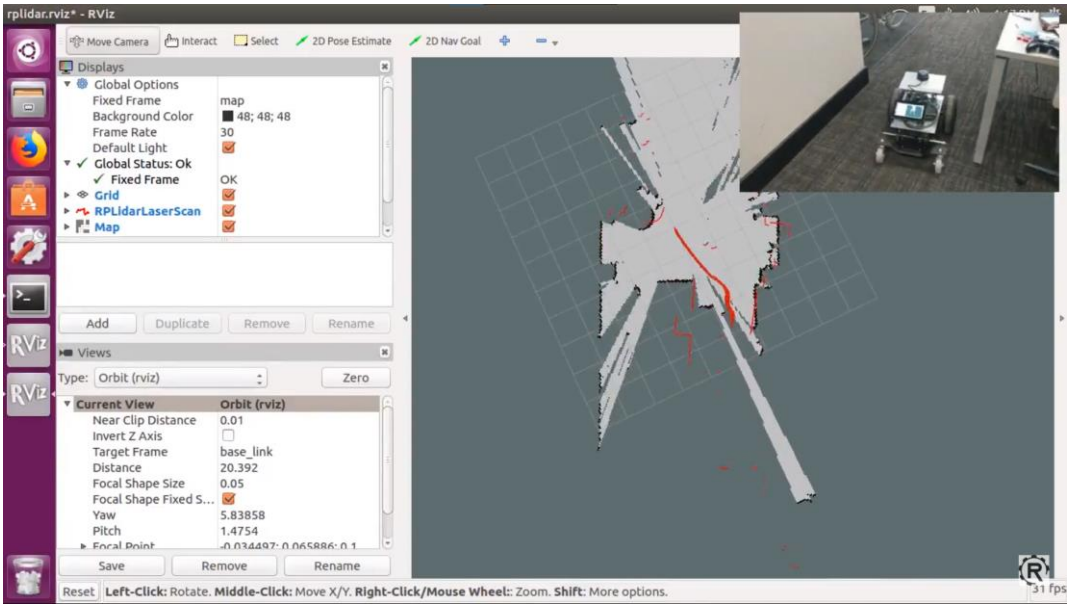
Example of Handheld Mapping System using Hector_Slam Package in ROS

Note. These imaged was acquired from YouTube https://www.youtube.com/watch?v=Cfq3s4-H2S4&t=43s

The *gmapping* package, on the contrary, produces a two-dimensional occupancy grip map from the robot's odometry data and laser range-finder data. Gmapping uses Rao-Blackwellized particle filter for grid mapping on to the laser telemeter data. Each particle in this filter carries information of the environment (Gerkey, 2020). Sample image of a gmapping application is illustrated in Figure 2.9.

Figure 2.9

Gmapping Application Using a Two-wheel Drive Rover and RPLiDAR A3 in ROS



Note. These imaged was acquired from YouTube https://www.youtube.com/watch?v=Cfq3s4-H2S4&t=43s

CHAPTER 3
METHODOLOGY

3.1 Introduction

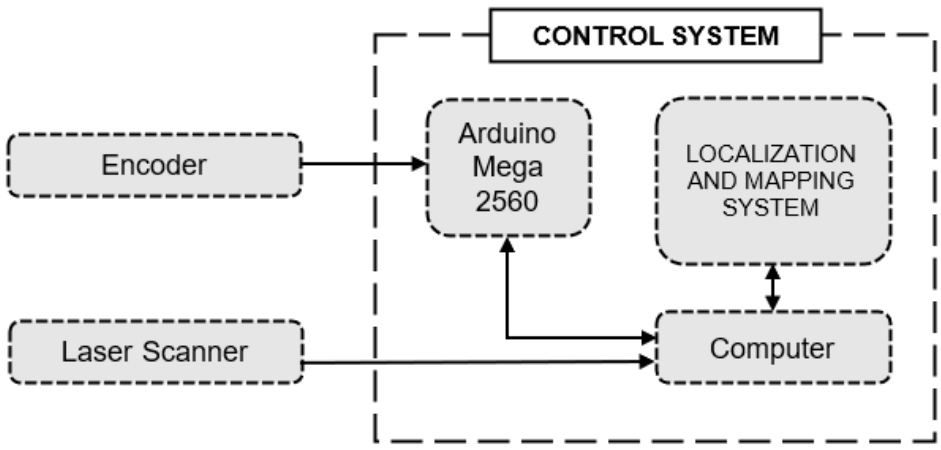
This chapter expounds the design of the wheeled mobile robot including the outline of the overall control system structure together with mapping and localization, the robot's kinematic model and mechanical platform, hardware serial connections, calculations for the wheel distance and speed, laser scanner output raw data conversion, procedure of the EKF-SLAM, and ROS implementation with the mobile robot.

3.1.1 Overall System Structure

To implement the localization and mapping in the mobile robotic system, a middleware called ROS is installed and applied with its dependent packages. The computer then processes the raw output data from the laser scanner and feeds the processed data into the control system. Moreover, the computer sends a message to the microcontroller which delivers an equivalent control to the motors based from the data from the external sensors such as encoders and laser scanner. Figure 3.1 shows the flow diagram of the control system together with its external sensors.

Figure 3.1

Main System Structure



3.2 Synchronous-drive Mobile Robot Design

The design of the synchro-drive robot is actuated using two independent motors, driving and steering motors, which are mechanically coupled by gears and chains. The distinction of the design compared to other drive configuration is its inability to change heading. Despite this impediment, the synchronous-drive has an advantage of minimal control effort with simultaneous turning or steering of the wheels.

3.2.1 Kinematic Model for a Synchronous-drive Robot

The kinematic model of a mobile robot is used to predict the motion of the robot based from the previous state and its control input without considering the forces affecting this motion. A synchro-drive configuration has similarities with the kinematic model of the unicycle model since the synchro-drive's wheels are actuated in unison either driving or steering. The discrete version of the kinematic model where the robot state $(X_{(k)})$, consisting of the x-position, y-position, and robot heading, and control input $(u_{(k)})$, containing the speed $(v_{(k)})$ is given by Equation (3.1).

$$X_{(k+1)} = f\left(X_{(k)}, u_{(k)}\right) \tag{3.1}$$

where the index k represents the k[th] sample. The robot state and control input vectors are given by

$$X_{(k)} = \begin{bmatrix} x_{(k)} \\ y_{(k)} \\ \theta_{(k)} \end{bmatrix} \tag{3.2}$$

$$u_{(k)} = \begin{bmatrix} v_{(k)} \end{bmatrix} \tag{3.3}$$

Based from the control input, the robot state is predicted using the discrete time kinematic model written in Equation (3.4)

$$X_{(k+1)} = \begin{bmatrix} x_{(k+1)} \\ y_{(k+1)} \\ \theta_{(k+1)} \end{bmatrix} = \begin{bmatrix} x_{(k)} \\ y_{(k)} \\ \theta_{(k)} \end{bmatrix} + \begin{bmatrix} \Delta t v_k \cos\left(\theta_k + \Delta t \varphi_k / 2\right) \\ \Delta t v_k \sin(\theta_k + \Delta t \varphi_k / 2) \\ \Delta t \varphi_k \end{bmatrix} \tag{3.4}$$

where the first term represents the robot state at k interval and the second term represents the velocity based kinematic model.

Solving for the Jacobian of the robot kinematic model with respect to the robot state and control input yields to

$$G = \frac{\partial}{\partial (x, y, \theta)^T} \left[\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + \begin{bmatrix} \Delta t v_k \cos\left(\theta_k + \Delta t \varphi_k / 2\right) \\ \Delta t v_k \sin(\theta_k + \Delta t \varphi_k / 2) \\ \Delta t \varphi_k \end{bmatrix} \right] \tag{3.5}$$

$$G = \begin{bmatrix} 1 & 0 & -\Delta t v_k \sin(\theta_k + \Delta t \varphi_k/2) \\ 0 & 1 & \Delta t v_k \cos(\theta_k + \Delta t \varphi_k/2) \\ 0 & 0 & 1 \end{bmatrix} \quad (3.6)$$

3.2.2 Mechanical Platform

From the literature, the circular shape of the chassis has similarities with the hexagonal form. A regular hexagonal-shaped platform was adopted for the design of the body with a diagonal length of 0.381 m. Additionally, the platform includes two layers; the lower layer and the upper layer. The former layer comprises of the driving and steering mechanism of the mobile robot while the latter layer functions as a location for the laser scanner, battery, and other hardware components. The physical properties of the mobile robot together with its gear and bevel gear transmission details are listed in Table 3.1.

Table 3.1

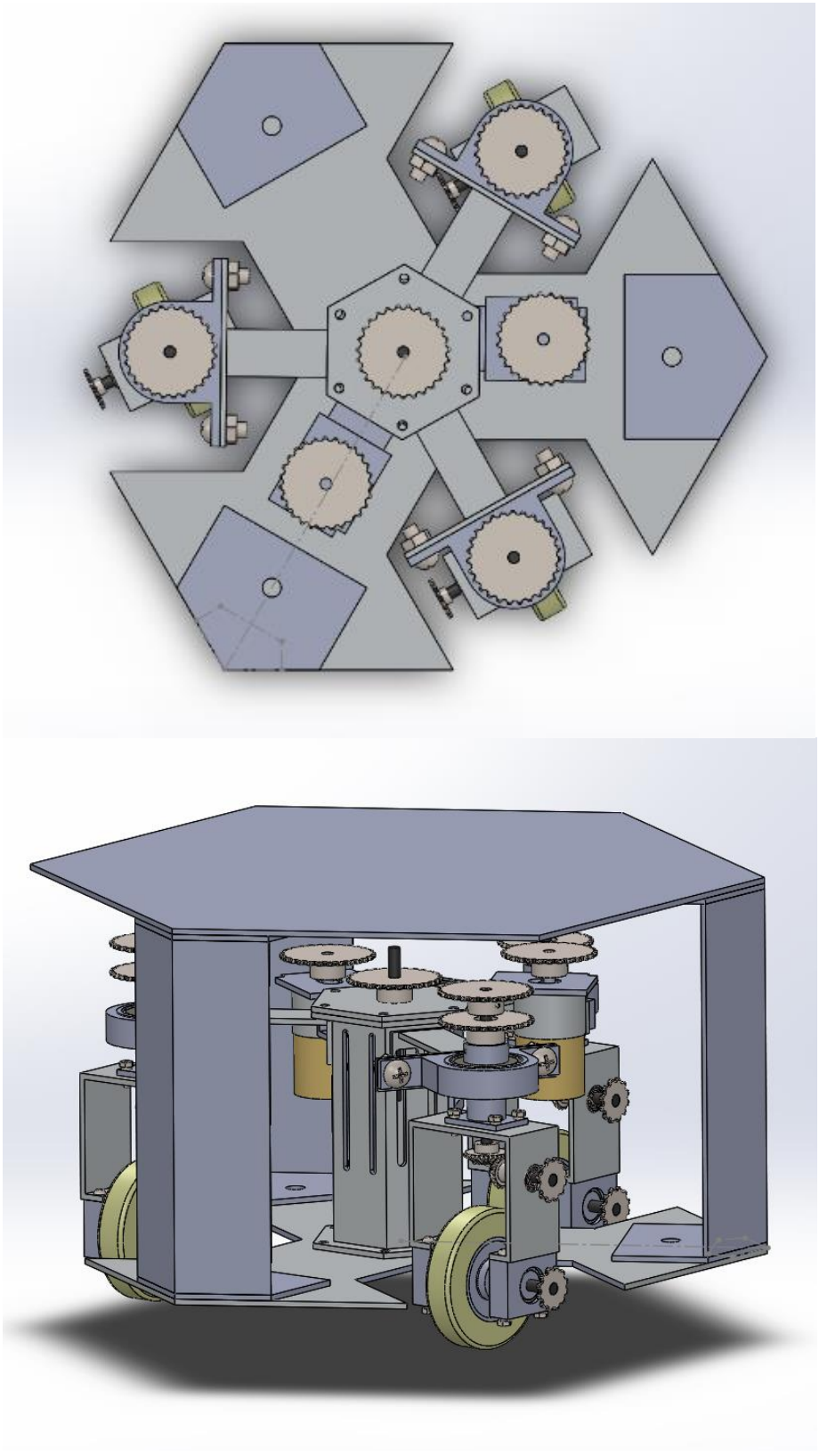
Physical Properties of the Synchronous-drive Mobile Robot

No.	Parameters	Value (Unit)
1	Dimension	
	Diameter length	0.381 (m)
	Platform height	0.4064 (m)
	Robot weight	3.0 (kg)
	Robot platform material	Aluminum
2	Gear Transmission	
	Gear teeth (Big)	27 (T)
	Gear teeth (Small)	9 (T)
	Driving Mechanism GR	2/3
	Steering Mechanism GR	1
3	Bevel Gear Transmission	
	Bevel gear teeth (driven)	20 (T)
	Bevel gear teeth (drive)	30 (T)
	Bevel Gear Ratio	1:1.5

The rotational motion from the driving motor and steering motor was delivered to the standard wheels through the chain-sprocket mechanism because of its efficiency in slippage compared to the belt type. These motions can be alternatively called as the driving and steering mechanisms and will be further discussed in Sections 3.2.4 and 3.2.5. The upper chain layer in the drive system actuates the robot forward or backward while the lower chain layer turns the system at a specific angle. Figure 3.2 illustrates the 3-dimensional model of the robotic system in SolidWorks.

Figure 3.2

3-D Model of the Mobile Robot Platform in SolidWorks

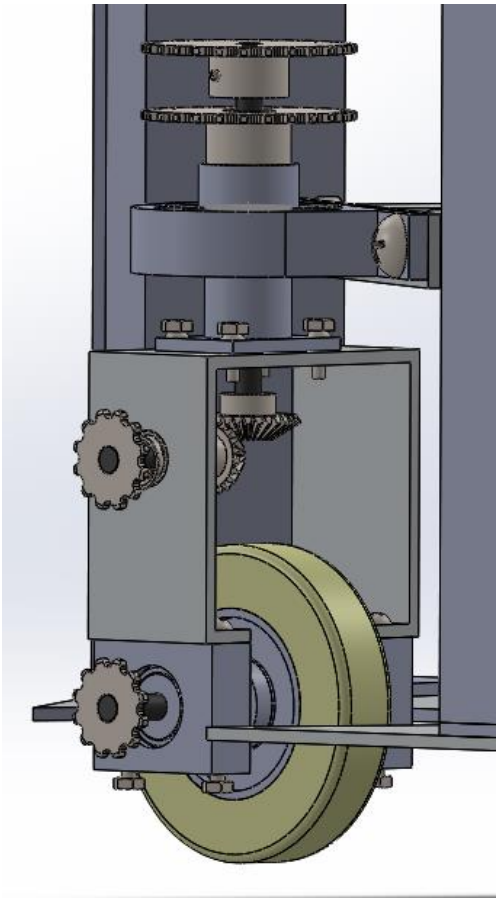


Additionally, the wheel transmission design is incorporated in this study to separately the movement of the steering and driving motors. Each main sprocket in the upper chain layer is connected to a shaft that is fastened to a straight bevel gear. The gear ratio of the driver and driven gears is 1:1.5. The driven gear is again connected to a shaft where a small sprocket is attached to the other end. The small sprocket is then mechanically couped by a chain to another small sprocket having a gear ratio of 1. The driven sprocket is then connected to a shaft where the standard wheels are fixed.

