UNIVERSITY OF MUMBAI

A PROJECT REPORT ON

Simultaneous Navigator for Autonomous Identification and Localization Robot

Submitted in partial fulfillment of the requirements

Of the degree of

Bachelor of Electronics Engineering

By

RAJAS JOSHI

AMBASHRI PURKAYASTHA

DARPAN BHAIYA

SHRUTI PATIL

Under the Guidance of

Dr. ANJALI DESHPANDE
Department of Electronics Engineering

Prof. AKHIL MASURKAR
Department of Electronics Engineering

Vidyalankar Institute of Technology

Wadala (E), Mumbai 400 037

CERTIFICATE OF APPROVAL

This is to certify that

RAJAS JOSHI

AMBASHRI PURKAYASTHA

DARPAN BHAIYA

SHRUTI PATIL

have successfully carried out Project work entitled

Simultaneous Navigator for Autonomous Identification and Localization Robot

In partial fulfillment of B.E. Degree in Electronics Engineering.

As laid down by the University of Mumbai during the academic year

2020-2021

Dr. ANJALI DESHPANDE　　　　　　　　Prof. AKHIL MASURKAR

Examiner　　　　　　Head of Department　　　　　　Principal

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Name of student	Roll No.	Signature
1. RAJAS JOSHI	17103A0015	
2. AMBASHRI PURKAYASTHA	17103A0060	
3. DARPAN BHAIYA	17103A0063	
4. SHRUTI PATIL	17103A0051	

Date:

ABSTRACT

The 20th century featured a boom in the development of industrial robots. Through the rest of the century, robots changed the structure of society and allowed for safer conditions for labor. The industry has benefited drastically from the expanse of a robotic workforce. Observing evolution, it has become important to introduce these robots as a replacement for humans in a menacing territory. Automated machines have taken over the duties of hazardous jobs from humans, allowing greater productivity. Identifying and analyzing the habitat in which you intend your robot to navigate is among the most crucial steps of getting started with the research. The reason is that a robot's ability to perform its job safely and securely is heavily dependent upon its understanding of its surroundings. The robots learn both familiar and unspecified circumstances with the help of sensors. This navigation can be either indoor or outdoor depending on the capability of the robot to dodge the dynamic or static obstacles in the path. The potential to choose the shortest track is another factor considered in an intelligent mobile device. Numerous researches have been carried out by adopting Artificial Intelligence techniques to improve the performance of the mobile robot navigation in terms of the accuracy in avoiding obstacles, the shortest path traveled and total time consumed. This kind of a system has been proposed to be implemented with the help of a 2D real-time terrain mapping LIDAR sensor node and a wheel odometry node to which the robot is connected and uses the data to acquire its position along with the position of the dynamic obstacles and the path to be followed thus moves autonomously in the environment.

Table of contents

CHAPTER 1

Introduction

Throughout history, humans have been used to scout for new territories, enemy lines, and potentially dangerous landscapes that could collapse or bring about disaster onto a larger group. Although this is done to assure that more people are not inadvertently hurt, this poses a huge mortality risk to the people who go out to scout these areas, as any wrong turn could result in a fatality, away from areas of immediate medical attention.

We aim to deal with this issue by proposing an automated, fully autonomous bot that could be launched into a hostile/unknown environment, to map both- the environment's boundaries and also the location of any objects present in the said environment and communicate with a remote user to provide them with real-time data sets. A study of the nature of the obstacles present could provide the user with insight into the complexity and temperament of the landscape, and hence aid in decision making, and planning.

The SNAIL Bot (Simultaneous Navigator for Autonomous Identification and Localization Robot) is designed to work in remote areas, with little to no human activity. It uses a LiDAR to find the boundaries of the environment, and the odometry data from the encoder motors are used to localize the bot and all the obstacles present in the range of the LiDAR.

This data from the sensor is then sent raw, to the user at the other end of the system, who then processes it using various SLAM (Simultaneous Localisation and Mapping) algorithms like gmapping, Hector SLAM, et cetera, using odometry data for computation of pose estimation and movement of the sensor.

The visual data of the environment is captured using a YD LIDAR X4, and the live position of the bot is calculated using a 75RPM Encoder-Motor. This data is transmitted wirelessly to the controller and remote user, where the data is processed and the map is recreated using the point-cloud data that is converted from the laser-scan data and transmitted.
The methods used for these are as follows:

- SLAM algorithms for developing the grid and extracting the laser-scan data.
- Odometry algorithms to calculate the position and movement of the bot.
- URDF modeling for the development of the bot structure.

CHAPTER 2

1. Motivation

Simultaneous Localisation and Mapping (SLAM) is the central information engineering problem in mobile robotics research, simple to state but challenging to solve: ``how can a mobile robot operate in an unknown environment, using only onboard sensors to simultaneously navigate and build a map of its workspace?". The difficulty of SLAM lies in a robot's interactions with the real world.

Solutions to SLAM are of core importance in providing mobile robots with the ability to operate with real autonomy. There is now a strong incentive to use SLAM in whole new sectors such as the number of service robots in commercial use is substantial and growing fast in warehouse management, manufacturing, hospital courier systems, and security systems. Low-cost consumer-level robots for cleaning, home-care, and entertainment are now also becoming a reality. Without SLAM, such robots require costly and inconvenient installation procedures and are intolerant of changes in their workspace. SLAM offers a viable way to make mobile robotics an all-pervasive and truly useful technology.

2. Objective

- Indoor mapping using SLAM algorithms
- Autonomous navigation of the bot after mapping of the outer boundaries of the room.
- Wheel odometry to calculate the distance and direction traveled by the bot
- Remote data processing for the development of the map from the data cloud

CHAPTER 3

1. Literature Review

1.1. Comparative study of various SLAM algorithms

A study of different SLAM algorithms was conducted to evaluate the efficiency and accuracy of the data for the appropriate solutions required.

1. **Hector SLAM:** It is an algorithm designed specifically for range sensors, that combines the data from a planar (2D) scan matching procedure with measurements provided by an external attitude estimator (i.e. an IMU) to solve the SLAM problem[1]. It is an open-source implementation that is based on using a laser scan to build a grid map of the surroundings.

