An Intelligent Mobile Robot that Localizes itself Inside a Maze

By

Abdalmenem Amleh

Firas Almohtaseb

Mohammad Fayez Mohtaseb

Supervisor

Dr. Iyad Hashlamon

Submitted to the College of Engineering

in partial fulfillment of the requirements for the

Bachelor degree in Mechatronics Engineering

Palestine Polytechnic University

August 2020

An Intelligent Mobile Robot that Localizes itself Inside a Maze

By

Abdalmenem Amleh

Firas Almohtaseb

Mohammad Fayez Mohtaseb

Submitted to the College of Engineering

in partial fulfillment of the requirements for the

Bachelor degree in Mechatronics Engineering

Supervisor Signature

……………………………

Chair of the Department Signature

………………………………

Abstract

In the past few years, some problems related to mobile robots have risen, such as mapping, localization, and object detection. Scientists and researchers have been racing to find ways to solve these problems ever since. Although, limited work has been developed to combine these problems into one intelligent mobile robot. This project makes use of the past researches about the mentioned problems, with the help of Robot Operating System (ROS), to make a simulation of a mobile robot capable of mapping a maze environment and localizing itself inside it, to navigate and detect a certain object in that environment.

In this project, the simulated robot uses a laser sensor (LIDAR) to scan the environment. ROS then transforms this data into occupancy grid map, which the robot then uses to localize itself through the adaptive Monte Carlo Localization algorithm (MCL). The robot makes use of a camera, and uses the You Only Look Once algorithm (YOLO) for object detection. The robot can also find the shortest path for navigation through Dijkstra's algorithm.

A software provided by ROS called Gazebo is used to simulate the robot navigating inside the simulated environment. One hundred simulated localization experiments have been done and distributed equally across 10 maze environments. The resulting success rate was 62%, and the average time of localization and navigation for the successful attempts was 139 seconds.

الملخص

في الآونة الأخيرة، ظهرت مجموعة من المشاكل المتعلقة بالروبوتات المتحركة، كرسم الخرائط، وتحديد الموقع، واكتشاف الأجسام. العديد من العلماء والباحثين يتسابقون للوصول لطرق لحل هذه المشاكل منذ ذلك الحين. على الرغم من ذلك، عدد محدود من الأبحاث والأعمال جرت لجمع هذه المشاكل وحلها باستخدام روبوت ذكي متحرك واحد. هذا المشروع يستفيد من الأبحاث السابقة في المشاكل المذكورة، بمساعدة برنامج ((Robot Operating System (ROS)، لعمل محاكاة لروبوت متحرك قادر على رسم خريطة لمتاهة وتحديد موقعه داخلها، للتنقل واكتشاف جسم محدد في هذه المتاهة.

في هذا المشروع، يستخدم الروبوت المحاكى مجس ليزر (LIDAR) لمسح البيئة. يقوم (ROS) بعدها بتحويل هذه البيانات إلى خريطة شبكية (Occupancy Grid Map)، التي يستخدمها الروبوت بعد ذلك لتحديد موقعه باستخدام خوارزمية مونتي كارلو لتحديد المواقع ((Monte Carlo Localization (MCL). يستخدم الروبوت كاميرا، وخوارزمية (You Only Look Once (YOLO)) لاكتشاف الجسم. يستطيع الروبوت أيضا معرفة أقصر طريق للتنقل باستخدام خوارزمية دايكسترا (Dijkstra’s algorithm).

تم استخدام برنامج (Gazebo) المقدم من (ROS) لمحاكاة حركة الروبوت داخل بيئة افتراضية. تم عمل مئة تجربة تحديد موقع مقسمة بالتساوي على 10 متاهات. نسبة النجاح الناتجة كانت 62%، ومعدل زمن تحديد الموقع والتنقل للمحاولات الناجحة كان 139 ثانية.

Dedication

We dedicate our project to our beloved parents for their continuous invaluable support and encouragement all through the years and to our dear siblings for providing us with a comfortable environment for study and research.

Acknowledgment

We would like to express our gratitude to our supervisor, Dr. Iyad Hashlamon, for his full support and guidance and remarkable suggestions. We would also like to thank our teachers for all the efforts they have exerted to make us qualified engineers who can assume, with confidence, our role in building our community. Thanks are also due to our classmates and friends for their cooperation and encouragement.