For the lower chain layer, each sprocket is attached to the body of the wheel design with the support of a bearing. The bearings are affixed to the main foundation of the mobile robot which allows the wheels to turn at a defined angle. Figure 3.3 shows the design of the wheel structure in SolidWorks.

Figure 3.3

Wheel Transmission Design of the Mobile Robot

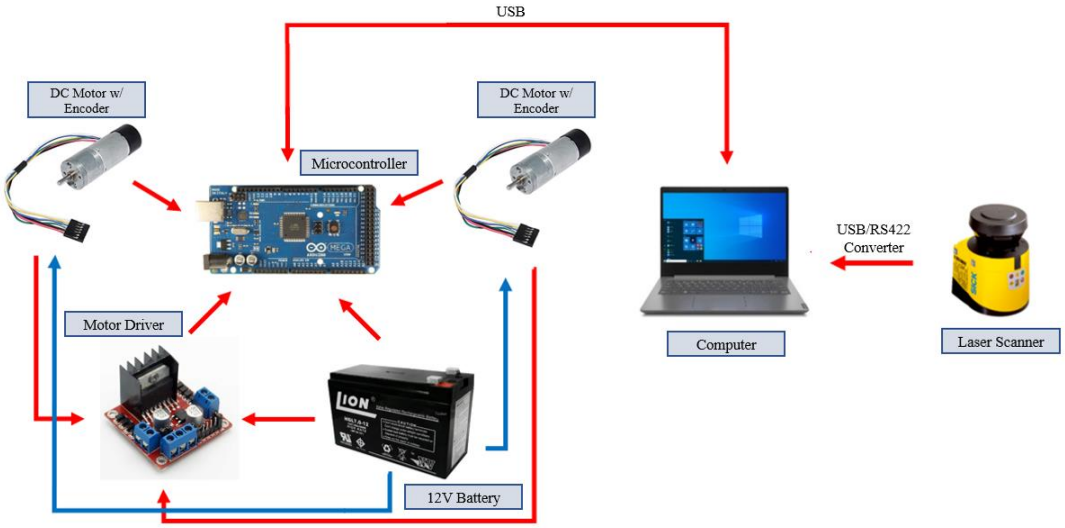


3.2.3 Hardware Design

Based from the SolidWorks model, the components of the mobile robotic system are illustrated in Figure 3.4. Components such as the two DC motor with encoder, an L298N motor driver and 12 DC battery are connected to the pins of the Arduino Mega 2560 microcontroller. The microcontroller is connected to the computer, the brain of the system, that executes the program and controls the motors using ROS. Connected to the computer is also the SICK S300 laser scanner which delivers the output raw data via the RS-422 to USB interface into the computer using the Python programing language. The raw data is processed and implemented in the localization and mapping package in ROS. The implementation of ROS to the actual hardware is further explained in Section 3.6. Figure 3.4 shows the connection diagram of the components in the mobile robot.

Figure 3.4

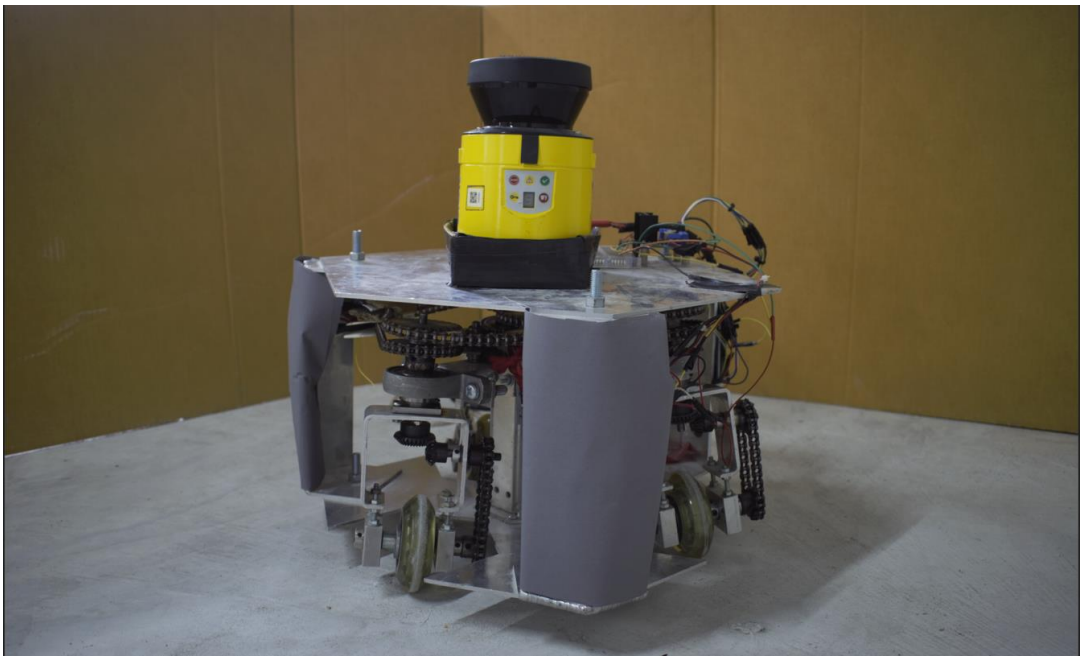
Hardware Connection Diagram



Given with the connections for each unit, the actual mobile robot used in this study was illustrated in Figure 3.5.

Figure 3.5

Actual Mobile Robotic Platform

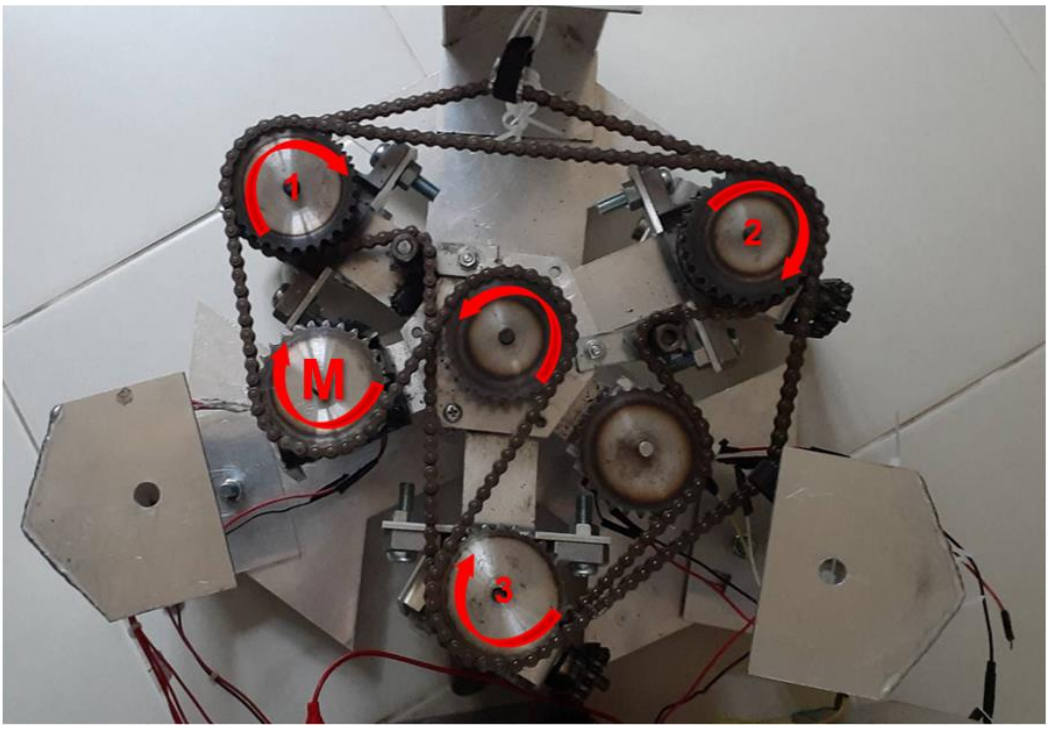


3.2.4 Driving Mechanism

A chain transmission of five sprockets contributing to the drive mechanism of the robot is shown in Figure 3.6. This design allows movement of the three wheels forward or backward in unison. Three of the sprockets (labeled as 1, 2, and 3 in Figure 3.6) are connected to the designed wheel structure mention in Section 3.2.2. The sprocket attached to the shaft of the driving motor is also specified in the figure with the letter "M". An additional sprocket in the middle of the robot was included in the chain transmission to provide a larger surrounding chain area in the motor sprocket and to tighten the chain. Without including this sprocket, the chain may loosen and unable to transmit the mechanical power of the motor.

Figure 3.6

Mobile Robot Driving Chain Mechanism



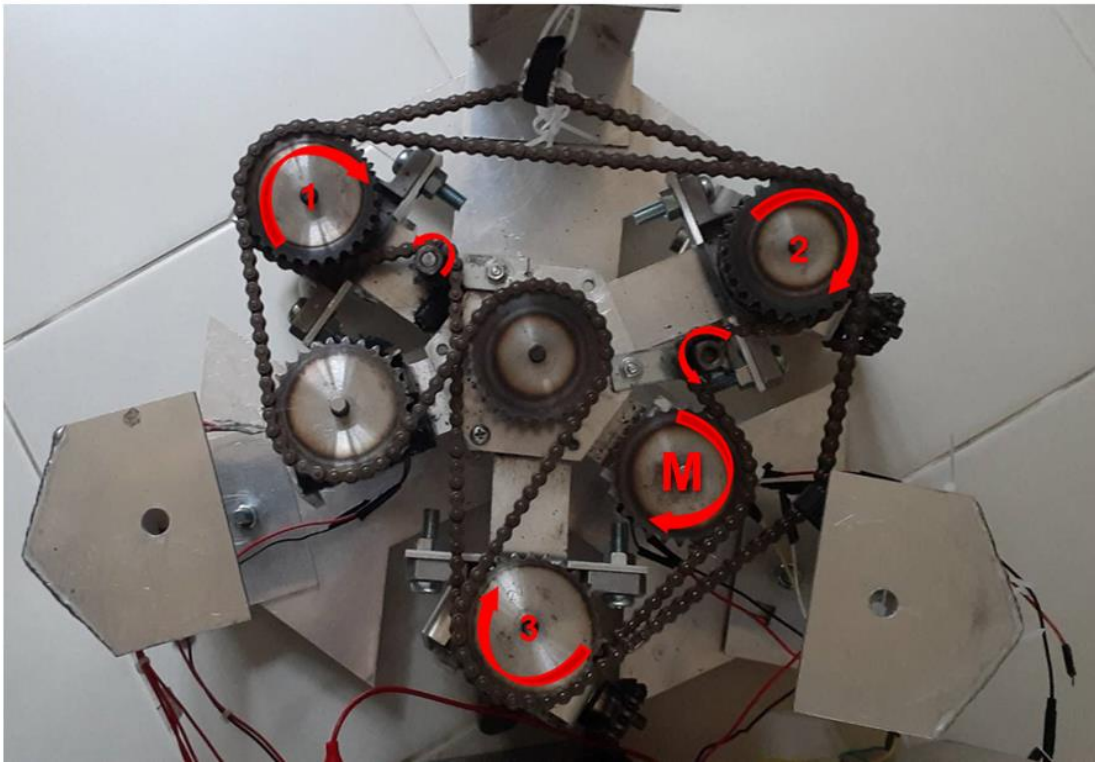
3.2.5 Steering Mechanism

The actual design of the steering mechanism is shown in Figure 3.7 which consisted of four large sprockets and two small sprockets. Numbered sprockets in the figure are connected to the body of the wheel structure for wheel rotation which was also discussed in Section 3.2.2. The sprocket connected to the steering motor shaft is labeled as “M” in the figure. Inclusion of small sprockets in the steer transmission secures the chain in place and don’t change the direction of rotation of the consequent larger sprocket. Moreover, the chain transmission arrangement synchronously rotates the wheel structure either in the clockwise or counterclockwise manner.

Note that for each layer in the chain-transmission, the sprockets must be horizontally aligned for the chain not to be uncoupled with the sprocket. Unaligned sprockets will not move the mobile robot and potentially break the chain.

Figure 3.7

Mobile Robot Steering Chain Mechanism



3.3 Mobile Robot Components and Sensors

Sensors are crucial for localization and mapping as it serves as eyes for perceiving the environment and provides information of the robot itself. As mentioned in Section 3.2.3, these devices include DC motors and laser scanners. Furthermore, the information and communication of the external sensors is transmitted to the microcontroller which delivers an equivalent signal to the motors. Discussion of these components and sensors are in the following sections.

3.3.1 DC Motor with Encoder and Calculations

A quadrature encoder in the DC motor is used to determine the distance travelled by the mobile robot and provided feedback for the system. The rotation from each DC motor is transferred to the wheels through the chain-sprocket transmission. Figure 3.8 shows the image of the DC motor with encoder used in this study. In addition, the specifications of the 12 ppr DC motor with encoder are listed in Table 3.2.