2. **Gmapping:** The Gmapping algorithm was proposed to solve grid-based SLAM problems. It requires odometry information and the sensor's observations to estimate the trajectory of the robot and both, the incremental movement of the sensor and how the robot is localized within the map. Gmapping implementation allows an accurate SLAM solution by significantly reducing the needed number of particles kept on the filter[2].

3. **Cartographer:** Cartographer is a system developed at Google[3] that provides real-time simultaneous localization and mapping. SLAM algorithms combine data from various sensors (e.g. LIDAR, IMU, and cameras) to simultaneously compute the position of the sensor and a map of the sensor's surroundings. Cartographer primarily consists of two subsystems, global SLAM, and local SLAM. Local SLAM is used to generate good submaps of the region and global SLAM is used to tie the submaps together as consistently as possible.

Our interest is in practical aspects of the algorithms, from which we establish the criteria to carry out the experimental evaluation, which are assessed using datasets collected from the literature depicting realistic operation scenarios for autonomous mobile robots.

1.2. Odometry data

The development of a navigation system is one of the major challenges in building a fully autonomous platform. Full autonomy requires a dependable navigation capability not only in a perfect situation with clear GPS signals but also in situations, where the GPS is unreliable. Therefore, self-contained odometry systems have attracted much attention recently[9].

Usually, the SLAM techniques that apply an odometry algorithm to obtain the pose of the moving platform were later fed into a global map optimization algorithm, i.e., loop closure, to reduce the drift as much as possible based on the history of the pose, i.e., map[9].

1.3. URDF Modelling

Unified Robotics Description Format[5], is an XML specification used in academia and industry to model multibody systems such as robotic manipulator arms for manufacturing assembly lines and animatronic robots for amusement parks. URDF is especially popular with users of ROS[6], or Robotics Operating System—a framework that offers standard support for URDF models. The URDF is the ROS standard way to represent a robot model. This format can be read from the ROS Visualizer (RViz) and any simulator integrated into ROS should be able to import this kind of model. The virtual model is not manually composed, but we auto-generate it starting from some xacro scripts to better structure our model. The URDF (Unified

Robot Description Format) has been followed to maintain the model as general-purpose as possible. We are using it on our project to represent the appearance of the robot and its intended action.

1.4. Comparative study of open-source mobile robots

- NOX Robot
- The NOX Robot works with a Kinect in fusion with Odometry data. The algorithms used in this robot is Gmapping(7)
- We modified this model to use it with Laser data, but initially, we faced the issue of frame of references, as the Kinect and the LiDAR have different frames, the latter of which was not matching with the odometry data.
- This error has been compensated for.

- Lino Robot
- The Lino Robot uses Kinect, LiDAR, and IMU in fusion with the Odometry data(8). The algorithms used in this method are:
  Hector SLAM, Gmapping, Cartographer
- Sensor fusion is compulsory in this method, and any changes made throws an error.

CHAPTER 4

1. Proposed System

The three major components of this project are depicted in the figure below.

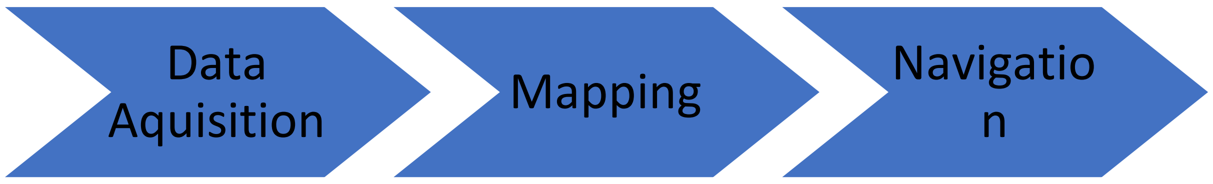


Fig. 1 Flow of major components

To build a cost-effective system capable of autonomous mapping and navigation we propose the above system. For autonomously navigating any region the most important part is to know the region, i.e. the map. Moreover, to build the map we need sensor data. We gather environment data via two types of sensors, a laser distance sensor called LIDAR and wheel encoders to gather the wheel odometry data. Both of these sensors provide us with data simultaneously using which we build the map of the environment using the method of Simultaneous Localization and Mapping (SLAM). Using this map data we can autonomously navigate the region using path planning algorithms.

To come up with this system, it is important to use a systematic approach. We have divided the system into three parts 1) Data Acquisition 2) Mapping 3) Navigation. Each of these blocks is connected. Each section is further divided into multiple parts. The diagram below presents the flow