Table of Contents

Chapter 1 Project Proposal 1
1.1 Introduction 2
1.2 Recognition of the need 2
1.3 Project Importance 3
1.4 Literature Review 3
1.4.1 Papers related to mapping and localization 3
1.4.2 Previous work on object recognition 5
1.4.3 Researches featuring control systems of mobile robots 6
1.5 Action plan 6
1.6 Cost table 9
Chapter 2 Conceptual Design 10
2.1 Introduction 11
2.2 Decomposition 11
2.3 Ideas, Evaluation, and Selection 11
2.3.1 Robot type 11
2.3.2 Distance sensors types 12
2.3.3 Mapping method 14
2.3.4 Localization method 14
2.3.5 Optimal path calculation method 14
2.3.6 Object detection and localization method 14
2.4 Relationship among components 15
Chapter 3 Robot Navigation 16
3.1 Introduction 17
3.2 Mapping 17
3.3 Localization 19
3.3.1 Monte Carlo Localization (MCL) 20
3.3.2 Recovery from failures 22
3.4 Optimal Path 23
Chapter 4 Object Detection 26
4.1 Introduction 27
4.2 YOLOv4 Custom Training 27
4.2.1 Introduction 27
4.2.2 Preparing Darknet 27
4.2.3 Collecting Data 28

4.2.4 Labeling 28
4.2.5 Google Colab 29
4.2.6 Start Training 30
4.3 Object Detection 32
Chapter 5 Experiments and Results 33
5.1 Introduction 34
5.2 Mapping 34
5.3 Localization and Navigation 38
5.4 Object Detection 45
5.5 Integration of Subsystems 48
Chapter 6 Conclusion and Future Work 50
6.1 Conclusion 51
6.2 Future Work 51
Appendix A Kinematics and Dynamics of the Mobile Robot 52
A.1 Introduction 53
A.2 Non-Holonomic motion 53
A.3 Kinematics model 54
A.4 Dynamic model 58
A.4.1 Lagrange dynamic approach 59
A.4.2 Actuator dynamics 62
Appendix B Robot Operating System (ROS) 65
B.1 What is ROS? 66
B.2 How does ROS work? 66
B.2.1 ROS Filesystem Level 66
B.2.2 ROS Computation Graph Level 67
B.2.3 ROS Community Level 68
B.3 ROS Capabilities 69
B.3.1 Robots using ROS 69
B.3.1.1 TurtleBot 2 70
Appendix C ROS Adjustable Parameters 78
C.1 AMCL Adjustable Parameters 79
C.2 gMapping Adjustable Parameters 80
Appendix D YOLO Algorithm 82
D.1 You Only Look Once (YOLO) 83
D.2 How YOLO Works 83

List of Figures

Figure 2.1 Simple DDMR on the internet [16] 11
Figure 2.2 Turtlebot 2 [17] 12
Figure 2.3 Ultrasonic sensor mounted on a servo motor 12
Figure 2.4 eight ultrasonic sensors mounted to cover 360 degrees 13
Figure 2.5 Kinect sensor 13
Figure 2.6 LIDAR sensor 13
Figure 2.7 Conceptual Design Schematic 15
Figure 3.1 One-dimensional hallway example of MCL 21
Figure 3.2 Pseudo code for Dijkstra's algorithm.[29] 23
Figure 3.3 A simple example illustrating Dijkstra's Algorithm 24
Figure 4.1 Images of Our Target 28
Figure 4.2 Labelling the Target 29
Figure 4.3 Plot of Average Training Loss and Overall Validation mAP Against Number of Iterations 31
Figure 4.4 Outcome of Learning Process 31
Figure 4.5 Object Detection Flowchart 32
Figure 5.1 A Block Diagram of All the Nodes and Topics Running by ROS During the Mapping Phase 34
Figure 5.2 A Block Diagram of All the Nodes and Topics Running by ROS During the Localization and Navigation Phase 39
Figure 5.3 Localization Status at T-4 40
Figure 5.4 Localization Status at T 41
Figure 5.5 Localization Status at T+4 41
Figure 5.6 Localization Status at T+10 42
Figure 5.7 Localization Status at T+11 42
Figure 5.8 Localization Status at T+23 43
Figure 5.9 Localization Status at T+33 43
Figure 5.10 Localization Status at T+44 44
Figure 5.11 Localization Status at T+55 44
Figure 5.12 Localization Status at T+82 45
Figure 5.13 99.69% Prediction Confidence 45
Figure 5.14 99.30% Prediction Confidence 46
Figure 5.15 95.10% Prediction Confidence 46
Figure 5.16 97.35% Prediction Confidence 47
Figure 5.17 90.12% Prediction Confidence 47
Figure 5.18 86.23% Prediction Confidence 47
Figure 5.19 95.95% Prediction Confidence 48
Figure 5.20 99.60% Prediction Confidence 48
Figure A.1 The generalized coordinates x, y, and φ. 54
Figure A.2 Geometry of differential drive WMR [35] 56
Figure A.3 Diagram illustrating the nonholonomic constraint [35] 56
Figure A.4 Circuit equivalent of a DC motor 62
Figure B.1 Data Exchange with ROS Publishers and Subscribers 68
Figure B.2 TurtleBot 2 70
Figure B.3 Anatomy of Kabuki. a) Top view. b) Bottom view. 71
Figure B.4 Kobuki base components 72
Figure B.5 Kinect components 74