Figure 3.8

Illustration of the DC Motor and Attached Optical Encoder

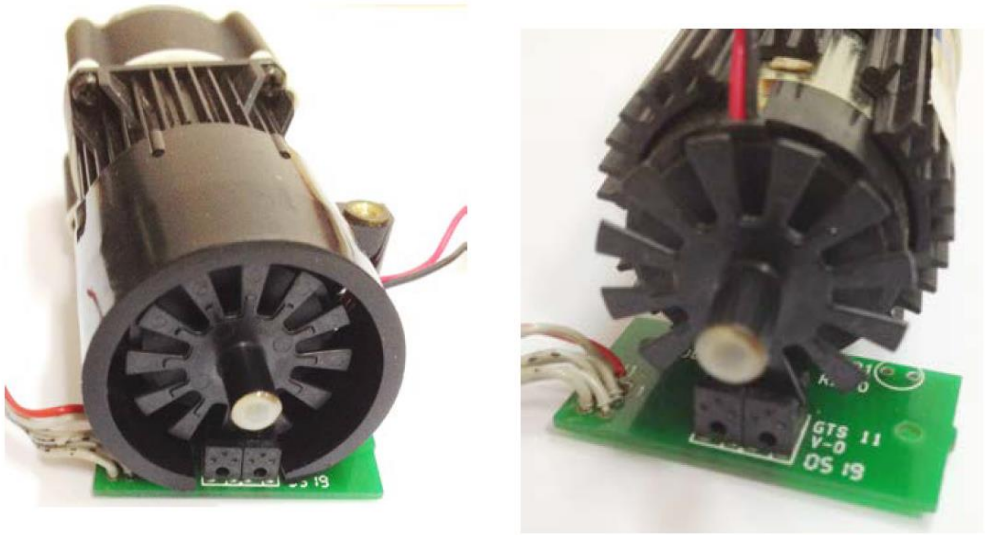


Table 3.2

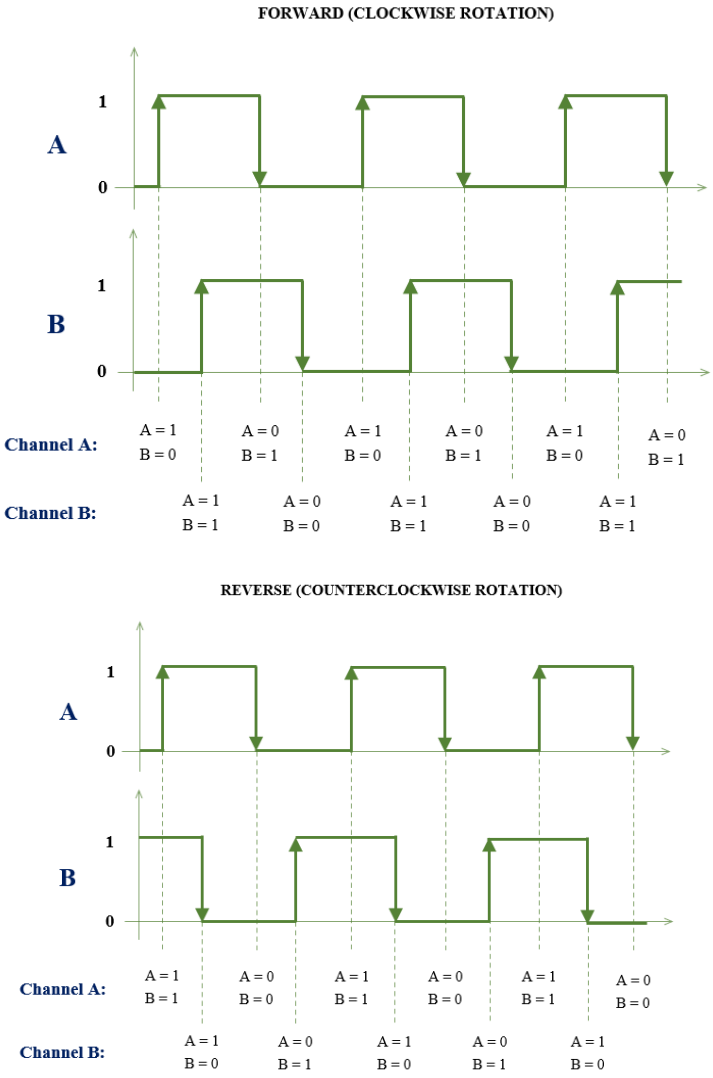
Technical Data of a DC Motor with Optical Encoder

Parameter	**Description**
Working Voltage	12 V (DC)
No load speed	8100 RPM (no gear)
Nominal output power	18 W No load current: 75 mA Load current: 1400 mA
Gearbox ratio	64:1
Encoder phase	AB
Encoder resolution	12 ppr

The position and direction measurement in the encoders is achieved through optical means and generated square-wave pulses. Furthermore, the direction of the counting movement was determined depending on the offset between the Channel A and Channel B shown in Figure 3.9.

Figure 3.9

Quadrature Encoder Output Pulses for Clockwise and Counterclockwise Rotation



Note. This image was adapted from "Encoders" H. H. Manh, 2020, p. 5.

If the signal output of Channel A leads the signal output of Channel B, the direction of the counting device is in the clockwise direction. Conversely, Channel A lagging the pulse of Channel B signifies a counterclockwise direction of the counting device. Thus, these behaviors identify the position and direction of the rotary motion.

To receive the necessary measurements, the number of edges (high to low or low to high transitions) is be considered and then converted to a corresponding position. The resulting positions from the encoder depends on three encoding types; X1, X2, and X4. This study used X4 encoding which provides a more precise counter reading and higher

resolution compared to the former encoding types. Implementing a X4 encoding into the conversion means having a total of 48 pulses per revolution (PPR) (Manh, 2020). Calculations of the mobile robot's wheel rotation, distance, speed and RPM are considered as follows.

3.3.1.1 **Wheel Rotation Calculation.** The wheel distance is calculated using the X4 encoding, number of generated pulses, and the gear ratio. The equation for the calculation of wheel revolution is shown in Equation (3.7).

$$Wheel\ Rotation = \frac{currentPos - lastPost}{ppr \times et\ \times GR} \tag{3.7}$$

where ppr is the pulses per revolution, et is the encoding type (X4), and GR is the gear ratio of the transmission system.

3.3.1.2 **Wheel Distance Calculation.** Based from the number of wheel rotations, the wheel distance is calculated using Equation (3.8).

$$Wheel\ Distance = 2\pi(Wheel\ Rotation) \tag{3.8}$$

3.3.1.3 **Time Rate of Change Calculation.** To determine the time difference between the starting point to its endpoint, the rate of change in time (minute) is calculated in Equation (3.9).

$$dt = \frac{currentTiming - lastTiming}{6000} \tag{3.9}$$

3.3.1.4 **Rotational Speed Calculation.** The speed of the shaft delivered to the coupled wheels is calculated using the rotation number calculation in Equation (3.7) and the time difference in Equation (3.9). Equation (3.10) shows the equation of the rotational speed.

$$Rotational\ Speed = \frac{d(Rotation\ Number)}{6000} \tag{3.10}$$

Equations (3.7) to (3.10) are incorporated in the Arduino and ROS code which are used for localizing the mobile robot.

3.3.2 Motor Driver and Microcontroller

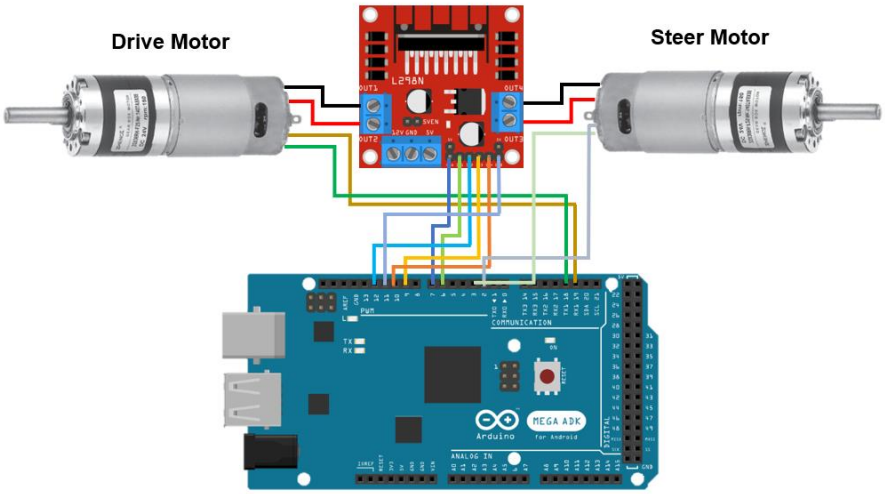
The L298N Motor Driver, which consists of an L298 motor driver IC and a 78M05 5V regulator, allows controlling two (2) DC motors. Apart from this function, the double H-bridge design controls the rotational direction of the motors while the Pulse Width Modulation (PWM) allows to control the speed.

This module consists of pins for the motor power supply (Vs), ground and 5V logic power supply. Moreover, the output pins for the drive motor and steer motor with voltages between 5 to 35V DC are incorporated in the driver. The motor driver also includes direction control pins (Input 1, Input 2, Input 3, Input 4) which controls the forward or backward motion and the speed control pins (Enable A and Enable B) which control the speed for each motor. The positive terminal of the power supply (battery) was connected to the VS while the negative terminal is connected to GND.

The main controller board used in this research is the Arduino Mega 2560 microcontroller which is based on the Atmega2560. It has 54 digital input/output pins (14 pins provide PWM output), 16 analog input pins, 4 hardware UARTs that operates at 5V, and a 256 KB flash memory. The Arduino Mega board can be powered either through a USB connection or an external power supply of 6 to 20V. The connections between the microcontroller and motor driver are shown in Figure 3.10 (See Appendix A1 for pin connections).

Figure 3.10

L298N Motor Driver Board and Arduino Mega 2560 Connection to DC Motor with Encoder

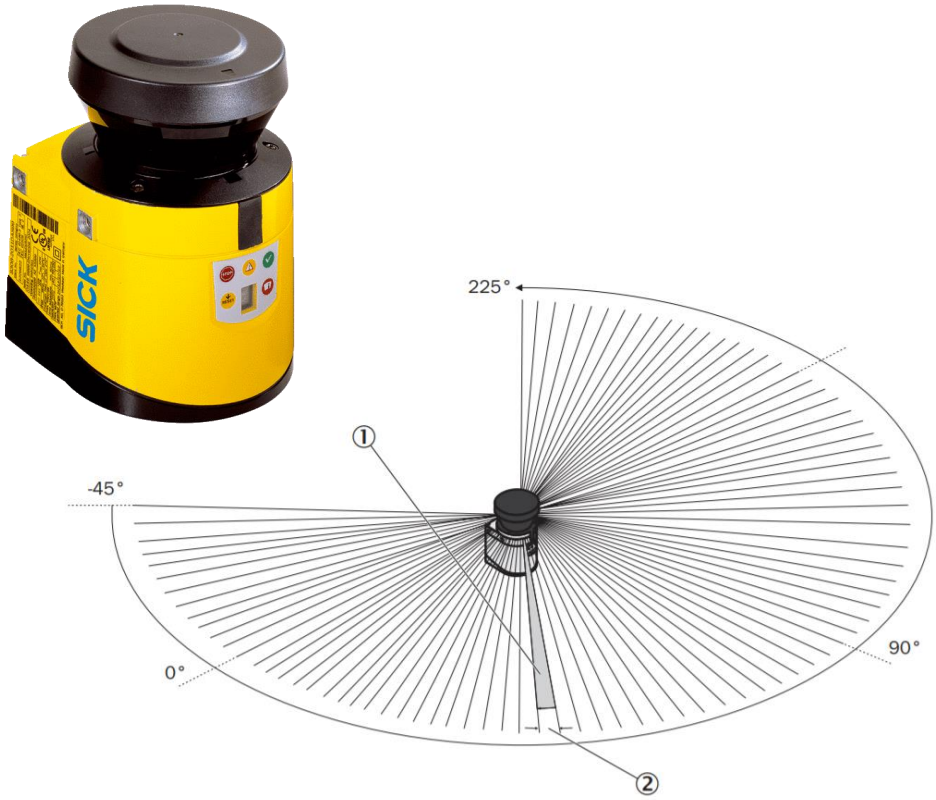


3.3.3 Sick S300 Laser Scanner Sensor

The SICK S300 safety laser scanner is used in this research to accurately determine the distance between the laser scanner's position in the mobile robot and to the environment (with or without obstacles). The actual image of the laser scanner which has a field of view of 270° is illustrated in Figure 3.11.

Figure 3.11

Illustration of S300 Expert SICK Laser Scanner and Its Scan Plane



The SICK laser scanner has an angular resolution of 0.5° meaning that every planar range scan yields to an increment of 0.5° starting from 0 until angle 270. The measuring range of this scanner is 30 m and has a warning field range of 8 m. Also, an RS-422 data interface is used to transmit the data into the computer with a transmission rate of 500k baud. Additional details of the laser scanner are recorded in Table 3.3.

Table 3.3

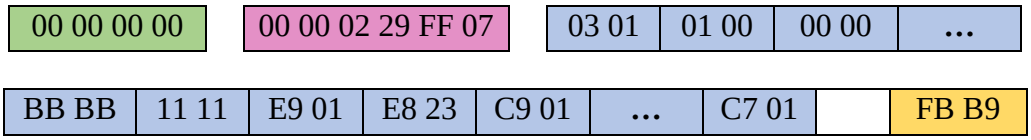
Technical Data of SICK S300 Laser Scanner

Parameter	Description
Type	S30B-3011GB (Expert)
Protective Field Range	2 m
Scanning Angle	270°
Response Time	80 ms
Angular Resolution	0.5°
Data Interface	RS-422
Transmission Rate	≤ 500 kBaud
Operating Voltage	24 V DC
Power Consumption	≤ 0.33 A (without output load) ≤ 1.7 A (with max. output load)
Weight	1.2 kg
Dimensions (W x H x D)	102 mm x 152 mm x 106 mm

3.3.3.1 **Laser Scanner Data Processing.** The laser scanner continuously sends the output raw data via RS-422 interface to the computer and is processed using the Python programming language. The structure of data output on every scan consist of a telegram header (4 bytes, green), an administration data (6 bytes, purple), a measured data (1,132 bytes, blue), and a CRC (2 bytes, yellow) that displays the scanner's details and version. An example of the telegram structure from the laser scanner is presented in Figure 3.12 (SICK Sensor Intelligence, 2015; Yan, 2019).

Figure 3.12

Sample Telegram Structure from a Continuous Data Output



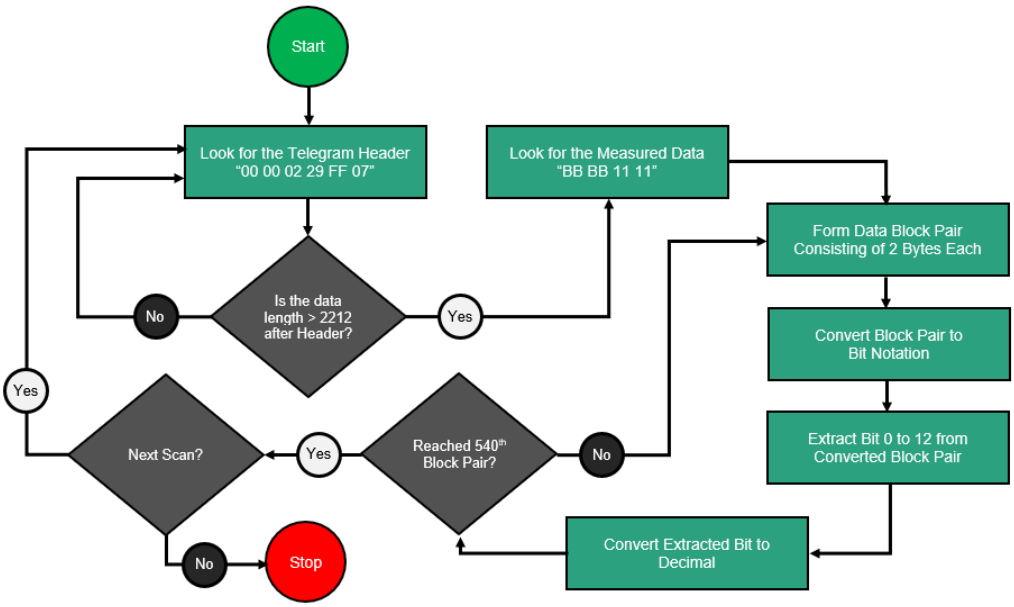
Three types of operation are available in the Sick S300 laser scanner: I/O Information, Measurement Data, and Reflective Data. These modes have starting characters of AA AA, BB BB, and CC CC, respectively, depending on the configuration. This research considered the measurement data to attain the estimated distance from the laser scanner and the considered environment at every 0.5 ° increment from angle -45 to 225.