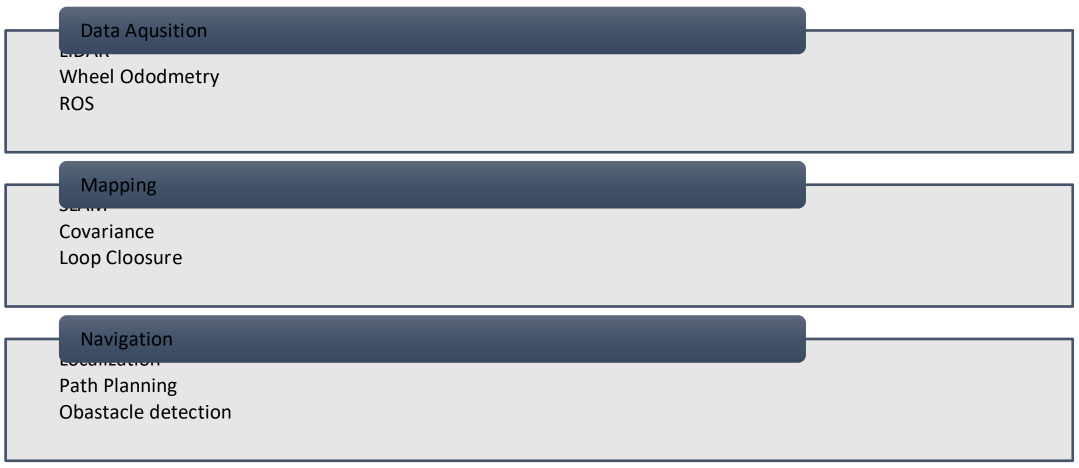


Fig. 2 Flow of Data acquisition, Mapping, and Navigation

2. Methodology for Data Acquisition

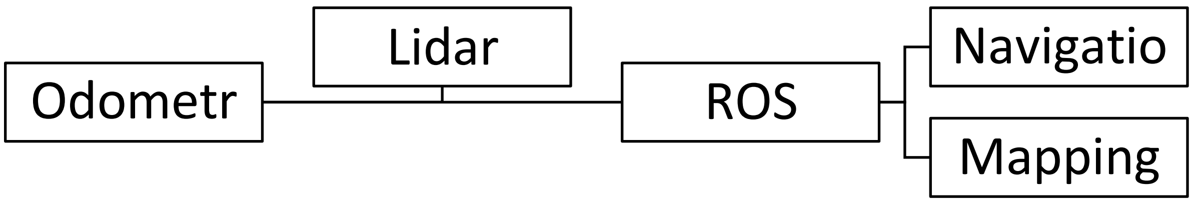


Fig. 3 Flow of Data acquisition

1. Wheel Odometry

   Wheel odometry provides linear and angular velocities of the robot using the differential drive model, which helps us to localize the robot in the environment. We gather the odometry data by using the Hall Effect encoder to calculate the distance the wheel calculates in each rotation and thus getting us the linear and angular values. Wheel odometry data is sent to ROS using Arduino as a controller via serial communication.

2. 2D LIDAR

   This sensor gives us the location of objects in our surroundings by using the method of Time of Flight. It is similar to how ultrasonic sensors work, but in place of sound waves, we use laser light. Considering this technology is quite new, thus expensive sensors, we can use RGBD cameras as an alternative, where we convert the depth data to point cloud data simulating a LIDAR. We can also use 3D LIDAR to generate point cloud data for 3D map generation.

3. ROS

   Robotics Operating System is an open-source middleware used by many in the industry to simplify robotics operation and process data. Here we use ROS to send the sensor data to the slam algorithms to generate the map and for autonomous navigation. ROS communicates with the sensors using Serial communication.

3. Mapping Algorithms

The most important thing about SLAM is that it is a probabilistic approach. So the accuracy of the map matters a lot as having high covariance can cause objects not being mapped accurately and can thus cause a lot of problems. Also, another important feature any SLAM algorithm should have is a loop closure, which means that once the robot scans a region; it should retain the data of the region so that if the robot goes to that region again, it should not map that region again.

4. Path Planning

Path planning is needed for autonomous navigation. We divide our navigation planning methods into two parts, local planner and global planner. Where local planner is used to navigating locally towards the global goal by planning path around obstacles and global planner creates a basic path towards the goal, which then needs to be adjusted locally. The important thing here is that each planner can have different path planning algorithms, which allows us to select as per our needs.

5. Design

In this section, we describe the setup designed to verify the efficacy of the developed algorithms and maps.

5.1.Setup

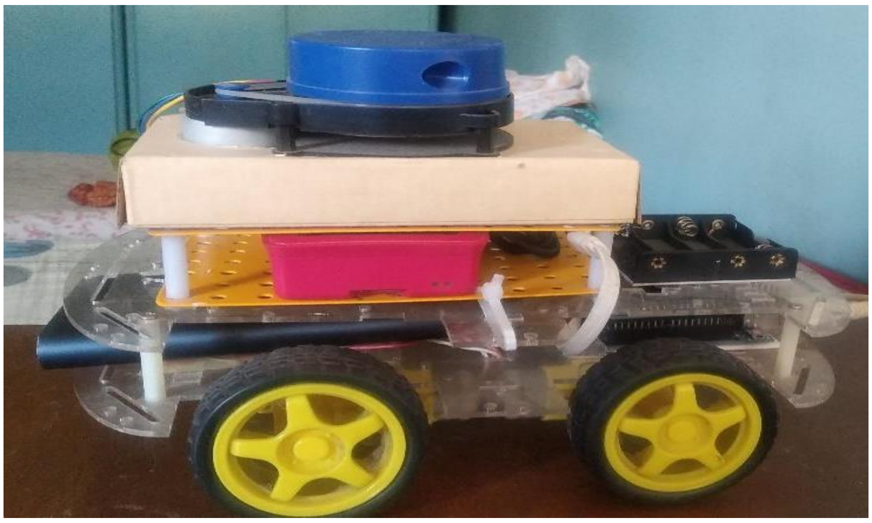


Fig. 4 Hardware Setup

Our current setup is built upon a 4wd chassis but is running a 2wd configuration. In this setup, wheel odometry is not yet deployed, as we want to test algorithms that are not dependent on wheel odometry. The setup comprises two-level modeling. First is the base level where the motors, controller boards, and batteries are placed. On the 2nd level of the chassis, we have mounted LIDAR in the center so that the given area can be mapped easily.

5.2. Component Analysis

1. Wheel Encoder

Sr. No.	**Types of Encoders**	**Working**	**Accuracy**
1.	Linear Encoder	This method uses a transducer to measure the distance between two points.	Low
2.	Mechanical Encoder	This method detects the rotational position with a variable resistor whose electrical resistance changes in proportion to the rotation angle.	Accurate
3.	Optical Encoder	This method uses a light sensor to detect whether light passes through a slit in the radial direction of a rotating disk called a code wheel attached to the motor shaft.	Extremely accurate
4.	Magnetic (Hall Effect) Encoder	This method uses a magnetic sensor to measure changes in the magnetic field distribution created by a permanent magnet attached to the motor shaft.	Extremely accurate

Table 1 - Types of wheel encoders

The above tabular data shows about types of encoders and their accuracy. Hall Effect encoder is used in this project as it has extremely accurate data of positioning, simple, compact, and durable.

We have used a 5V Quadrature Hall Effect Encoder attached to SPG30E-60K DC Geared Motor features both advantages of position sensing as well as direction sensing of the rotating shaft. This application flexibility of the encoders will allow you to control the speed of the base motor very precisely.