Figure B.6 LIDAR sensor 76
Figure B.7 Control panel for Kobuki 76
Figure D.1 YOLO Architecture 83
Figure D.2 Parameters of bounding box 84
Figure D.3 Splitting the Image by a 19×19 grid 84
Figure D.4 The Effect of non-max suppression 85

List of Tables

Table 1.1 Action plan for the first semester 7
Table 1.2 Action plan for the second semester 7
Table 1.3 Cost table 9
Table 5.1 The Experimented Mazes and the Resulting Maps 35
Table 5.2 Localization and Navigation Experemental Results 40
Table A.1 System parameters 64
Table B.1 Some specifications of Kinect 75
Table B.2 I/O specifications of power adapter 77

Chapter 1 **Project Proposal**

1.1 Introduction

In any time and in any place, there is always a need to move and to transfer objects from one place to another. Hands are used to remove a stone out of the way, a shopping cart is used to collect grocery, a conveyor belt is used to move products in a factory, and ships are used to transfer cargo. All of these are different methods to achieve the need of moving objects.

In the last few decades, robots have been replacing humans in a lot of aspects mainly to reduce efforts and to save time. Robots can do tasks that humans cannot do, such as going through a very thin tunnel, or moving heavy objects. They can also be used to do risky tasks, like moving objects in a very high place, or even defusing a bomb.

Mobile robots have the capability to move around in their environment and are not fixed to one physical location. Mobile robots can be autonomous (AMR - autonomous mobile robot), which means they are capable of navigating in an uncontrolled environment without the need for physical or electro-mechanical guidance devices. Alternatively, mobile robots can rely on guidance devices that allow them to travel in a pre-defined navigation route in relatively controlled space (AGV - autonomous guided vehicle).

The evolution of mobile robots nowadays lead for some problems to rise, such as Mapping, Local and Global Localization, Kidnapped Robot problem, and more. These problems have encouraged the scientists around the world to find methods, and to build algorithms to solve them.

This project is about making an intelligent mobile robot that can map a maze-like environment and can localize itself inside that environment, using Robot Operating System (ROS), and by interacting with the environment through a laser sensor (LIDAR).

1.2 Recognition of the need

To help humans save time and efforts, there is a need for mobile robots capable of mapping an environment and then be able to localize itself inside that environment and navigate in it.

Requirements

- The mobile robot should be able to map the environment around it.
- The robot should localize itself in the environment.
- The robot should choose an optimal path to reach a predefined target.
- The environment is a variable maze, not specific and it has no specific connotations.
- The robot must be portable in order to be placed in a random place within the maze.
- The robot must determine the best way to pass through the maze to reach the target.
- The maze should be suitable so that the robot is able to move within it.
- The robot must have a rechargeable battery.
- The robot should avoid hitting the maze walls.
- The robot must be user friendly.

1.3 Project Importance

The principle that we are working on in the project is that the robot works to determine its location within a maze and then to move a box located in a certain place within the maze.

The maze can be considered as a specific factory or facility such as a port, and the process of transporting and arranging products using automatically working robots has the same principle of the work of our project.

Nowadays, products are transported inside factories and other facilities by human effort, even machines like forklifts are manually operated and under the supervision of human. But if we look at the problems of this profession, we can see that man cannot transport goods and arrange them in the best way, unlike the robot, which transports and arranges the goods in the optimal way by some software lines, which provides storage space for goods, and also saves effort and time.

Moreover, mobile robots are also widely used in people’s daily lives, such as house cleaning, medical services, catering services military, intelligent transportation and entertainment.[1]

Nowadays, mobile robots are widely used in field of applications to reduce human efforts and save time. Mobile robots can navigate an uncontrolled environment without the need for physical or electro-mechanical guidance devices. It relies on guidance devices that allow them to travel a pre-defined navigation route in relatively controlled space.

1.4 Literature Review

This section presents the most important papers and previous work that are related to our project.

1.4.1 Papers related to mapping and localization

In [1], the authors focused on some application where mobile robot works in our lives, such as house cleaning, medical services, catering services, military, intelligent transportation and entertainment.