Furthermore, output raw data following 11 11 are the necessary data to be extracted. Data blocks after 11 11 is shown in Figure 3.12.

3.3.3.2 **Output Raw Data to Distance Calculation.** Given with a continuous raw output data from the laser scanner, an appropriate method in organizing the laser scan data is implemented. Figure 3.13 shows the process on how the output raw data from the laser scanner is extracted and converted into the desired distance format.

Figure 3.13

Step-by-step Process of Laser Scanner Output Raw Data



For every scan, a telegram header consisting "00 00 02 19 FF 07" is checked from the series of output raw data. The data length is then verified by counting the length after the administration data (see Figure 3.12). Data lengths with less than 2212 is rejected and looks for the next telegram header. Having a true condition continues to search for the measurement data of "BB BB 11 11".

After the measurement data, a data block pair consisting of 2 bytes is created. This data pair is then converted to bit notation and extracted only bits 0 to 12. The extracted bits are then converted to decimal notation to obtain the measured distance in centimeters. The code also checks whether the data block pair reached the 540th term. If false, the

code will move to create another data block pair. If true, the operation proceeds to the next laser scan. The process of finding the next scan is terminated once a stopping condition is initiated by the user.

To further understand the raw output data conversion and acquisition, a sample calculation is illustrated as follows.

(Data block no.1) E9 01
Measured value at 0 degree is 0x01E9
bit notation: 0000 0001 1110 1001
Bit 13: 0: no reflector detected
Bit 12 … 0: distance in cm: 0x01E9 = 0000 0001 1110 1001
In cm: 256 cm + 128 cm + 64 cm + 32 cm + 8 cm + 1 cm = 489 cm

(Data block no.2) E8 23
Measured value at 0.5 degree is 0x23E8
bit notation: 0010 0011 1110 1000
Bit 13: 0: no reflector detected
Bit 12 … 0: distance in cm: 0x23E8 = 0000 0011 1110 1000
In cm: 512 cm + 256 cm + 128 cm + 64 cm + 32 cm + 8 cm = 1,000 cm

(Data block no.3) C9 01
Measured value at 1.0 degree is 0x01C9
bit notation: 0010 0011 1110 1000
Bit 13: 0: no reflector detected
Bit 12 … 0: distance in cm: 0x01C9 = 0000 0001 1100 1001
In cm: 256 cm + 128 cm + 64 cm + 8 cm + 1 cm = 457 cm

⋮

(Data block no.540) DB 01
Measured value at 270.0 degree is 0x01DB
bit notation: 0000 0001 1101 1011
Bit 13: 0: no reflector detected
Bit 12 … 0: distance in cm: 0x01DB = 0000 0001 1101 1011
In cm: 256 cm + 128 cm + 64 cm + 16 cm + 8 cm + 2 cm = 474 cm

The following raw output data conversion is obtained from the Sick S300 Manual (SICK Sensor Intelligence, 2015).

3.4 Position Control Method

External sensors are insufficient in achieving the desired output and as a result, a feedback control system should be utilized to attain this output. The proportional-integral-derivative controller or PID controller consists of three gains (P-gain, I-gain, and D-gain) that controls the input based from the error value calculated from the measured process variable and set point. Mathematically, the PID controller can be expressed together with the three gain constants (K_P, K_I, K_D) given by the equation,

$$u(t) = K_p e(t) + K_I \int_0^t e'(t)\, dt' + K_D \frac{de(t)}{dt} \tag{3.11}$$

Each of the component in the equation represents a basic control behavior which is briefly explained. The proportional controller or P-control applies a correction proportional to the error which is the difference between the setpoint value (SP) and process variable (PV). A faster response is one of the advantages of this controller, however, it doesn't always reach the desired setpoint and may show an offset. The integral controller or I-control considers the sum of the error over time and serves as a solution to remove the steady-state error. The problem of this controller is showing a slower response time which can destabilize the controller. The third term and last term in the PID controller is the derivative control or D-control. This controller's role is to minimize the change of error which depends on the rate of change in error. It gives a more stable response since it acts as a brake or dampener that resists the change of value.

In this study, the PID library installable from the Arduino IDE Library Manager is applied and the controller parameters used in the position control for driving and steering are illustrated in Table 3.4.

Table 3.4

Proportional-Derivative (PD) Control Parameters

Position Control (Driving)		**Position Control (Steering)**	
$k_{p,d}$	4.0	$k_{p,s}$	3.0
$k_{d,d}$	0.01	$k_{d,s}$	0.01

3.5 2D EKF-SLAM Operation

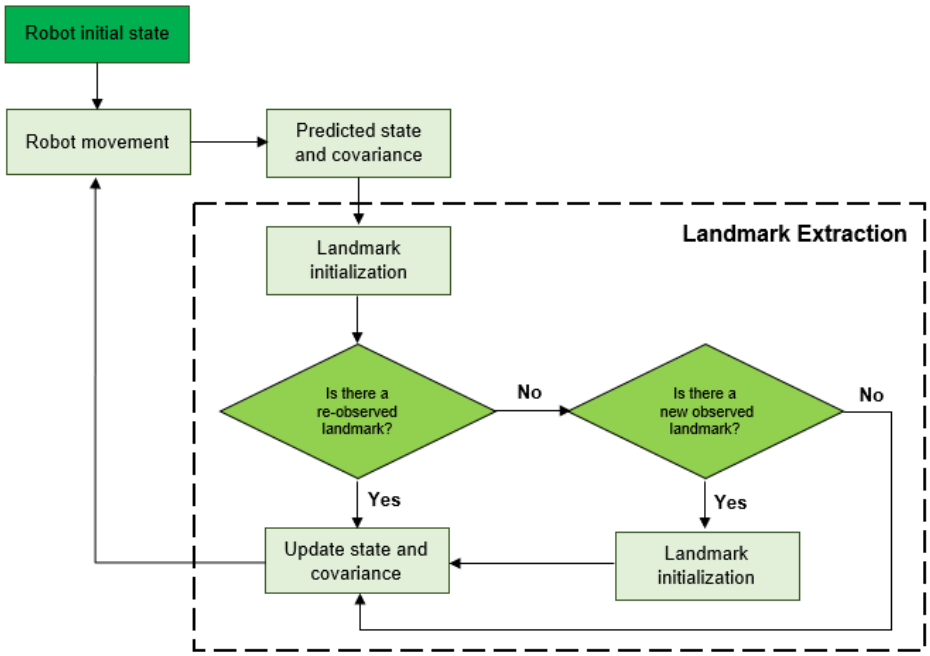
The overview of the EKF-SLAM implemented in ROS is shown in Figure 3.14.

The EKF-SLAM process starts by initializing the position and orientation of the robot where in this case, (x, y,θ) is (0,0,0). The robot then starts moving at a certain direction. A predicted state and covariance are produced based from the odometry of the robot. The estimated state consisted of the estimated robot state (X_v) and estimated landmark states (X_l) shown in Equation (3.12). The nth index represents the n number of registered landmarks. The laser scanner then measures the distance from the location

of the scanner attached to the robot and to the environment or obstacle. Point clustering is used for detecting the landmark with a minimum of 15 clustered points. Fewer clustered points result to a rejection of the point cluster that is not considered as a landmark.

Figure 3.14

General Operation Step for Implementation 2D SLAM



Two conditions exist for the landmark extraction, one for the re-observed landmark and the other is the new landmark. Once a registered landmark is detected, the state and covariance matrix are updated. For new observations, on the contrary, the landmark is first initialized then updated its state and covariance. With every new observed landmark, Equations (3.13) and (3.14) will expand in size. Equations (3.12) to (3.14) represent the correspondent estimated state error and covariance matrix, respectively.

$$X = \begin{bmatrix} X_v \\ X_l \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \\ L_{x_1} \\ L_{y_1} \\ \vdots \\ L_{x_n} \\ L_{y_n} \end{bmatrix} \tag{3.12}$$

$$\bar{X} = \begin{bmatrix} \overline{X_v} \\ \overline{X_l} \end{bmatrix} = \begin{bmatrix} \overline{X_v} \\ \overline{L_1} \\ \vdots \\ \overline{L_1} \end{bmatrix} \tag{3.13}$$

$$P = \begin{bmatrix} P_{X_v,X_v} & P_{X_v,X_l} \\ P_{X_l,X_v} & P_{M,X_l} \end{bmatrix} \tag{3.14}$$

3.5.1 Extended Kalman Filter SLAM Equation

For every time step, the EKF SLAM takes odometry and sensor measurements to generate an estimate of the full state vector.

3.5.1.1 Initialization. For initialization, the robot state is set at (0,0,0) and covariance is also initialized to zero which indicates the initial state of the robot is known. Additionally, there are still no registered landmarks to the map at this moment so the mean and covariance at the initial time is zero.

$$\bar{X} = \hat{X}_v = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad P = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.15}$$

3.5.1.2 Prediction Step. When the robot is in motion, the prediction step predicts the position of the robot based from motion commands. The updated estimated robot state is given in Equation (3.16) and the updated estimated landmark state is shown in Equation (3.17).

$$X_{(k+1)} = \begin{bmatrix} x_{(k)} \\ y_{(k)} \\ \theta_{(k)} \end{bmatrix} + \begin{bmatrix} \Delta t v_k cos\,(\theta_k + \Delta t \varphi_k / 2) \\ \Delta t v_k sin\,(\theta_k + \Delta t \varphi_k / 2) \\ \Delta t \varphi_k \end{bmatrix} \tag{3.16}$$

$$X_L \leftarrow X_L \tag{3.17}$$

where X_{k+1} represents its state at time interval k+1, the first term represents the previous robot state and the second term represents the robot motion model. Furthermore, the estimated landmarks are also updated. Also, the robot's covariance, P, is calculated using Equation (3.18) where F_x is the Jacobian matrix and Pn corresponds to the noise covariance

$$P \leftarrow G_x P {G_x}^T + Pn \tag{3.18}$$

3.5.1.3 Correction Step. The purpose of the correction step is to compare the actual observation from the sensors with the predicted measurements to correct the robot state. Equations (3.19) to (3.23) represent the updating process consisting of three

stages: Calculation of the Kalman gain, correction of the estimated mean, and correction of the estimated covariance.

$$\bar{z} = y_i - h_i(X, L_i) \tag{3.19}$$

$$Z = H_x P {H_x}^T + R \tag{3.20}$$

$$K = P {H_x}^T Z^{-1} \tag{3.21}$$

$$\bar{X} \leftarrow \bar{X} + K\bar{z} \tag{3.22}$$

$$P \leftarrow P + KZK^T \tag{3.23}$$

where $\bar{z}$ represents the measurement model, K represents the Kalman gain for the updating process, H represents the Jacobian (Saputra, 2015).

3.5.2 Experimental Setup of the Real Environment

Figure 3.15 shows the environment used in the experiment where the landmarks are labeled as Landmark 1, Landmark 2, and Landmark 3. Also, the dimensions of the landmarks are stated in Table 3.5.

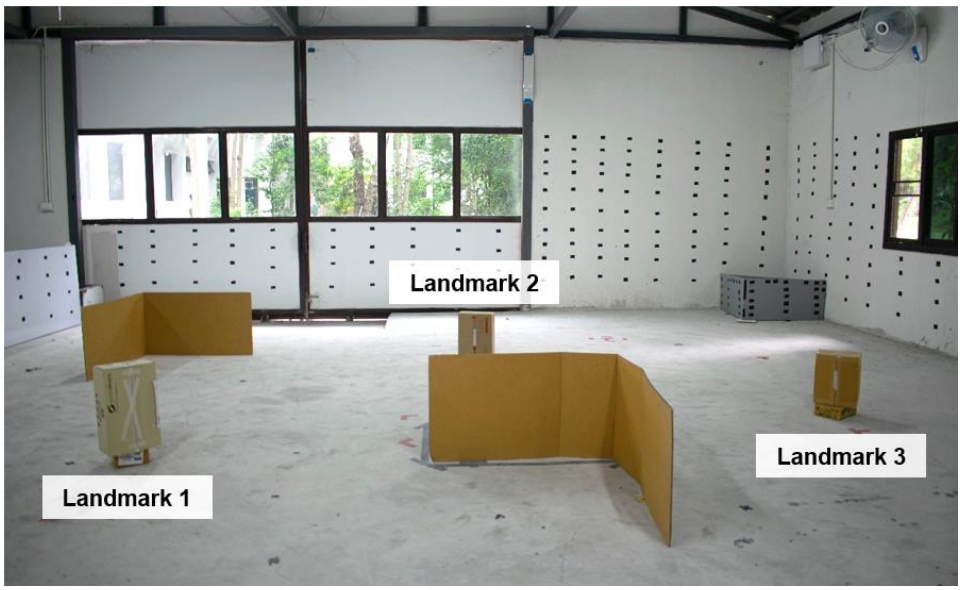
Table 3.5

Landmark Dimensions in terms of Length, Width, and Height

Box No.	Length (m)	Width (m)	Height (m)
Landmark 1	0.22	0.14	0.37
Landmark 2	0.23	0.13	0.30
Landmark 3	0.21	0.15	0.27

Figure 3.15

Image of the Real Environment with Landmark



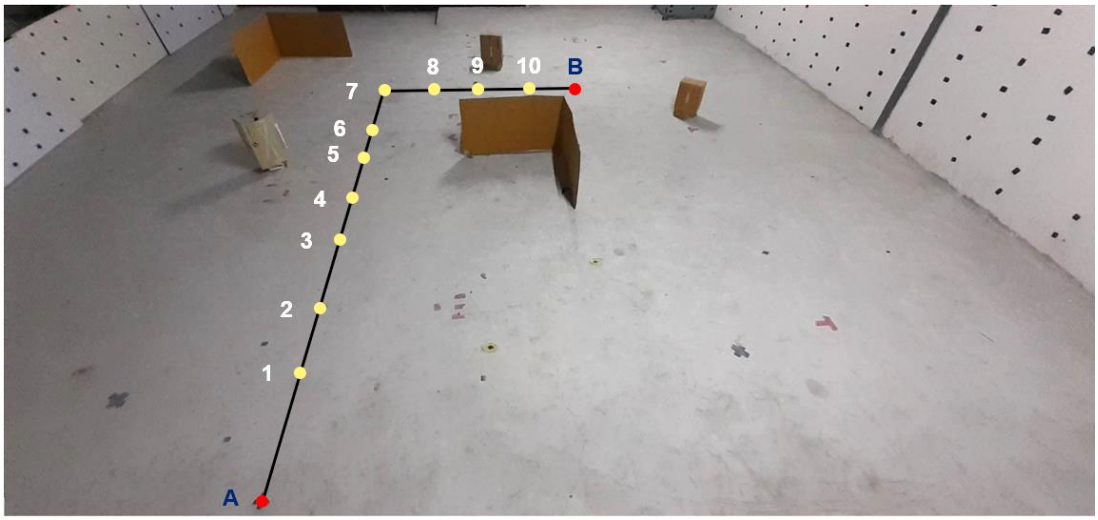
Moreover, the path of the robot from the starting position "A" to the final point "B" is shown in Figure 3.16. In between the starting and final points are 10 arbitrary points used for the comparing the ground truth and calculated path using EKF-SLAM. The comparison is calculated using the Root Mean Square Error shown in Equation (3.24).