Specifications: -

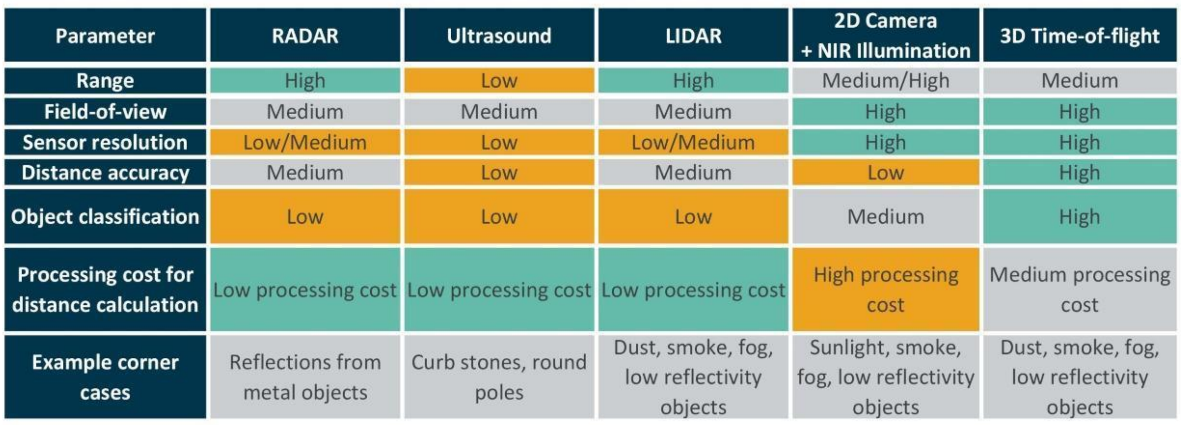
1) Rated Voltage: 12 V
2) No-load Current: 70 mA
3) Rated Current: 410 mA
4) RPM: 75 RPM
5) Rated Torque: 2.6 kg-cm
6) Shaft Diameter: 6 mm
7) Gear Ratio: 60:1

8) Encoder Voltage: 5V
9) Resolution of the encoder output:

- 3 pulses 7 pulses per rear shaft revolution, single-channel output, either channel A or B
- 420 counts per main shaft revolution, single-channel output, either channel A or B

2. Lidar

Lidar was selected because of its high accuracy and high-speed detection, which is quite important because we need to detect objects at high speed while it is mapping and navigating otherwise the robot will not be able to transverse around an obstacle close to it in enough time and would end up bumping into obstacles.



Parameter	RADAR	Ultrasound	LIDAR	2D Camera + NIR Illumination	3D Time-of-flight
Range	High	Low	High	Medium/High	Medium
Field-of-view	Medium	Medium	Medium	High	High
Sensor resolution	Low/Medium	Low	Low/Medium	High	High
Distance accuracy	Medium	Low	Medium	Low	High
Object classification	Low	Low	Low	Medium	High
Processing cost for distance calculation	Low processing cost	Low processing cost	Low processing cost	High processing cost	Medium processing cost
Example corner cases	Reflections from metal objects	Curb stones, round poles	Dust, smoke, fog, low reflectivity objects	Sunlight, smoke, fog, low reflectivity objects	Dust, smoke, fog, low reflectivity objects

Table 2 - Types of Range sensors

The LIDAR we selected is called YD-Lidar X4. The features are as follows

- 360° Scanning Range
- 10meter Range
- Strong ability to resist environmental light interference
- Configurable Motor Speed, 6Hz~12Hz
- Maximum scan sampling frequency 5Khz

3. Raspberry Pi 3b+

Raspberry Pi is used as the brain for the system. It runs Ubuntu 18.04 with ROS Melodic as the middleware for our application. ROS communicates with the Arduino board and YDLIDAR serially which are connected to it via USB interface. We choose Raspberry Pi because it is small, fast SBC, and is cheap as compared to its counterparts. It is powered by a 10000mAh power bank providing 2.1A at the output.

4. Arduino board

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560. It includes digital input/output pins-54, where 16 pins are analog inputs, 14 are used like PWM outputs hardware serial ports (UARTs) – 4, a crystal oscillator-16 MHz, an ICSP header, a power jack, a USB connection, as well as an RST button. This board mainly includes everything essential for supporting the microcontroller. So, the power supply of this board can be done by connecting it to a PC using a USB cable, or battery, or an AC-DC adapter.

5. ARDUINO MOTOR SHIELD

The L293D is a dedicated module to fit in Arduino UNO R3 Board, and Arduino MEGA, It is a motor driver shield that has a full-featured Arduino Shield can be used to drive 2 to 6 DC motor and 4 wire Stepper motor and it has 2 sets of pins to drive a SERVO. L203D is a monolithic integrated that has a feature to adopt high voltage, high current at four-channel motor driver designed to accept loads such as relays solenoids, DC Motors and Stepper Motors and switching power transistor. To simplify to use as two bridges on each pair of channels and equipped with an enable input. A separate supply input is provided for the logic, allowing operation at a lower voltage, and internal clamp diodes are included.