So, if mobile robots work autonomously in these areas, those robots must be able to build maps of the surrounding environment. The robot explores every corner of the surroundings and determines its position in the map and the orientation of the body according to the surrounding environment, by using the open source mapping algorithms, karto SLAM[1], hector SLAM software package for indoor SLAM, which can get the indoor grid maps in ROS graphical tool RVIZ.

Their methodology depends on implementing the algorithm of SLAM. It depends on that the mobile robot contains a LIDAR to scan the indoor environment, realizes self-localization and builds a map of an anonymous environment. To measure the mapping error of each SLAM algorithm, run gmapping[2], karto SLAM and graph-based SLAM using UTM- 30LX LIDAR to ensure the robot navigates correctly in the surrounding environment.

[1] SLAM: Simultaneous Localization and Mapping
[2] Gmapping: Grid Mapping

They used a Raspberry Pi as the main controller to run their robot (four wheeled car) to complete the indoor SLAM, and all experiments were conducted based on the Robot Operating System (ROS). The Urban Search and Rescue (USAR) environment was built using the ROS simulation tool Gazebo, and their car was used to test hector SLAM in Gazebo.

They used single-line LIDAR as a 2D laser scan to match data acquisitioned in the practical experiments with the map built using the open source algorithms gmapping, karto SLAM, and hector SLAM software package for indoor SLAM.

In their experiment they drew the maze, which was 10 meters long and 6 meters wide, and established the URDF[1] model of the car by using the Rviz tool and running gmapping to simulate the results in the match environment. They used the hector SLAM algorithm for further simulations, when the speed is set to 0.2 m/s and the angular velocity to 1 rad/s, it works fine, however, once set the speed of the car to more than 0.2 m/s and the angular velocity to more than 1 rad/s, the mapping effect of hector SLAM is not satisfactory, because hector SLAM does not use the data of the odometer in the robot. So when the car spins too fast, it does not know that it is rotating. At this time, the laser data collected is not matched with other parts in the map resulting in ‘ghosting’. Hector SLAM uses only 2D LIDAR data, and there is no obvious choice for loop closure detection, this is why the hector SLAM results are less accurate.

These experiments were carried out in the laboratory, the experimental environment is a closed environment with dimensions of 16 meters by 10 meters, and all experiments of the SLAM algorithm were performed in the same environment.

They implemented three different SLAM algorithms in order to better implement the algorithm. The robot was equipped with rpLIDAR A1, IMU[2] sensor and an odometer to provide more information feedback for the actual implementation of the gmapping in the experimental environment, the map was constructed by using the karto SLAM algorithm.

In some experimental tests, especially in complicated environments, the robot has to stay for a while waiting for the LIDAR to collect more data, so high-precision grid maps can be constructed.

In [2], the authors talked about replacing industrial robots or manipulator robots with smarter robots in terms of navigation and recognizing the environment autonomously. The environment was implemented as a maze in the studied case. Robots should be able to follow the maze and get out of it intelligently.

They pointed out that there are many solutions that can be applied to solve maze problems, such as maze mapping algorithm that applies the concept of left / right hand rule or wall follower rule in searching for a way out of the maze.

The goal of their study is to design a robot that has a good safe navigation system that prevents it from colliding with the walls of the maze.

For the navigation system, the wall follower robot uses the maze mapping method with the left-hand rule, to learn the wall distance and its position, which uses RAM[3]-based

[1] URDF: Unified Robot Description Format, which is an XML format for representing a robot model
[2] IMU: Inertial Measurement Unit
[3] RAM: Random Access Memory

artificial neural network methods. This neural network has 3 RAM nodes to process the received environmental patterns. The left sensor, the right sensor, and the front sensor have 8 bits of the input pattern, the pattern will be optimized into 4 bits stored in the RAM node, so that the computing process in the robot becomes faster and simpler.

The algorithm of maze mapping here is used when the target position is unknown, the robot will see the right or left wall and navigate along the maze until it finds a target.

Research methodology is divided into several stages, the robot will process the map every time it is facing an intersection and a dead end, the codes are stored in the robot's memory and will be arranged continuously every time the robot faces an intersection and a dead end until the robot reaches the finish position.

The codes used when mapping are:

- "L", this code indicates that the robot has turned left because it passed an intersection.
- "F", this code indicates that the robot keeps going because it meets an intersection three times with the right or the left turn.
- "0", this code means that the robot encounters a dead end and walks back to the last intersection.

After exploring and storing the entire maze and knowing the finish position of the maze, the entire route is analyzed so that it can be optimized. The process of simplifying the route is by identifying a deadlock code on the route that has been saved. If there is a deadlock code, the code will be simplified.

The robot was able to get out of the maze using the left-hand rule algorithm, the data analysis on RAM also runs as desired displayed in data simulation.