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{x}_i - x_i)^2}{n}} \tag{3.24}$$

where n is the number of measurements, *i* is the i[th] index term, $\hat{x}_i$ is the predicted value, and x_i is the observed value.

Figure 3.16

Illustration of the Path of the Mobile Robot in the Environment



3.6 Implementation of EKF-SLAM in Robotic Operating System (ROS)

This research used the ROS Melodic Morenia version (released in May 23, 2018) using Linux Ubuntu 18.04 (Bionic). The laptop used has a dual booted Linux kernel to implement ROS. The Melodic version provides well documented packages necessary for mapping and localization and allows the communication between the Arduino microcontroller and ROS.

3.6.1 Arduino Microcontroller and ROS

The process of connecting the microcontroller to ROS is through the installation of the rospackage named *rosserial* in the Ubuntu terminal. To move the robot at a specific

direction and speed, each wheel mechanism (drive or steer) velocity was published under the topic of “velocity_cmd” which ROS can subscribe. This topic delivers the velocity information from the motors. Additionally, the topic “cmd_vel” was published from the rospackage *teleop_twist_keyboard* which sends velocity commands to actuate the driving and steering motors when the microcontroller subscribes to it. In this research, the device port number used is */dev/ttyACM0*. The crucial and the most important part of this study is the mapping and localization using EKF in ROS. The steps for implementing SLAM together with its requirements are as follows.

3.6.2 Coordinate Transformations

First part of the requirements in the SLAM implementation is the connection between coordinate frames that allows transformation between the active frames to a desired point in time. This is also known as tf or the *transformation* package in ROS. Frames such as **/map**, **/odom**, **/base_link**, and **/base_laser** are included in this study (See Figure 3.17). Moreover, the *static_transform_publisher* publishes static coordinate

Figure 3.17

ROS Transformation Trees

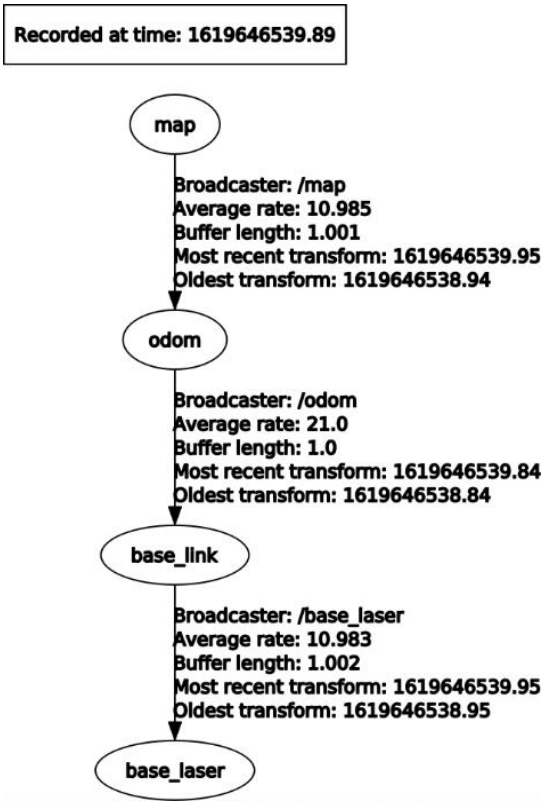
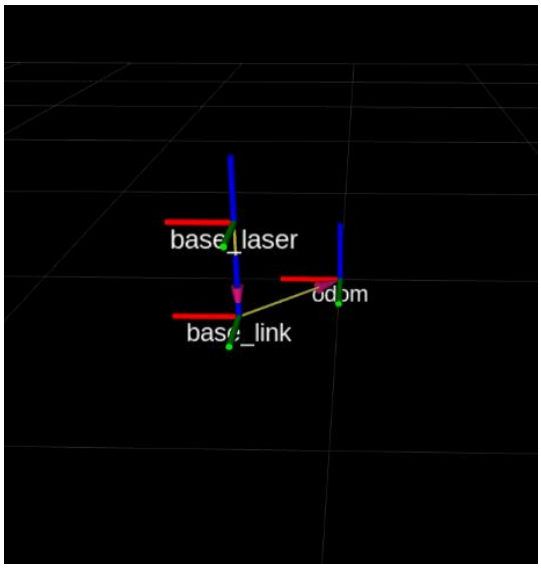


Figure 3.18

ROS Coordinate Frame Transformation on Rviz



transformation (x/y/z: meters, yaw/pitch/roll: radians) which is configured for the **/base_link** to **/base_laser** transformation. Tf, however, doesn't provide any information about the velocity of the robot so to transform the frames from **/map** to **/odom** and frames from **/odom** to **/base_link,** a dynamic transformation through odometry information was used. The odometry information, based from the *odometry_publisher* package, is incorporated into the *slam_in_control.cpp* file. The resulting transformation of the aforementioned frames are shown in rviz, 3d visualization tool for ROS, which was illustrated in Figure 3.18.

3.6.3 SICK Laser Scanner and ROS

ROS Melodic has an available package for the SICK S300 laser scanner for publishing the laser scan message which is the *cob_sick_s300* package. However, the said package showed error and didn't provide the desired output. To aid this dilemma, a code based in Python language is created to obtain the laser scanner data under the filename of *SickS300Scanner.py* (See Appendix B1 for more details). By following the telegram listing provided by the manufacturer of the laser scanner, the python file publishes the processed laser data based from the discussed procedure in Section 3.3.3 (SICK Sensor Intelligence, 2015). Details of the configuration between the laser scanner and ROS are shown in Table 3.6.

Table 3.6

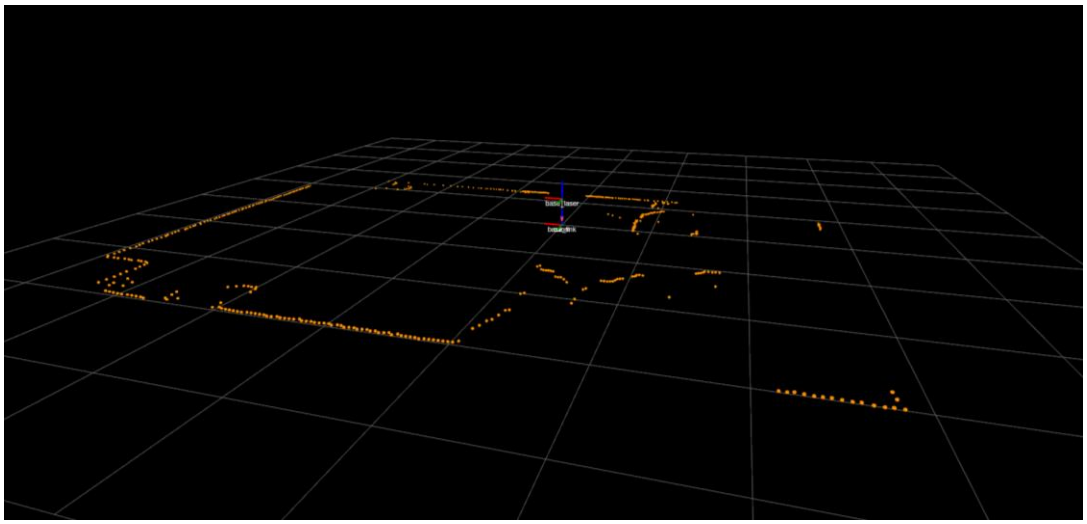
Sick S300 Scanner Parameters

Parameter	Description
Node Name	base_laser
Port	/dev/ttyUSB0
Baud Rate	500000
Parity Bit	None
Stop Bit	1 bit
Byte Size	8 bits
Scan Header ID	base_laser
Scan Angle (rad)	
- Maximum	3.926990
- Minimum	-0.7853983
Scan Range (m)	
- Maximum	29.00
- Minimum	0.03

A sample laser scan in a room including the laser coordinate frame is visualized in rviz and is shown in Figure 3.19.

Figure 3.19

Laser Scanner Output on Rviz



3.6.4 EKF-SLAM and ROS

The process of running EKF-SLAM and ROS starts with the terminal having each tab addressed to a specific IP Address with the command

```
export ROS_MASTER_URL=http://10.90.4.127:11311
export ROS_IP=10.90.4.127
```

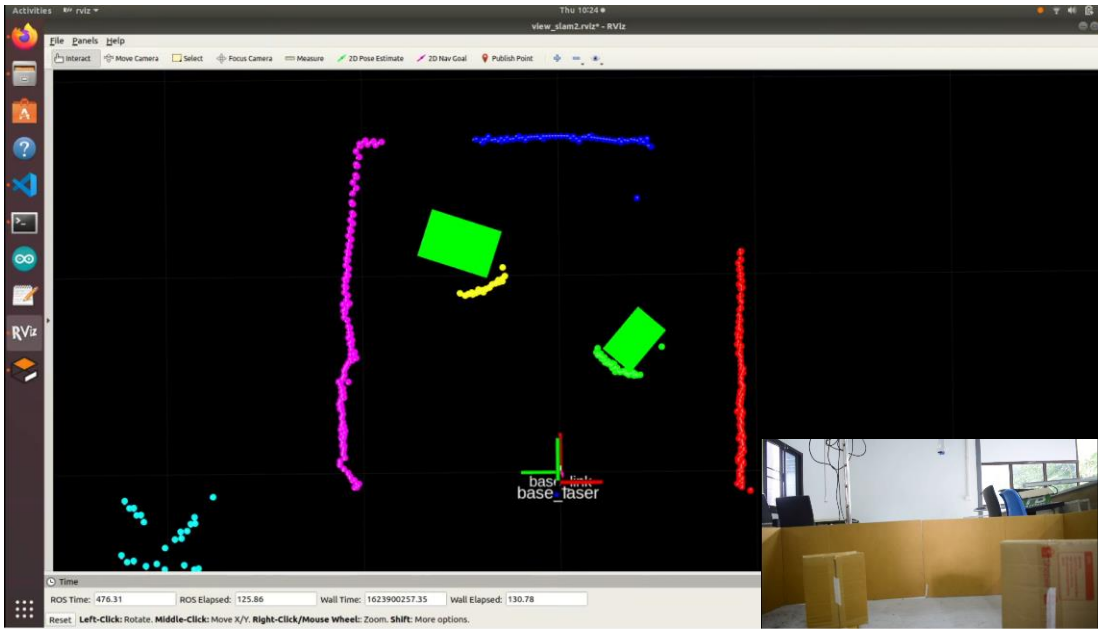
Including this command in the terminal connects the active terminals together. Afterwards, a set of instructions to open the simulator and data acquisition for each terminal tab is stated below.

```
1 roscore
2 rosrun rosserial_python serial_node.py port:=/dev/ttyACM0
3 roslaunch main main.launch
4 rostopic echo slam_path
```

Command 1 represents the master and is a requirement to run and allow communication between nodes. The second command initializes and connects the Arduino to the ROS middleware through the assigned port (**/dev/ttyACM0** is used in this research). Command 3 represents the main node for the EKF-Slam which includes the landmark detection and mapping the estimated landmark. This node also includes the transformation frames between the mobile robot parts, information from the wheel velocities from Arduino, and simulation of the actual environment. Lastly, Command 4 shows the numerical information of the where the robot travels to. Figure 3.20 shows the sample image of the landmark detection in rviz having landmarks shown in green.

Figure 3.20

Landmark Dectection on Rviz

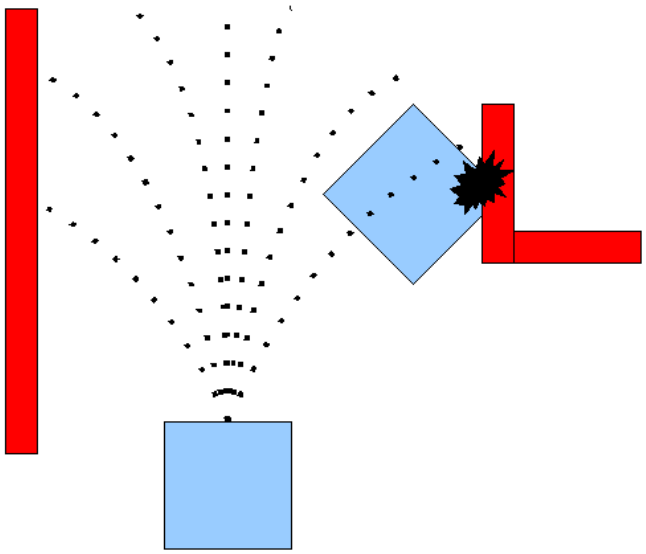


3.6.5 Obstacle Avoidance

The obstacle avoidance is included in this research through the navigation stack implemented in ROS using the Dynamic Window Approach (DWA) set as a local planner. The idea behind the DWA is that the system samples multiple sets of velocities which then simulates the valid and invalid trajectories of the robot. These trajectories are evaluated and the optimal trajectory resulting from the speed is chosen to be able to drive the robot. The dynamic window approach also limits the speed sampling space and calculates the lowest possible cost function. Figure 3.20 shows the illustration of possible trajectories for the mobile robot under investigation.