6. Operation flow diagram and theory

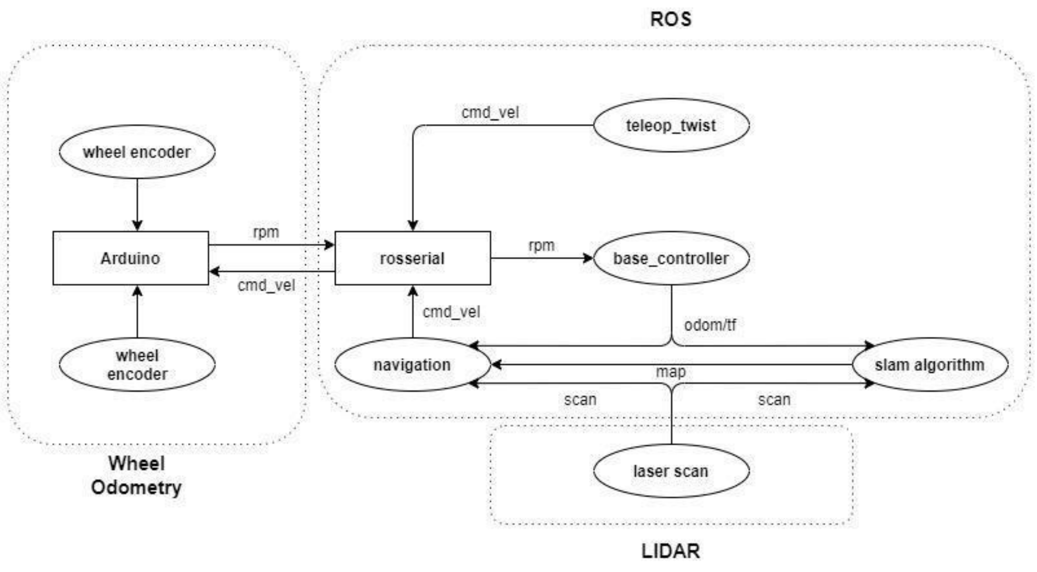


Fig. 5 Block diagram

6.1. Wheel Odometry

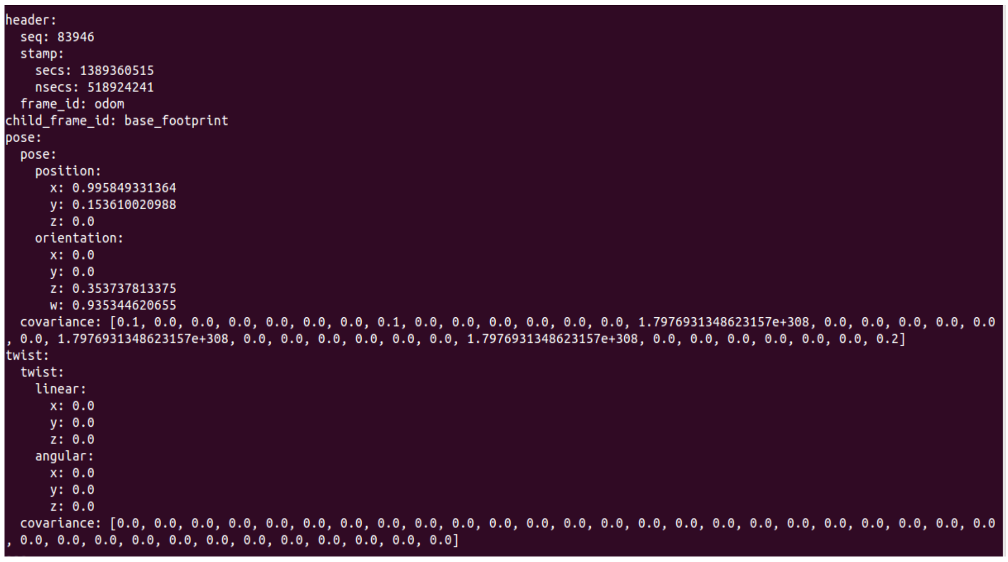
The technique of measuring the movement of the vehicle, called odometry, requires an encoder that translates the turn of the wheels into the corresponding traveled distance.

Odometry is not always as accurate as one would like, but it is the cornerstone of tracking robot movement. Odometry is simple, inexpensive, and easy to accomplish in real-time. It can be used as a feedback mechanism for SLAM based robotic vehicles for accurate positioning.

Properties of odometry

Properties of odometry as they relate to differential-drive vehicles (i.e., vehicles that have two independently driven wheels). Encoders are typically mounted on the drive motors to count the wheel revolutions. Using simple geometric equations, it is straightforward to compute the momentary position of the vehicle relative to a known starting position. This computation is called odometry. It is important to note that when considering errors in odometry, orientation errors are the main source of concern. This is so because once incurred, orientation errors grow without bound into lateral position errors. Odometry is based on the assumption that wheel revolutions can be translated into linear displacement relative to the floor. This assumption is only of limited validity.