1.4.2 Previous work on object recognition

The idea of the graduation project in [3] is based on replacing humans with robots to do repetitive or risky tasks. For example storage management in a warehouse/industrial environment, the authors designed a mobile manipulator with three main tasks, to find itself within a maze, to go to a certain location in the maze and find a known object, pick up the object and move it to a designated area.

They used Monte Carlo Localization (MCL) Technique, which uses Particle Filter algorithm to determine the robot's current position accurately, then updates it every time the robot moves.

MCL first generates all the possible poses of the robot, initially, the poses are all over the map. Then as the robot moves and measures distances to the surrounding environment, MCL takes two inputs, the robot's pose relative to the starting position, and a LIDAR measurement from the Time of Flight (TOF) sensor.

To pick up an object and bring it to a particular location in the map, they used computer vision.

For object detection, the authors implemented deep learning to train the robot to detect the given object in a picture.

The chosen object was designed with white body and black corner so that the black corners can be detected easily to use manipulator after that to pick it up.

1.4.3 Researches featuring control systems of mobile robots

Many researches have been conducted to study trajectory tracking control problem of wheeled mobile robot (WMR) with several control strategies, for example, adaptive control [4, 5], fuzzy logic control [6], neural network control [7], Fuzzy adaptive iterative learning control [8], and survey on various motion control problems of WMR [9].

Mobile robot control system can be classified into three categories. The first one is the sensor-based control-based approach. Such control system is emphasized on how to model the motion of a robot in a dynamic environment [10], The control process to produce estimation and predictions of the mobile robot movement is dependent on the information coming from the sensors [11].

The second category is the approach of using a path planning [12]. The control system organizes the movement of the robot through the planned path, and thus can move according to the goal position that is set up. The mapping of the environment is created to have collision-free path of robot movement. The control of the robot is depending on minimum distance, energy and time.

The third category presents the optimization algorithm and is developed for controlling with accurate trajectory. The controller design is based on mathematical model of the robot that we are working on. The approach is for tracking the mobile robot errors between reference and actual trajectory [13]. However, the whole categories of the control system only operate when the linear velocity is not zero. Therefore, a mobile robot is hard to control, especially in a short time trajectory following with Minimum errors. So nonlinear control is an approach which is used to solve this problem [14, 15]. Although the problem is solved, , it yielded slow convergence[15].

1.5 Action plan

Table 1.1 and Table 1.2 show the action plan for the first and second semesters distributed on 32 weeks.

Table 1.1 Action plan for the first semester

Weeks / Tasks	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Identification of Project Idea	■	■	■	■												
project requirement and collecting data			■	■	■	■	■									
Project Proposal Chapter 1				■	■	■	■	■								
Conceptual Design Chapter 2					■	■	■									
Robot Operating System (ROS) Appendix B								■	■	■	■					
Kinematics and Dynamics Appendix A											■	■	■			
Robot Navigation Chapter 3										■	■	■	■	■		
Object Localization Chapter 4													■	■	■	■

Table 1.2 Action plan for the second semester

Weeks / Tasks	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Maze Generation	■															
Maze Building (Simulation)		■	■													

Mapping Simulation				■	■	■	■									
Localization Simulation					■	■	■	■								
Real Maze Building									■	■						
Building Connection Between the Robot and ROS									■	■						
Mapping experiments											■	■	■			
Data Collection				■	■											
Labeling the Object						■	■									
Build the Model										■	■	■				
Object Detection													■	■	■	
Localization and Navigation Experiments											■	■	■			
Experiments and Results Chapter 5														■	■	
Conclusion and Future Work Chapter 6																■

1.6 Cost table

Table 1.3 shows the estimated cost of the project, where the hardware parts are provided by the department of Mechanical Engineering and the department of Computer and Information Technology in the Palestine Polytechnic University.

Table 1.3 Cost table

Tools and device	Number	Piece price (NIS)	Price (NIS)
Turtlebot 2	1	2515	2515
Workspace	1	50	50
Cables, Wires	1	50	50
Kinect sensor	1	160	160
Laptop	1	500	500
Workstation	1	3000	3000
2D Laser range finder (LIDAR)	1	3870	3870
	Total (NIS)		10145

Chapter 2 **Conceptual Design**

2.1 Introduction

This chapter describes the workflow of the intelligent mobile robot that can map the environment and then can localize itself and navigate inside that environment, including the subsystems, the system components, and the relationship among them.

2.2 Decomposition

The main problem was divided into several decomposed tasks:

- Robot type
- Distance sensors types
- Mapping method
- Localization method
- Optimal path calculation method
- Object detection and localization method

2.3 Ideas, Evaluation, and Selection

2.3.1 Robot type

A differential-drive mobile robot was decided to be used because it's very easy to control and move inside the maze.