Figure 3.21

Dynamic Window Approach

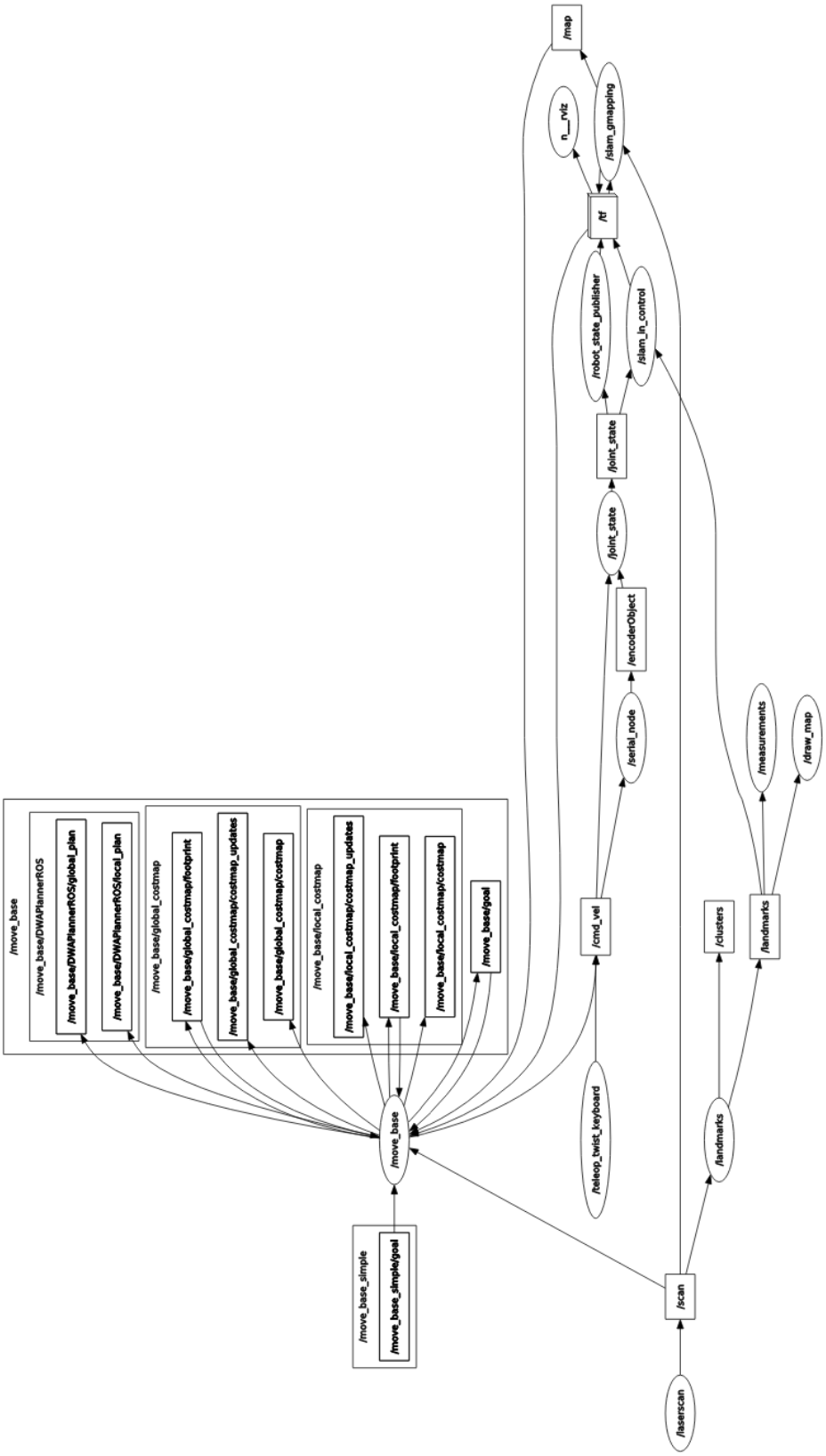


Note. This image is acquired from the website of dwa_local_planner in ROS. http://wiki.ros.org/dwa_local_planner

Figure 3.21 shows the nodes involved in the implementation of EKF-SLAM. Packages involved in the EKF-SLAM operation consists of the laser scanner node, the navigation stack, serial communication node to the microcontroller, EKF-SLAM node, landmark detection node, and the gmapping.

Figure 3.22

Illustration of the Full Rosgraph



CHAPTER 4
RESULTS AND DISCUSSION

4.1 EKF-SLAM Operation Results

The performance of the EKF-SLAM is evaluated through the comparison of the ground truth and the actual measurement by calculating the root mean square. Before this comparison, the results from the landmark extraction are discussed.

4.1.1 Landmark Detection from Laser Scanner

Figure 4.1 illustrates the scanner output data on the real environment. The orange point clouds represent each scan angle from the laser scanner.

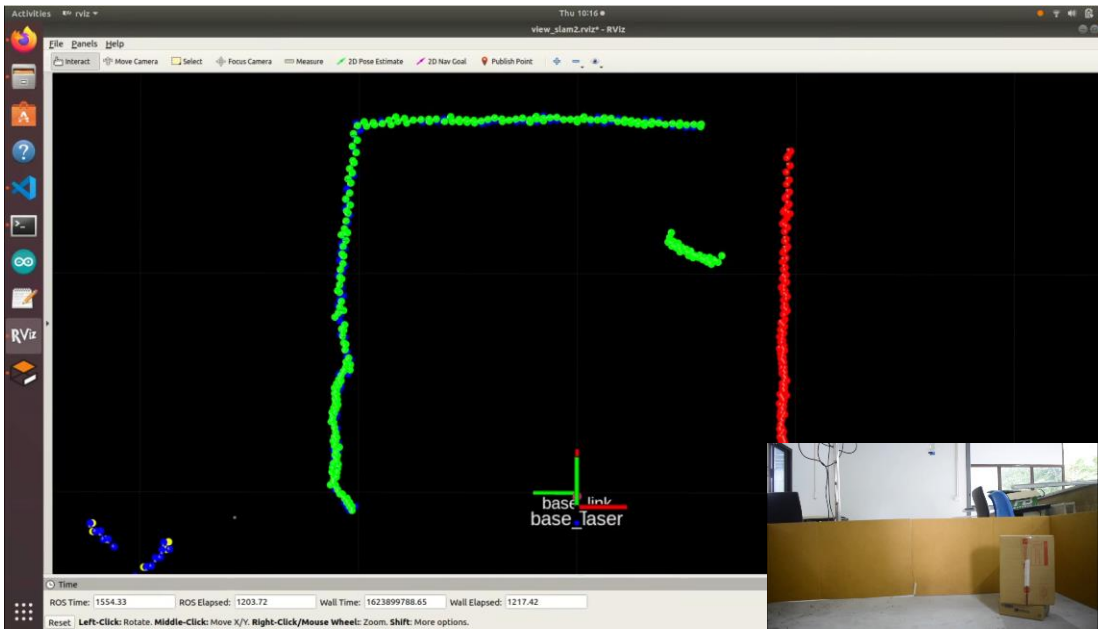
Figure 4.1

Laser Scanner Output on Real Environment on Rviz

It was also observed that point clusters merge with the side of the wall if the position of the landmark is near to the wall. Based from the trial shown in Figure 4.2, the feature detection fails to distinguish a landmark having the landmark's coordinates (X:1.16 m and Y:0.55 m). The green point cloud shows that the landmark is also considered as a wall.

Figure 4.2

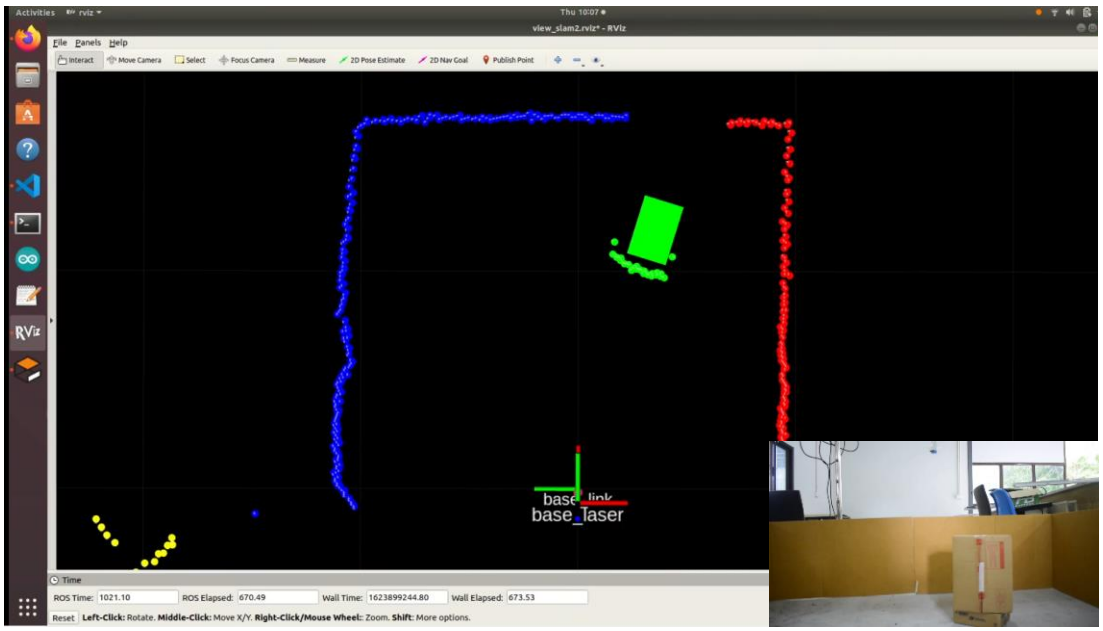
Laser Scanner Output on Rviz without Landmark Consideration



By moving the landmark away from the wall, the landmark, having it's coordinates at (X:1.04 m and Y:0.30 m), is considered. Figure 4.3 shows the considered landmark on Rviz.

Figure 4.3

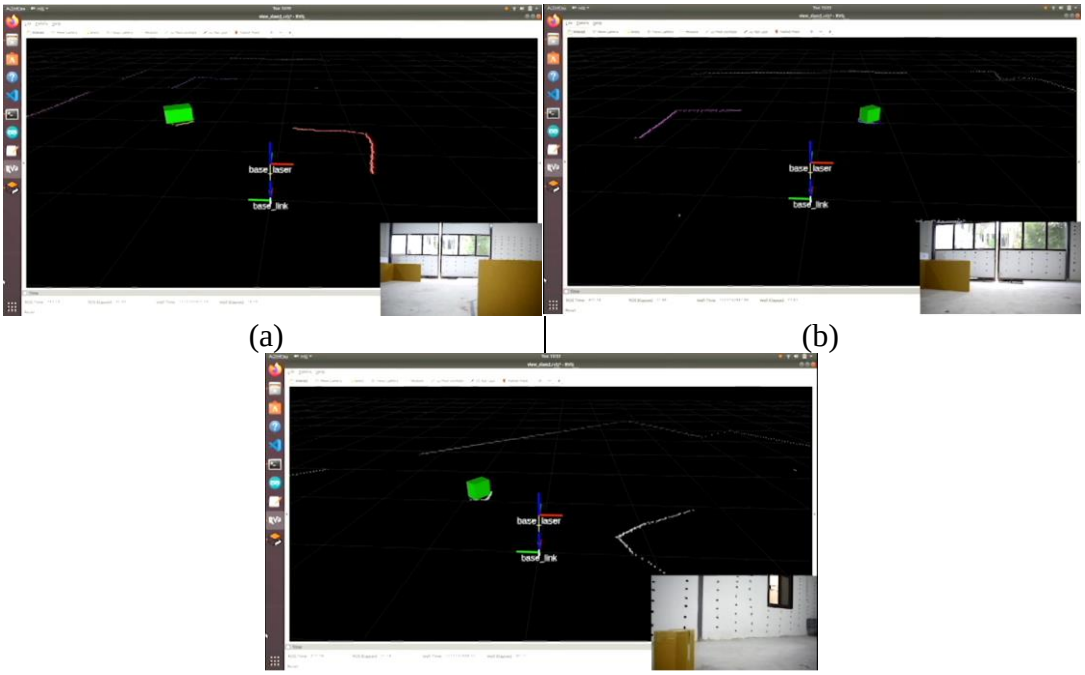
Laser Scanner Output on Rviz with Landmark Consideration



The landmarks are shown in the visualization tool while the laser scanner is roaming through the environment. It is observed that failures of landmark detection exist at a certain angle which may due to the number of point clusters. Moreover, the landmarks being considered is not projected in the simulator all the time.

Figure 4.4

Landmark One, Two, and Three on Rviz



Note. These images show the three landmarks labeled as (a), (b), and (c) for landmarks 1, 2, and 3 respectively.

4.1.2 Comparison of Ground Truth and Robot Position in EKF

The distance data from the ground truth and the calculated robot position are obtained from the actual measurement in the environment using a tape measure and from the ros command *rostopic echo /slam_path*. In detail, the ground truth represents the actual measurement from the environment. Each specified location numbered from Point 1 to 10 in Table 4.1 is noted and these points are compared to the predicted measurement coming from the calculated state using the EKF. The information or data coming from the sensors are fed to the system which will compute for the estimated state of the robot.

There are 10 arbitrary points between the starting position and ending position. The calculation of the RMSE in this research considered two conditions.

The first calculation uses the x and y coordinates between the actual and predicted measurement of the robot under consideration. By using Equation (3.24), the RMSE in the x and y direction yields to 0.2981 m and 0.1589 m, respectively (See Table 4.1). An observation is made with the RMSE results since the error in the x-axis is close to the error in the y-axis. Every time the driving motor moves the chains, a drag is affecting the steering chain mechanism. Moreover, it is also the case when the steering motor move. Thus, this results to a deviation of the actual movement of the robot and yields to a different value. Another thing to consider is the cumulative error presented in the values of the RSME of 0.2981 m and 0.1589 m. For longer paths, the errors will increase over time.