The two wheels are mounted with a wheel encoder that increases or decreases a tick counter following the forward-backward movements of the wheels, respectively. Wheel encoders are implemented to fetch the relative position of the robot. This conception is called the odometry. The important processed data needed for odometry are the radius of the wheel and the distance between wheels. When the velocity of the left wheel is less than the velocity of the right wheel, the mobile robot takes a left turn. Similarly, when the velocity of the right wheel is less than the velocity of the left wheel, the mobile robot takes right turn and with equal velocities, the robot moves straight. Their respective encoders determine the velocities. After initializing, the robot first determines its initial position, then by the set of equations it estimates new positions and stores in the form of coordinates and the execution goes on until the robot is reached to the final position.



```
header:
  seq: 83946
  stamp:
    secs: 1389360515
    nsecs: 518924241
  frame_id: odom
child_frame_id: base_footprint
pose:
  pose:
    position:
      x: 0.995849331364
      y: 0.153610020988
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.353737813375
      w: 0.935344620655
  covariance: [0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.7976931348623157e+308, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 1.7976931348623157e+308, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.7976931348623157e+308, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.2]
twist:
  twist:
    linear:
      x: 0.0
      y: 0.0
      z: 0.0
    angular:
      x: 0.0
      y: 0.0
      z: 0.0
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Fig. 6 Wheel odometry

6.2. LIDAR

The basic principles for traditional LiDAR have been seen in time-of-flight-based distance measurement and triangulation-based measurement. The ToF concept uses the delay time between the signal emission and its reflection to compute the distance to the target. The time delay is not measured for one particular beam's round trip. Instead, the phase shifts for multiple signals are used to indirectly obtain the ToF and then compute the distance. The triangulation method is used to measure the distance between the source and the obstruction. Using 3D LiDAR offers richer information about the environment and more accurate and finessed maps in real-time. For using YD Lidar X4 we need to send to ROS the following data in the following form

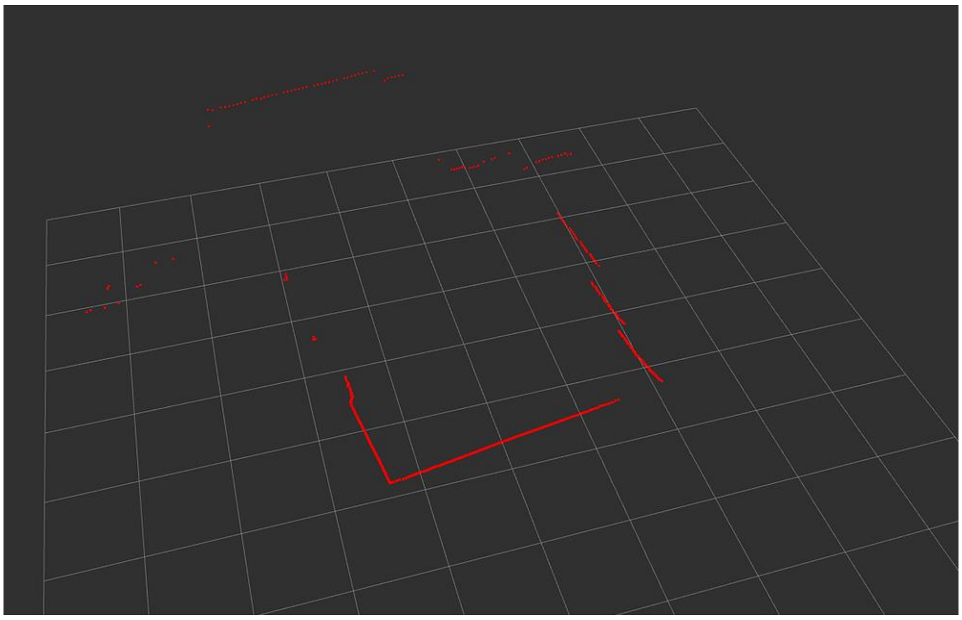


Fig.7 LIDAR data

CHAPTER 5

1. Results and Discussion

The results were generated by running hector SLAM and cartographer SLAM on our setup using ROS for processing all the data and then visualizing it on a robotics visualization application called Rviz. ROS stores the map data in a .pgm file for later use for navigation.

Hector SLAM

Hector SLAM is a SLAM approach that can be used without odometry as well as on platforms that exhibit roll/pitch motion (of the sensor, the platform, or both). It leverages the high update rate of modern LIDAR. While the system does not provide explicit loop closing ability, it is sufficiently accurate for many real-world scenarios.

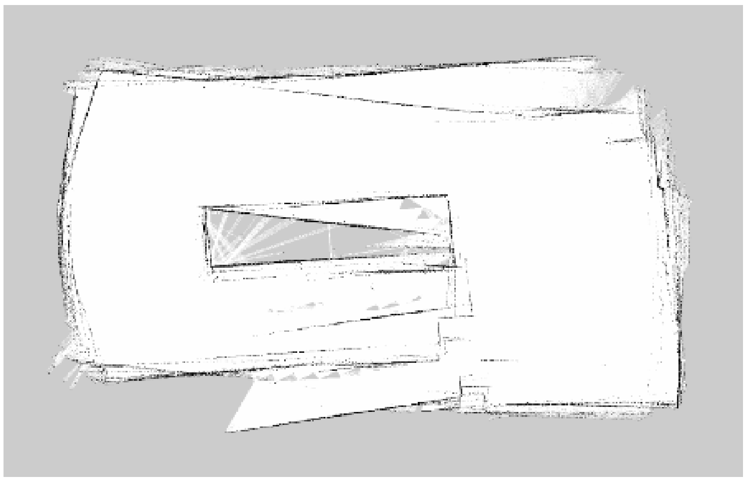


Fig. 8 Hector Slam output using ROS

Three problems are observed in tests about hector slam:

- The corridor problem: Hector slam thinks the robot is not moving when laser scan data is almost the same at the long corridor. Because hector slam is not using an odometer (exclude roll & pitch), the problem cannot be solved easily.
- The quake problem: hector slam suffers when the robot is during a body shake (probably when overcoming an obstacle or uneven surface), it has a great possibility to mess up the map (drift in yaw, the map direction is then changed) and can't recover since then.

Cartographer SLAM

Local SLAM generates a series of submaps, which are meant to be locally consistent but drift over time. Local SLAM also creates a local trajectory. Global SLAM runs in separate threads and finds loop closure constraints between submaps generated by local SLAM. It executes loop closure by scan matching scans of sensor data against the submaps. It also incorporates other sensors to attempt to get the most consistent global solution.

Cartographer is mostly a LIDAR SLAM method but can also be paired with IMU to increase accuracy. It is an indoor slam method and generally needs to be paired with other sensors if being used outside. Since this method has loop closure so it doesn't suffer from the problems, hector slam had to face.

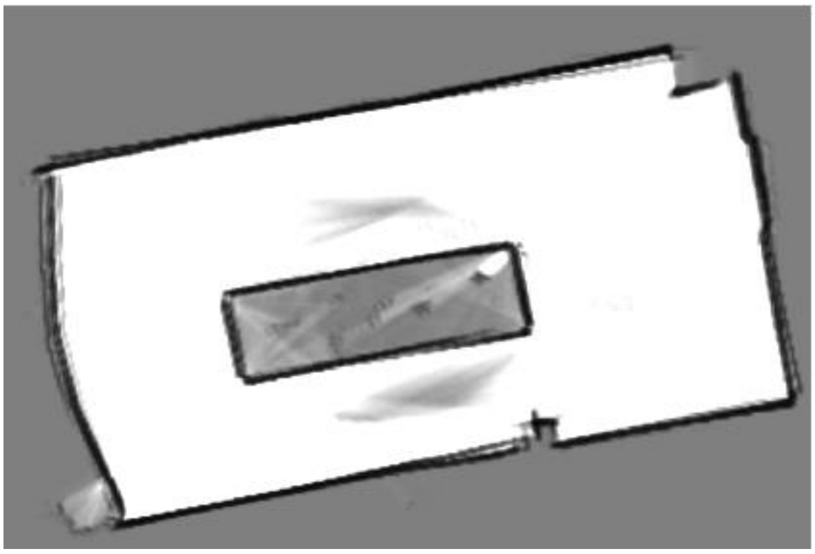


Fig. 9 Cartographer output using ROS

2. Conclusion

In this project, our objective is to develop an indoor mapping system using an autonomous robot that is used to map the unknown area and follow it with avoidance of obstacles, if any.

Various algorithms for mapping and navigation, have been tested and verified as software and are being implemented on the proposed hardware set up. The sensors being used to gather information about the surrounding area Lidar and Odometry sensors. We plan to use the sensor fusion technique to navigate the robot through an unknown environment.

Based on the two slam algorithms tested, it is noted that cartographer can perform slam with quite a better accuracy than hector slam but we need odometry data if we want to map bigger regions and generate an accurate map. The important thing here is that the mapping is not autonomous exploration-based because the robot can't navigate autonomously since we do not have odometry data necessary to localize the robot properly. Thus we can build a map without odometry but we need odometry data if the robot needs to autonomously navigate.

CHAPTER 6

APPENDIX

A. CODES