Initially, the idea of designing and building a robot was suggested; however, this is time consuming and is not the project goal. Therefore, it was decided that using an already made robot from the internet is a better idea (Figure 2.1).

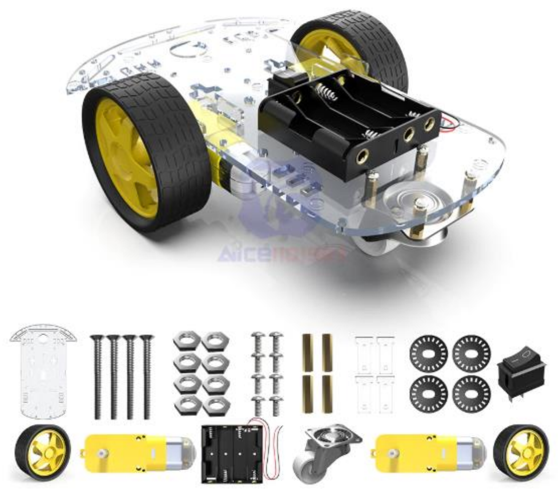


Figure 2.1 Simple DDMR on the internet [16]

This robot and its actuators have the drawback of the very small and inaccurate, so a Turtlebot 2 robot was suggested (Figure 2.2). This robot has high accuracy, it's available at the university, it's compatible with many programs, its mathematical model is available, and it comes with a number of useful sensors we can use in the project.

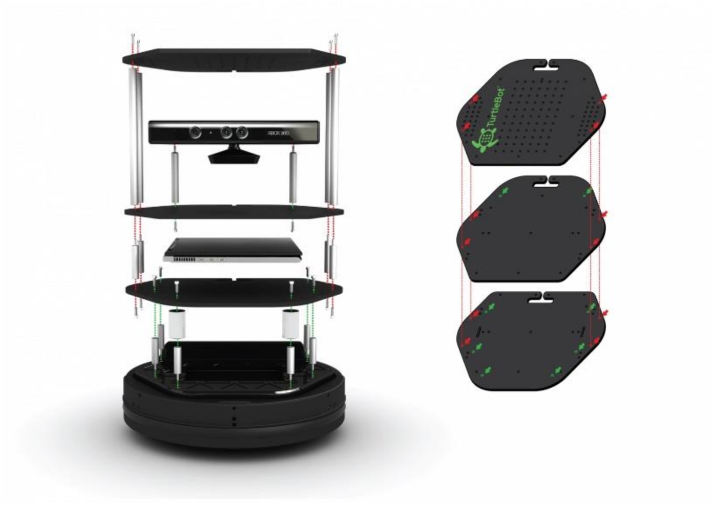


Figure 2.2 Turtlebot 2 [17]

2.3.2 Distance sensors types

To move inside the maze, a robot needs some sensors to detect obstacles around it and measure the distance of the walls from the robot.

The initial idea was using an ultrasonic sensor that rotates 360 degrees and collects information as in Figure 2.3, but this would cause the robot to stop over and over again and it is time consuming, so it was suggested to put four or eight ultrasonic sensors around the robot as in Figure 2.4, to save some time. However, a measurement problem raised up, when the walls are not perpendicular with the sensor beams, the sensor cannot read any signal.