Table 4.1

RMSE Calculation for x and y Measurement Values

Point No.	**Ground Truth (m)**	**Predicted Measurement (m)**	**Residuals (x-axis) (m)**	**Residuals (y-axis) (m)**	**Square (x-axis) (m^2)**	**Square (y-axis) (m^2)**
A	(0.00, 0.00)	(0.00, 0.00)	0	0	0	0
1	(0.00, 0.58)	(0.02, 0.73)	0.02	0.15	0.0004	0.0225
2	(0.00, 1.03)	(-0.05, 0.98)	-0.05	-0.05	0.0025	0.0025
3	(0.00, 1.58)	(-0.20, 1.64)	-0.2	0.06	0.04	0.0036
4	(0.00, 2.02)	(-0.29, 2.04)	-0.29	0.02	0.0841	0.0004
5	(0.00, 2.59)	(-0.37, 2.73)	-0.37	0.14	0.1369	0.0196
6	(0.00, 3.00)	(-0.33, 3.18)	-0.33	0.18	0.1089	0.0324
7	(0.00, 3.59)	(-0.18, 3.74)	-0.18	0.15	0.0324	0.0225
8	(0.00, 4.00)	(-0.10, 4.17)	-0.1	0.17	0.01	0.0289
9	(1.10, 4.00)	(0.57, .4.20)	-0.53	0.2	0.2809	0.04
10	(1.52, 4.00)	(1.07, 4.24)	-0.45	0.24	0.2025	0.0576
B	(2.01, 4.00)	(1.60, 4.27)	-0.41	0.27	0.1681	0.0729
			Sum Square Error [SSE] (m^2)		1.0667	0.3029
			Mean of Sum Square Error (m^2)		0.0889	0.0252
			Root Mean Square Error [RMSE] (m)		0.2981	0.1589

***A** – starting point
B – final position

Table 4.2

RMSE Calculation for the Overall Travelled Distance

Point No.	Ground Truth x-axis (m)	Predicted Measurement x-axis (m)	Ground Truth y-axis (m)	Predicted Measurement y-axis (m)	Ground Truth Distance Travelled (m)	Predicted Distance Travelled (m)	Residuals (m)	Square (m^2)
A	0	0	0	0	0	0	0	0
1	0	0.02	0.58	0.73	0.58	0.7302	0.1502	0.0225
2	0	-0.05	1.03	0.98	1.03	0.9812	-0.0487	0.0023
3	0	-0.2	1.58	1.64	1.58	1.6521	0.0721	0.0052
4	0	-0.29	2.02	2.04	2.02	2.0605	0.0405	0.0016
5	0	-0.37	2.59	2.73	2.59	2.7549	0.1649	0.0272
6	0	-0.33	3	3.18	3	3.1970	0.1970	0.0388
7	0	-0.18	3.59	3.74	3.59	3.7443	0.1543	0.0238
8	0	-0.1	4	4.17	4	4.1711	0.1711	0.0293
9	1.1	0.57	4	4.2	4.1484	4.2385	0.0900	0.0081
10	1.52	1.07	4	4.24	4.2790	4.3729	0.0938	0.0088
B	2.01	1.6	4	4.27	4.4766	4.5599	0.0833	0.0069
					Sum Square Error [SSE] (m^2)			0.1748
					Mean of Sum Square Error (m^2)			0.0145
					Root Mean Square Error [RMSE] (m)			0.1207

***A –** starting point
B – final position

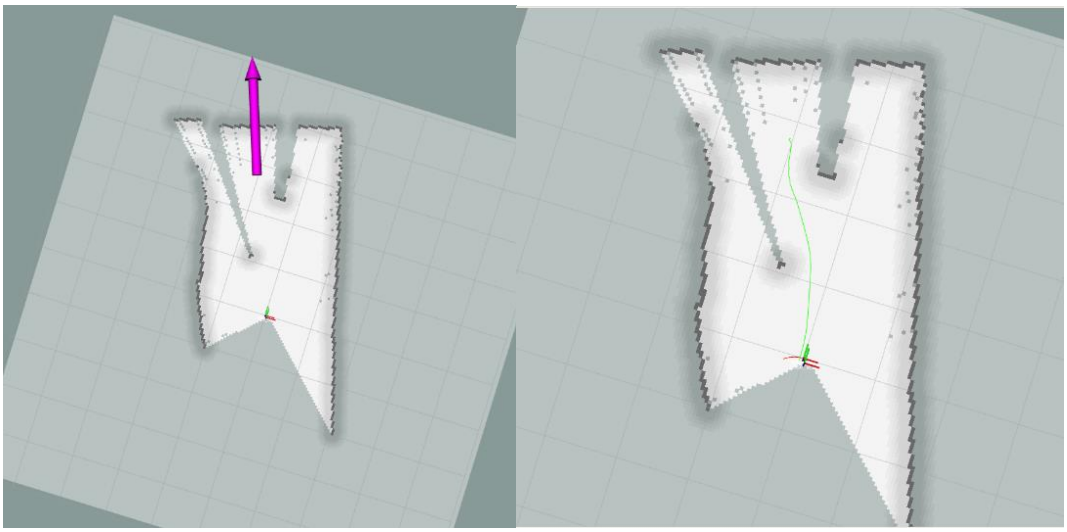
Another condition for the RMSE calculation is the total distance. Solving the RMSE shows that the root mean square error yields to 0.1207 m (See Table 4.2). Looking into the theoretical distance traveled by the robot of 4.48 m, the error is 2.70%.

4.1.3 Application of Obstacle Avoidance

To show the application, a mobile robot navigation is implemented in the proposed system. Figure 4.1.3.1 show the calculated path of the robot with the two obstacles in the visualization tool.

Figure 4.5

Obstacle Avoidance Mobile Robot Application



The pink arrow represents the location and orientation of the mobile robot in the environment where the green line shows the path of the robot to the goal. Figure 4.5 shows the travel of the robot to the final destination given with two obstacles. Additionally, the screenshot of the robot travelling to the destination is shown in Figures 4.6 and 4.7.

Figure 4.6

Sample Screenshot of the Mobile Robot performing Obstacle Avoidance

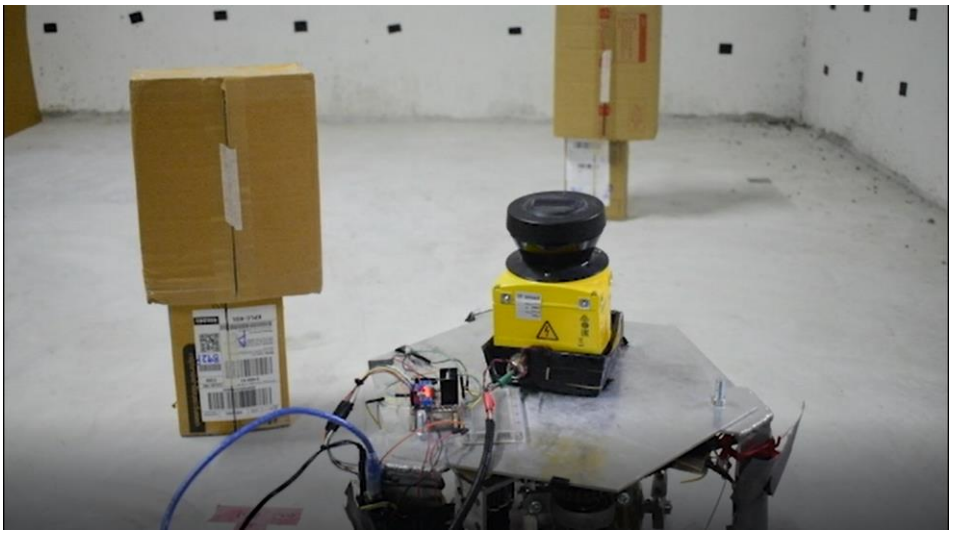
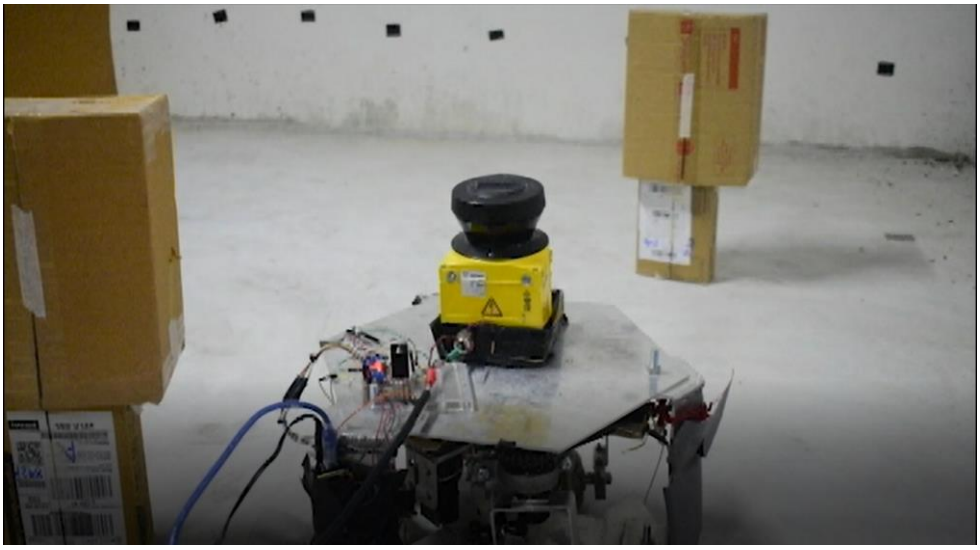


Figure 4.7

Screenshot Example of the Mobile Robot performing Obstacle Avoidance



CHAPTER 5
CONCLUSION AND RECOMMENDATION

5.1 Conclusion

This research proposed a synchronous-drive design mobile robot using odometry and laser scanner to implement the EKF-SLAM. An addition of obstacle avoidance is covered in this study to show a potential application of the mobile robot. To summarize, three tasks were accomplished in this research. First is the design of the synchronous drive robot where its steering and driving mechanisms are independently moving. This results to an easier control compare to other drive configuration. Another point to consider in this task is that the mobile robot design is mechanically complex. Slightest misalignment from the sprockets results to loosen chains. Second task is implementation of EKF-SLAM including the detection from the observed landmarks. Problems exist in the landmark detection due to its distance from the assumed wall or the angle how the laser scanner sensed that object. Lastly is the incorporation of the obstacle avoidance in the proposed robot. This shows that potential applications of the proposed robot are possible.

Based from the experiment, the mean square error between the actual and calculated measurement yields to 0.2981 m and 0.1589 m for the x and y axis error, respectively. When considering the total distance travelled by the robot, a mean square error of 0.1207 m is evident. Moreover, the drag factor from the other chain mechanism initiates a slight change in the other mechanism resulting to an error. Error also exists due to the cumulative error using odometry. This shows that the drive configuration is mechanically complex. Also, the theoretical distance travelled by the robot is 4.48 m which yielded to an error of 2.70%.

5.2 Recommendation

This research recommends to explore the possibility of 3-dimensional implementation which includes the visual approach of SLAM. Incorporating cameras or kinetic sensor provides a depth information from the environment. Another recommendation is the introduction of hybrid filters such as Radial Basis Function (RBF) or Multilayer

Perception (MLP) with EKF for investigating the learning properties of neural networks.

REFERENCES

Agrawal, P., Sahai, S., Gautam, P., Kelkar, S. S., & D, M. R. (2021). Designing Variable Ackerman Steering Geometry for Formula Student Race Car. *International Journal of Analytical, Experimental and Finite Element Analysis*, *8*(1), 1–11. http://ischolar.info/index.php/ijaefea/article/view/209004

Aulinas, J., Petillot, Y., Salvi, J., & Lladó, X. (2008). The SLAM problem: A survey. *Artificial Intelligence Research and Development*, 363–371. https://doi.org/10.3233/978-1-58603-925-7-363

Borenstein, J., & Feng, L. (1996). *Measurement and correction of systematic odometry errors in mobile robots*. *12*(6), 869–880. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=544770

Bräunl, T. (2008). Embedded robotics: Mobile robot design and applications with embedded systems. In *Springer Science & Business Media* (2nd Editio). Springer-Verlag Berlin6 Heidelberg. https://doi.org/10.1007/978-3-662-05099-6

Bruzzone, L., Baggetta, M., Nodehi, S. E., Bilancia, P., & Fanghella, P. (2021). Functional design of a hybrid leg-wheel-track ground mobile robot. *Machines*, *9*(1), 1–11. https://doi.org/10.3390/machines9010010

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., & Leonard, J. J. (2016). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, *32*(6), 1309–1332. https://doi.org/10.1109/TRO.2016.2624754

Chan, T. H., Hesse, H., & Ho, S. G. (2021). LiDAR-Based 3D SLAM for Indoor Mapping. *2021 7th International Conference on Control, Automation and Robotics (ICCAR)*, 285–289. https://doi.org/10.1109/ICCAR52225.2021.9463503

Chatterjee, A., Ray, O., Chatterjee, A., & Rakshit, A. (2011). Development of a real-life EKF based SLAM system for mobile robots employing vision sensing. *Expert Systems with Applications*, *38*(7), 8266–8274. https://doi.org/10.1016/j.eswa.2011.01.007

Chen, C. H., Lin, C. J., Jeng, S. Y., Lin, H. Y., & Yu, C. Y. (2021). Using ultrasonic sensors and a knowledge-based neural fuzzy controller for mobile robot navigation control. *Electronics (Switzerland)*, *10*(4), 1–22. https://doi.org/10.3390/electronics10040466

Choi, Y.-H., Lee, T.-K., & Oh, S.-Y. (2008). A line feature based SLAM with low grade range sensors using geometric constraints and active exploration for mobile robot. *Autonomous Robots*, *24*(1), 13–27. https://doi.org/10.1007/s10514-007-9050-y

Chung, W., & Iagnemma, K. (2016). Wheeled robots. In B. Siciliano & O. Khatib (Eds.), *Springer Handbook of Robotics* (pp. 575–593). Springer-Verlag Berlin Heidelberg. https://doi.org/10.1007/978-3-319-32552-1

Doucet, A., Freitas, N. De, Murphy, K. P., & Russell, S. (2013). Rao-Blackwellised particle filtering for dynamic bayesian networks. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, 176–183. https://arxiv.org/ftp/arxiv/papers/1301/1301.3853.pdf

Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping (SLAM): Part II. *IEEE Robotics & Automation Magazine*, *13*(3), 108–117. https://doi.org/10.1109/MRA.2006.1678144

Gerkey, B. (2020). *gmapping - ROS Wiki*. http://wiki.ros.org/gmapping

Goel, P., Roumeliotis, S. I., & Sukhatme, G. S. (n.d.). Robot localization using relative and absolute position estimates. *Proc. 1999 IEEE. In RSJ International Conference on Intelligent Robots and Systems*, 17–21.

Goris, K. (2005). *Autonomous mobile robot mechanical design* [Vrije Universiteit Brussel]. http://mech.vub.ac.be/multibody/final_works/ThesisKristofGoris.pdf

Grisetti, G., Rainer, K., Stachniss, C., & Burgard, W. (2010). A tutorial on graph-based SLAM. *IEEE Intelligent Transportation Systems Magazine*, *2*(4), 31–43. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5681215

Haykin, S. (Ed.). (2004). *Kalman filtering and neural networks* (Vol. 47). John Wiley & Sons, Inc. https://doi.org/10.1530/jrf.0.0320129

Jones, J., Seiger, B., & Flynn, A. (1998). Mobile robots: Inspiration to implementation. In *Leonardo* (2nd Editio). CRC Press. https://doi.org/10.2307/1576020

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *ASME. J. Basic Eng.*, *82*(1), 35–45. https://doi.org/10.1115/1.3662552

Kim, D. (2019). *Commercializing Localization for Automated Driving: Absolute vs. Relative*. https://static1.squarespace.com/static/5869dbb6ff7c505afd38a083/t/5d3f574d2928800001c7b4b1/1564432206257/Commercializing+Localization+for+Automated+Driving+-+Absolute+vs.+Relative.pdf

Kohlbrecher, S., & Meyer, J. (2020). *hector_slam - ROS Wiki*.

http://wiki.ros.org/hector_slam

Lei, S., & Li, Z. (2012). SLAM and navigation of a mobile robot for indoor environments. In F. Sun, T. Li, & H. Li (Eds.), *Proceedings of the Seventh International Conference on Intelligent Systems and Knowledge Engineering* (Vol. 213, pp. 151–162). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-37829-4

Lindholm, R., & Palsson, C.-J. (2015). *Simultaneous localization and mapping for vehicle localization using LIDAR sensors* [Chalmers University of Technology]. https://odr.chalmers.se/bitstream/20.500.12380/219126/1/219126.pdf

Lu, F., & Milios, E. (1997). Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, *4*(4), 333–349. https://doi.org/10.1023/A:1008854305733

Lv, J., Kobayashi, Y., Emaru, T., & Ravankar, A. A. (2015). Indoor slope and edge detection by using two-dimensional EKF-SLAM with orthogonal assumption. *International Journal of Advanced Robotic Systems*, *12*(4), 1–7. https://doi.org/10.5772/60407

Lv, J., Kobayashi, Y., Ravankar, A. A., & Emaru, T. (2014). Straight line segments extraction and EKF-SLAM in indoor environment. *Journal of Automation and Control Engineering*, *2*(3), 270–276. https://doi.org/10.12720/joace.2.3.270-276

Maliha Monsur. (2021). *GPS/WI-FI Integration for Challenging Environments using Raw GNSS Data*. http://dspace.ucuenca.edu.ec/bitstream/123456789/35612/1/Trabajo de Titulacion.pdf%0Ahttps://educacion.gob.ec/wp-content/uploads/downloads/2019/01/GUIA-METODOLOGICA-EF.pdf

Manh, H. H. (2020). *Encoders*.