```
#include <Adafruit_MotorShield.h>

#include <Wire.h>

#include <PID_v1.h>

#include <ros.h>

#include <std_msgs/String.h>

#include <geometry_msgs/Vector3Stamped.h>

#include <geometry_msgs/Twist.h>

#include <ros/time.h>

 //initializing all the variables

#define LOOPTIME                    100     //Looptime in millisecond

const byte noCommLoopMax = 10;                //number of main loops the robot will execute
without communication before stopping

unsignedintnoCommLoops = 0;               //main loop without communication counter

 charlog_msg[50];

char result[8];

double speed_cmd_left2 = 0;

 constint PIN_ENCOD_A_MOTOR_LEFT = 2;

constint PIN_ENCOD_B_MOTOR_LEFT = 4;

constint PIN_ENCOD_A_MOTOR_RIGHT = 3;

constint PIN_ENCOD_B_MOTOR_RIGHT = 5;            //B channel for encoder of right motor

 constint PIN_SIDE_LIGHT_LED = 46;                //Side light blinking led pin
 unsigned long lastMilli = 0;
const double radius = 0.04;                   //Wheel radius, in m
const double wheelbase = 0.187;               //Wheelbase, in m
 doublespeed_req = 0;                         //Desired linear speed for the robot, in m/s
doubleangular_speed_req = 0;                  //Desired angular speed for the robot, in rad/s
 doublespeed_req_left = 0;                    //Desired speed for left wheel in m/s
doublespeed_act_left = 0;                     //Actual speed for left wheel in m/s
doublespeed_cmd_left = 0;                     //Command speed for left wheel in m/s
 doublespeed_req_right = 0;                   //Desired speed for right wheel in m/s
doublespeed_act_right = 0;                    //Actual speed for right wheel in m/s
doublespeed_cmd_right = 0;                    //Command speed for right wheel in m/s
const double max_speed = 0.4;                 //Max speed in m/s
intPWM_leftMotor = 0;
intPWM_rightMotor = 0;
volatile float pos_left = 0;
volatile float pos_right = 0;
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
 Adafruit_DCMotor *leftMotor = AFMS.getMotor(1);
Adafruit_DCMotor *rightMotor = AFMS.getMotor(2);
ros::NodeHandlenh;
//function that will be called when receiving command from host
voidhandle_cmd (constgeometry_msgs::Twist&cmd_vel) {
  noCommLoops = 0;
  speed_req = cmd_vel.linear.x
  angular_speed_req = cmd_vel.angular.z;                        speed_req_left = speed_req -
angular_speed_req*(wheelbase/2);
```

```
  speed_req_right = speed_req + angular_speed_req*(wheelbase/2

}

ros::Subscriber<geometry_msgs::Twist>cmd_vel("cmd_vel",  handle_cmd);     //create  a
subscriber to ROS topic for velocity commands (will execute "handle_cmd" function when
receiving data)

geometry_msgs::Vector3Stamped speed_msg;

ros::Publisher speed_pub("speed", &speed_msg);

void setup() {

  pinMode(PIN_SIDE_LIGHT_LED, OUTPUT);     //set pin for side light leds as output

  analogWrite(PIN_SIDE_LIGHT_LED, 255);    //light up side lights

  nh.initNode();                        //init ROS node

  nh.getHardware()->setBaud(57600);        //set baud for ROS serial communication

  nh.subscribe(cmd_vel);                  //suscribe to ROS topic for velocity commands

  nh.advertise(speed_pub);                //prepare to publish speed in ROS topic

  AFMS.begin();

  //setting motor speeds to zero

  leftMotor->setSpeed(0);

  leftMotor->run(BRAKE);

  rightMotor->setSpeed(0);

  rightMotor->run(BRAKE);

  // Define the rotary encoder for left motor

  pinMode(PIN_ENCOD_A_MOTOR_LEFT, INPUT);

  pinMode(PIN_ENCOD_B_MOTOR_LEFT, INPUT);

  digitalWrite(PIN_ENCOD_A_MOTOR_LEFT, HIGH);             // turn on pullup resistor

  digitalWrite(PIN_ENCOD_B_MOTOR_LEFT, HIGH);

  attachInterrupt(0, encoderLeftMotor, RISING);
  // Define the rotary encoder for right motor
  pinMode(PIN_ENCOD_A_MOTOR_RIGHT, INPUT);
  pinMode(PIN_ENCOD_B_MOTOR_RIGHT, INPUT);
  digitalWrite(PIN_ENCOD_A_MOTOR_RIGHT, HIGH);          // turn on pullup resistor
  digitalWrite(PIN_ENCOD_B_MOTOR_RIGHT, HIGH);
  attachInterrupt(1, encoderRightMotor, RISING);
}
void loop() {
  nh.spinOnce();
  if((millis()-lastMilli) >= LOOPTIME)
  {                                                     // enter timed loop
    lastMilli = millis
    if (abs(pos_left) < 5){                             //Avoid taking in account small
disturbances
      speed_act_left = 0;
    }
    else {
      speed_act_left=((pos_left/990)*2*PI)*(1000/LOOPTIME)*radius;      // calculate speed
of left wheel
    }
    if (abs(pos_right) < 5){                            //Avoid taking in account small
disturbances
      speed_act_right = 0;
    }
    else {
    speed_act_right=((pos_right/990)*2*PI)*(1000/LOOPTIME)*radius;      // calculate
speed of right wheel
```

```
  }
  pos_left = 0;
  pos_right = 0;
  speed_cmd_left = constrain(speed_cmd_left, -max_speed, max_speed);
  if (noCommLoops>= noCommLoopMax) {                    //Stopping if too much time without command
    leftMotor->setSpeed(0);
    leftMotor->run(BRAKE);
  }
  else if (speed_req_left == 0){                        //Stopping
    leftMotor->setSpeed(0);
    leftMotor->run(BRAKE);
  }
  else if (PWM_leftMotor> 0){                           //Going forward
    leftMotor->setSpeed(abs(PWM_leftMotor));
    leftMotor->run(BACKWARD);
  }
  else {                                                //Going backward
    leftMotor->setSpeed(abs(PWM_leftMotor));
    leftMotor->run(FORWARD);
  }
  speed_cmd_right = constrain(speed_cmd_right, -max_speed, max_speed);
  if (noCommLoops>= noCommLoopMax) {                    //Stopping if too much time without command
    rightMotor->setSpeed(0);
    rightMotor->run(BRAKE);
```