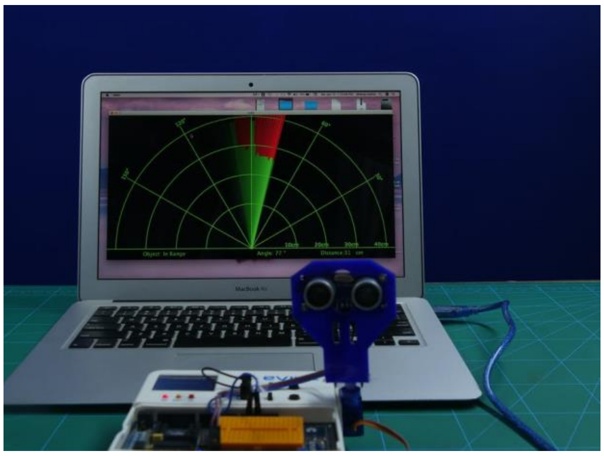


Figure 2.3 Ultrasonic sensor mounted on a servo motor

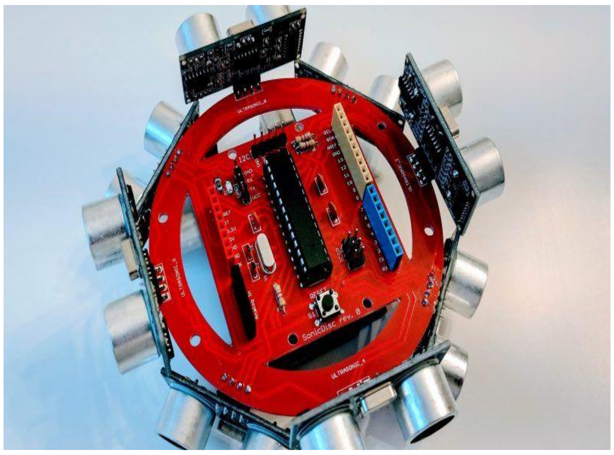


Figure 2.4 eight ultrasonic sensors mounted to cover 360 degrees

As a result. The Turtlebot 2 is used with its sensors. Further, the Kinect (Figure 2.5) and LIDAR (Figure 2.6) can be used.

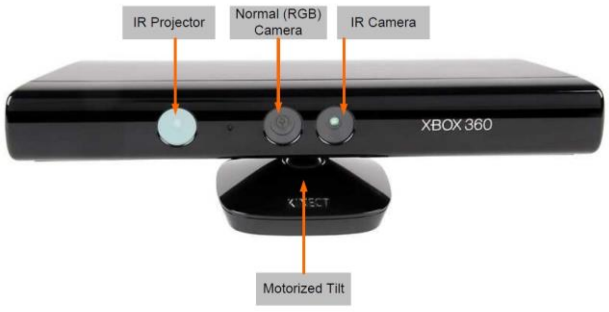


Figure 2.5 Kinect sensor

Figure 2.6 LIDAR sensor

2.3.3 Mapping method

The environment can be mapped in two ways:

- Manual:

  By adjacency matrix which is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph. This way is good for small environments, since it doesn’t require the robot to map itself, also the maze must be regular and cellular, and each cell is similar to each other. This method can be very exhausting and inaccurate for large and irregular environments.[18]

- Stochastic:

  This method is more complex, although it needs an accurate sensor for scanning and high processing capabilities. It is more accurate, reliable, and it can scan large areas.[19]

  The latter method is chosen because the maze is variable, unspecified, and irregular.

2.3.4 Localization method

The robot must figure out its position inside the maze that it has already mapped.

The two suggested methods are:

- Particle filter (Monte Carlo localization) [20]
- Markov localization [21]

The first method is chosen because it is more accurate, easier to be implemented, and it does not require high memory space compared to the second one. However, it has high computational complexity.

2.3.5 Optimal path calculation method

The best way to choose the best path is by using Dijkstra's algorithm, because it always finds the shortest path and it does not require a lot of time. On the other hand it requires a lot of space. Dijkstra's algorithm is considered greedy since it goes for the closest node hoping to find the shortest path. [18]

2.3.6 Object detection and localization method

There were two devices to do object detection that were suggested:

1) Webcam
2) Kinect

Kinect is better than the webcam because Kinect has depth sensor and it's compatible with ROS.[22]

2.4 Relationship among components

Figure 2.7 shows a schematic of the conceptual design, explaining the relationship between the system components.