Marin-Reyes, H., & Tokhi, M. O. (2010). Control system adaptation of a synchro drive mobile robot for target approximation. *Mobile Robotics: Solutions and Challenges - Proceedings of the 12th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines, CLAWAR 2009*, 1063–1070. https://doi.org/10.1142/9789814291279_0130

Miller, D. P. (1986). *Low error path planning for a synchro-drive mobile robot*. https://vtechworks.lib.vt.edu/bitstream/handle/10919/19768/TR-86-28.pdf?sequence=3

Montella, C. (2014). The kalman filter and related algorithms: A literature review.

Research Gate, 1–17.

Murphy, K. P. (1999). Bayesian map learning in dynamic environments. *Advances in Neural Information Processing Systems*, 1015–1021. https://neuro.bstu.by/ai/To-dom/My_research/Papers-1/Spiking-N/Murphy.pdf

Nguyen, H. K. (2014). *Integrated approach to simultaneous localization and mapping with path planning algorithms for indoor mobile robots* [Chulalongkorn University]. https://www.car.chula.ac.th/display7.php?bib=b2139968

Nguyen, V., Harati, A., Martinelli, A., Siegwart, R., & Tomatis, N. (2006). Orthogonal SLAM: A step toward lightweight indoor autonomous navigation. *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5007–5012. https://doi.org/10.1109/IROS.2006.282527

Riisgaard, S., & Blas, M. R. (2003). SLAM for dummies: A tutorial approach to simultaneous localization and Mmapping. *Small Flying Drones: Applications for Geographic Observation*, *22*, 1–127. https://doi.org/10.1007/978-3-319-66577-1

Saputra, R. P. (2015). *Implementation 2D EKF-SLAM for wheeled mobile robot* [University of New South Wales, Australia]. https://arxiv.org/ftp/arxiv/papers/1905/1905.06529.pdf

SICK Sensor Intelligence. (2015). *Telegram listing CMS: S3000 Expert/Anti Collision, S300 Expert*. https://cdn.sick.com/media/docs/1/91/891/telegram_listing_s3000_expert_anti_collision_s300_expert_de_en_im0022891.pdf

Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). *Introduction to autonomous mobile robots* (2nd Editio). The MIT Press. https://doi.org/10.1109/ROBOT.2010.5509725

Singh, A. (2018). *An intro to Kalman Filters for Autonomous Vehicles*. https://towardsdatascience.com/an-intro-to-kalman-filters-for-autonomous-vehicles-f43dd2e2004b

Smith, R. C., & Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research*, *5*(4), 56–68. https://doi.org/10.1177/027836498600500404

Takahashi, M., Suzuki, T., Shitamoto, H., Moriguchi, T., & Yoshida, K. (2010). Developing a mobile robot for transport applications in the hospital domain. *Robotics and Autonomous Systems*, *58*(7), 889–899. https://doi.org/10.1016/j.robot.2010.03.010

Tătar, M. O., & Cirebea, C. (2018). Modular reconfigurable robot. In I. Doroftei, C. Oprisan, D. Pisla, & E. C. Lovasz (Eds.), *Proceedings of The 12th IFToMM International Symposium on Science of Mechanisms and Machines* (Vol. 57, pp. 291–299). Springer International Publishing. https://doi.org/10.1007/978-3-030-20131-9

Tătar, M. O., Haiduc, F., & Mândru, D. (2015). Design of a synchro-drive omnidirectional mini-robot. *Solid State Phenomena*, *220*, 161–167. https://doi.org/10.4028/www.scientific.net/SSP.220-221.161

Thomas Genevois, & Zielinska, T. (2014). A simple and efficient implementation of EKF-based SLAM relying on laser scanner in complex indoor environment. *Journal of Automation Mobile Robotics and Intelligent Systems*, *8*, 58–67. https://doi.org/10.14313/JAMRIS

Thrun, S. (2003). *Robotic mapping: A survey*. 1–35. https://doi.org/10.1126/science.298.5594.699f

Yan, H. W. (2019). *Sensor fusion based speed and heading control of an intelligent vehicle*. Asian Institute of Technology.

APPENDICES

APPENDIX A
ARDUINO PIN CONNECTIONS

Table A1 shows the pin connection between the motor encoder, motor driver, and microcontroller.

Table A1

Pin Connection of Motor Encoders and Motor Driver to Arduino Mega 2560

Parts	Arduino Pin
Motor Driver	
Input 1 (M1)	Digital 6
Input 2 (M1)	Digital 12
Input 3 (M2)	Digital 11
Input 4 (M2)	Digital 10
Enable A (M1)	Digital 7
Enable B (M2)	Digital 11
Encoder	
Pin A (D)	Digital 18
Pin B (D)	Digital 19
Pin C (S)	Digital 2
Pin D (S)	Digital 3

*Abbreviation in the parenthesis above are defined as follows. **M1 –** driving motor**,** **M2 –** steering motor, **D** – encoder for driving motor, and **S** – encoder for steering motor.

APPENDIX B

LASER SCANNER PYTHON CODE

Table B1

Python Code for Sick S300 Laser Scanner

```
# File Name:  SickS300Scanner.py
# Description: Publishes laser scanner messages using the Python language through ROS

#!/usr/bin/env python
from __future__ import division

import rospy
import serial
import os
import binascii
import time
import re
import math
from serial import Serial
from sensor_msgs.msg import LaserScan
import sensor_msgs.msg


rospy.init_node('laser_scan_publisher')

scan_pub = rospy.Publisher('scan', LaserScan, queue_size = 50)

serialPort = serial.Serial(port='/dev/ttyUSB0',
      baudrate = 500000,
      parity=serial.PARITY_NONE,
      stopbits=serial.STOPBITS_ONE,
      bytesize=serial.EIGHTBITS,
      timeout=None)

def __function_SPLIT(stringToSplit, countPerSplit):
   return [stringToSplit[i:i+countPerSplit] for i in range(0, len(stringToSplit), countPerSplit)]

__GLOBAL_DataBlocksToFind = "00000229ff07"
__GLOBAL_RequiredDataLength = 2212
__GLOBAL_RequiredDataLengthAfter = 2200
__GLOBAL_MeasurementDataBlockLength = 2160
__GLOBAL_MeasurementHeader = "bbbb1111"
__PRINT_FinalScanPrint = []

__DEBUG_ScanID = 0
__DEBUG_IndexErrorCount = 0

while not rospy.is_shutdown():
   current_time = rospy.Time.now()

   scan = LaserScan()

   #print("Scan Number %i" %  __DEBUG_ScanID)
```

```
    __SERIAL_BaseHex = serialPort.readline()
    __SERIAL_BaseString = binascii.hexlify(__SERIAL_BaseHex)

    #print(__SERIAL_BaseString)
    #break

    __SERIAL_StartingDataBlockIndeces = []

    __THIS_ArraySearchIndeces = []
    for __THIS_SearchIndex in re.finditer(__GLOBAL_DataBlocksToFind, __SERIAL_BaseString):
        __THIS_ArraySearchIndeces.append((__THIS_SearchIndex.start(), __THIS_SearchIndex.end()));

    __DEBUG_NextFirstIndex = ""
    __DEBUG_status = ""
    for __THIS_ArrayCounter in range(len(__THIS_ArraySearchIndeces)):
        __DEBUG_CurrentArray = __THIS_ArraySearchIndeces[__THIS_ArrayCounter]
        __DEBUG_CurrentStartIndex = __THIS_ArraySearchIndeces[__THIS_ArrayCounter][0]
        __DEBUG_CurrentLastIndex = __THIS_ArraySearchIndeces[__THIS_ArrayCounter][1]

        # DEBUG PRINTS
        #print("Scan Lenght", len(__SERIAL_BaseString))
        #print("Current Array", __DEBUG_CurrentArray)
        #print("Current Start Index", __DEBUG_CurrentStartIndex)
        #print("Current Last Index", __DEBUG_CurrentLastIndex)
        #print(__SERIAL_BaseString[(__DEBUG_CurrentLastIndex-4):__DEBUG_CurrentLastIndex])

        try:
            __DEBUG_NextFirstIndex = __THIS_ArraySearchIndeces[__THIS_ArrayCounter+1][0]
            #print("Next First Index", __DEBUG_NextFirstIndex)
        except IndexError:
            __DEBUG_NextFirstIndex = len(__SERIAL_BaseString)
            #print("Last Index", __DEBUG_NextFirstIndex)
            #print("::> This is the last")

        __SERIAL_CalculateDataBlockStart = __DEBUG_CurrentLastIndex
        __SERIAL_CalculateDataBlockEnd = __DEBUG_NextFirstIndex

        #print("Data Block Start", __SERIAL_CalculateDataBlockStart)
        #print("Data Block End", __SERIAL_CalculateDataBlockEnd)
        #print("Difference Amount", (__SERIAL_CalculateDataBlockEnd-
__SERIAL_CalculateDataBlockStart))
        #print(len(__SERIAL_BaseString[__DEBUG_CurrentLastIndex:__DEBUG_NextFirstIndex]))

        if (__SERIAL_CalculateDataBlockEnd-__SERIAL_CalculateDataBlockStart) >=
__GLOBAL_RequiredDataLengthAfter:
            #print("::> There is enough data for us to use.")
            #print("Accepted Difference Amount", (__SERIAL_CalculateDataBlockEnd-
__SERIAL_CalculateDataBlockStart))
            __SERIAL_StartingDataBlockIndeces.append(__SERIAL_CalculateDataBlockStart)
        #else:
        #    print("::> Not enought data, skipped")

    #print("Array Search Results Count: %i" % len(__THIS_ArraySearchIndeces))

    __DEBUG_ForceStopper = False
    try:
        __SERIAL_FirstStartDataBlockIndex = __SERIAL_StartingDataBlockIndeces[0]
        __SERIAL_BaseDataBlocks = __SERIAL_BaseString[__SERIAL_FirstStartDataBlockIndex:]
```

```
        __SERIAL_PreparedDataBlockIndex_START =
__SERIAL_BaseDataBlocks.find(__GLOBAL_MeasurementHeader)
        __SERIAL_PreparedDataBlock =
__SERIAL_BaseDataBlocks[__SERIAL_PreparedDataBlockIndex_START:__SERIAL_PreparedData
BlockIndex_START+(__GLOBAL_MeasurementDataBlockLength+8)]
        #print(len(__SERIAL_PreparedDataBlock))
        __SERIAL_PreparedDataBlock = __function_SPLIT(__SERIAL_PreparedDataBlock, 4)
        __SERIAL_PreparedDataBlock.remove("bbbb")
        __SERIAL_PreparedDataBlock.remove("1111")

        __DEBUG_SplitBaseDataBlocks = __function_SPLIT(__SERIAL_BaseDataBlocks, 4)
        __DEBUG_SplitBaseString = __function_SPLIT(__SERIAL_BaseString, 4)
        #print(len(__DEBUG_SplitBaseString))
        #print(__DEBUG_SplitBaseString)
        #print(len(__DEBUG_SplitBaseDataBlocks))
        #print(__DEBUG_SplitBaseDataBlocks)

        __SERIAL_PreparedDataBlock_BasePOST = []
        __SERIAL_PreparedDataBlock_BinaryPOST = []
        __SERIAL_PreparedDataBlock_DecimalPOST = []

        for dataBlock in __SERIAL_PreparedDataBlock:
            __TEMP_datablock = __function_SPLIT(dataBlock, 2)
            #print("Before: %s" % ", ".join(__TEMP_datablock))
            __TEMP_datablock[0], __TEMP_datablock[1] = __TEMP_datablock[1],
__TEMP_datablock[0]
            #print("After: %s" % ", ".join(__TEMP_datablock))
            __TEMP_datablock = "".join(__TEMP_datablock)
            __TEMP_dataBlock_binary = bin(int(__TEMP_datablock, 16))[2:].zfill(16)
            __TEMP_dataBlock_binary = __TEMP_dataBlock_binary[4:]
            __TEMP_dataBlock_decimal = int(__TEMP_dataBlock_binary, 2)
            __TEMP_dataBlock_decimal = __TEMP_dataBlock_decimal / 100.0
            #print("Datablock: %s" % __TEMP_datablock)
            #print("Binary: %s" % __TEMP_dataBlock_binary)
            #print("Decimal: %i" % __TEMP_dataBlock_decimal)

            __SERIAL_PreparedDataBlock_BasePOST.append(__TEMP_datablock)
            __SERIAL_PreparedDataBlock_BinaryPOST.append(__TEMP_dataBlock_binary)
            __SERIAL_PreparedDataBlock_DecimalPOST.append(__TEMP_dataBlock_decimal)


        scan.header.stamp = current_time
        scan.header.frame_id = 'base_laser'
        scan.angle_min = -0.785398
        scan.angle_max = 3.92699
        scan.angle_increment = 0.0087222222
        scan.time_increment = (1 / 40) / (541)
        scan.range_min = 0.0
        scan.range_max = 100.0


        scan.ranges = __SERIAL_PreparedDataBlock_DecimalPOST

        scan_pub.publish(scan)
        #print("Before: %s" % ", ".join(__SERIAL_PreparedDataBlock))
        #print("After : %s" % ", ".join(__SERIAL_PreparedDataBlock_POST))
        #print("Data Block Length: %i" % len(__SERIAL_PreparedDataBlock))
        #print(__SERIAL_PreparedDataBlock)
        #print(__SERIAL_PreparedDataBlock_BasePOST)
        #print(__SERIAL_PreparedDataBlock_BinaryPOST)
```

```
    #print(__SERIAL_PreparedDataBlock_DecimalPOST)
    __DEBUG_ForceStopper = True
except IndexError:
    __DEBUG_IndexErrorCount = __DEBUG_IndexErrorCount + 1

__DEBUG_ScanID = __DEBUG_ScanID + 1

#if __DEBUG_ForceStopper:
#    break
```