```
   }
   else if (speed_req_right == 0){                     //Stopping
     rightMotor->setSpeed(0);
     rightMotor->run(BRAKE);
   }
   else if (PWM_rightMotor> 0){                       //Going forward
     rightMotor->setSpeed(abs(PWM_rightMotor));
     rightMotor->run(FORWARD);
   }
   else {                                             //Going backward
     rightMotor->setSpeed(abs(PWM_rightMotor));
     rightMotor->run(BACKWARD);
   }
   if((millis()-lastMilli) >= LOOPTIME){          //write an error if execution time of the loop in
longer than the specified looptime
     Serial.println(" TOO LONG ");
   }
   noCommLoops++;
   if (noCommLoops == 65535){
     noCommLoops = noCommLoopMax;
   }
   publishSpeed(LOOPTIME);   //Publish odometry on ROS topic
  }
}
voidpublishSpeed(double time) {
  speed_msg.header.stamp = nh.now();      //timestamp for odometry data
  speed_msg.vector.x = speed_act_left;    //left wheel speed (in m/s)
  speed_msg.vector.y = speed_act_right;   //right wheel speed (in m/s)
  speed_msg.vector.z = time/1000;            //looptime, should be the same as specified in
LOOPTIME (in s)
  speed_pub.publish(&speed_msg);
  nh.spinOnce();
  nh.loginfo("Publishing odometry");
}
voidencoderLeftMotor() {
  if(digitalRead(PIN_ENCOD_A_MOTOR_LEFT)==digitalRead(PIN_ENCOD_B_MOTOR
_LEFT))
pos_left++;
  elsepos_left--;
}
voidencoderRightMotor() {
  if (digitalRead(PIN_ENCOD_A_MOTOR_RIGHT) ==
digitalRead(PIN_ENCOD_B_MOTOR_RIGHT)) pos_right--;
  elsepos_right++;
}
template<typename T>intsgn(T val) {
    return (T(0) <val) - (val< T(0));
}


<launch>
                                <!--  ************** Odometry ***************  -->
        <arg name="gui" default="True" />
        <param name="use_gui" value="$(arggui)"/>
        <param name="robot_description" command="cat $(find
nox_description)/urdf/nox.urdf" />


        <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher" />


        <node name="robot_state_publisher" pkg="robot_state_publisher"
type="state_publisher" />

        <node name="serial_node" pkg="rosserial_python" type="serial_node.py">

                <param name="port" value="/dev/ttyACM0"/>

        </node>

        <node name="nox_controller" pkg="nox" type="nox_controller">

                <param name="publish_tf" value="true" />

                <param name="publish_rate" value="10.0" />

                <param name="linear_scale_positive" value="1.025" />

                <param name="linear_scale_negative" value="1.025" />

                <param name="angular_scale_positive" value="1.078" />

                <param name="angular_scale_negative" value="1.078" />

                <param name="angular_scale_accel" value="0.0" />

        </node>

                        <!--  ************** Sensors ***************  -->

        <node name="depthimage_to_laserscan" pkg="depthimage_to_laserscan"
type="depthimage_to_laserscan">

                <remap from="image" to="camera/depth/image_raw" />

</node>

</launch>

<launch>
```

```
        <include file="$(find nox)/launch/nox_navigation.launch" />
                                        <!--  ************** gmapping ***************  -->
        <node name="slam_gmapping" pkg="gmapping" type="slam_gmapping"
output="screen"/>
                                <!--  ************* Visualisation **************  -->
        <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
nox)/cfg/rviz_slam_base_local_planner.rviz" required="true" />
</launch>
<launch>
        <!--  ************** Navigation ***************  -->
        <node pkg="move_base" type="move_base" respawn="false" name="move_base"
output="screen">
                <rosparam file="$(find nox)/cfg/costmap_common_params.yaml"
command="load" ns="global_costmap" />
                <rosparam file="$(find nox)/cfg/costmap_common_params.yaml"
command="load" ns="local_costmap" />
                <rosparam file="$(find nox)/cfg/local_costmap_params.yaml"
command="load" />
                <rosparam file="$(find nox)/cfg/global_costmap_params.yaml"
command="load" />
                <rosparam file="$(find nox)/cfg/dwa_local_planner_params.yaml"
command="load" />
                <param name="base_local_planner"
value="dwa_local_planner/DWAPlannerROS" />
                <param name="controller_frequency" value="5.0" />
                <param name="controller_patience" value="15.0" />
                <param name="clearing_rotation_allowed" value="true" /><!--Nox is able to
rotate in place -->
```

```
        </node>

</launch>
```

CHAPTER 7

References

1. Kohlbrecher, Stefan & Meyer, Johannes & Graber, Thorsten & Petersen, Karen &Klingauf, Uwe & Von Stryk, Oskar. (2014). Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots.
2. Grisetti, Giorgio &Stachniss, Cyrill&Burgard, Wolfram. (2007). Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters.
3. W. Hess, D. Kohler, H. Rapp, and D. Andor, Real-Time Loop Closure in 2D LIDAR SLAM, in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016. pp. 1271–1278.
4. Yeon Kang, Donghan Kim, Kwangjin Kim, 2019, URDF Generator for Manipulator Robot. Third IEEE International Conference on Robotic Computing (IRC).
5. http://gazebosim.org/tutorials/?tut=ros_urdf
6. https://ocw.tudelft.nl/course-lectures/2-2-1-introduction-to-urdf/
7. https://linorobot.org/
8. https://github.com/RBinsonB/Nox_robot
9. Sherif A. S. Mohamed, Mohammad-Hashem Haghbayan&TomiWesterlund, JukkaHeikkonen, HannuTenhunen&JuhaPlosila. A Survey on Odometry for Autonomous Navigation Systems.
10. Implementing Odometry and SLAM Algorithms on a Raspberry Pi to Drive a Rover
11. Position and Velocity control for Two-Wheel Differential Drive Mobile Robot
12. Deploying on Mars: Rock solid odometry for wheeled robots

ACKNOWLEDGEMENT

We would like to express our deep gratitude towards everyone who supported and assisted in our project work. We would also like to express our sincere gratitude towards our project guide Dr. Anjali Deshpande ma'am and Prof.AkhilMasurkar sir for providing valuable suggestions, for their ongoing support during the project, from initial advice, and provision of contacts in the first stages through ongoing advice and encouragement, which led to the final report of this project.

We would like to thank our Principal, for providing the necessary facilities required for the completion of this project. We are also grateful to our lab assistant, who has helped us with the setup for the project and gave practical insight.

A special acknowledgment goes to my colleagues who helped me in completing the project by exchanging interesting ideas and sharing their experiences.