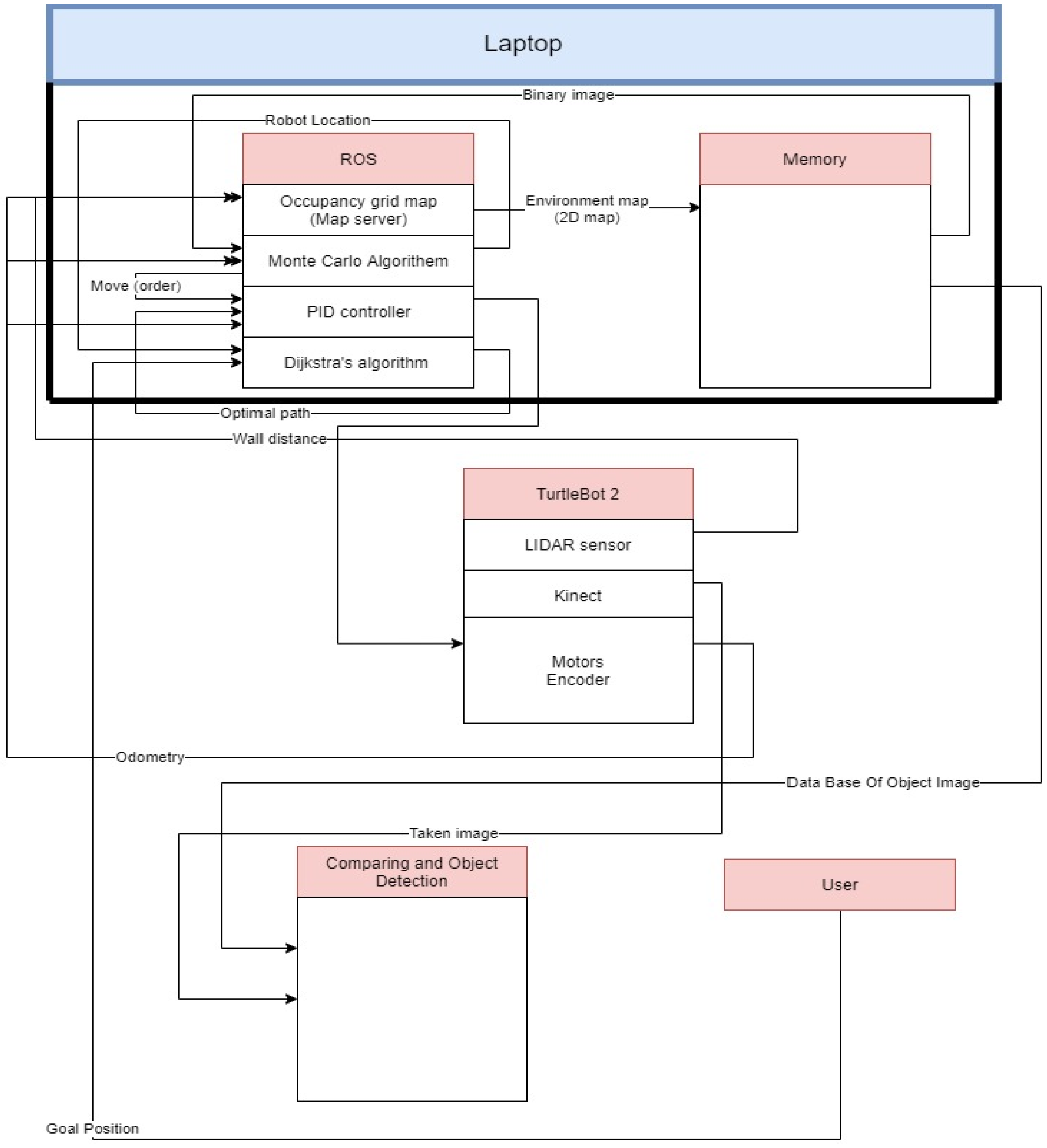


Figure 2.7 Conceptual Design Schematic

Chapter 3 **Robot Navigation**

3.1 Introduction

Robot navigation means the robot’s ability to determine its own position in a frame of reference and then to plan a path towards the goal location. In order to navigate in its environment, the robot requires a representation of the environment and the ability to interpret that representation.[23]

In this project, we divided our robot’s navigation problem into three sub-problems, mapping, localization, and finding the optimal path for navigation. The following sections present each one of them in details.

3.2 Mapping

For the robot to localize itself inside an environment, it needs to have a map of that environment. And because we don’t have maps for all environments, we need a way to make this map and save it, so the robot can refer to it when it navigates. The process of making a map for an environment is called mapping.[24]

Mapping with mobile robots is not an easy task, because maps are defined over a continuous space, resulting in a huge space of possible maps. Even using discrete approximations, such as the grid approximation, could result in more than 10^5 variables to describe a map.[24]

Of course, the difficulty level is not the same for all mapping problems. There are some factors that affect the difficulty of mapping problems, below are the most important of them[24]:

- **Size:** The larger the environment, the more difficult it is to map.
- **Noise in perception and actuation:** Mapping would be very easy if there was no noise in the robot’s sensors and actuators. Mapping gets more difficult if the noise is large.
- **Perceptual ambiguity:** If the environment has different places look alike, the mapping problem gets more difficult.
- **Cycles:** When the robot goes point to point through a path and returns in the same path, the odometric error is cancelled, but if the robot returns in a different path, the error could accumulate to a huge amount.

There are several methods used to map an environment. In this project, we are using Occupancy Grid Mapping method.[24]

Occupancy grid maps address the problem of generating consistent maps from noisy and uncertain measurement data, assuming that the robot pose is known. The occupancy grids represent the map as a field of random binary variables arranged in an evenly spaced grid, and each variable corresponds to the occupancy of the location it covers. Approximate posterior estimation is implemented by occupancy grid mapping algorithms for those random variables.[24]

Occupancy Grid Mapping Algorithm

The main concept when talking about occupancy grid mapping is the posterior probability over maps given the data from the sensors and the path of the robot, which can be represented by the equation below[24]:

$$p(m \mid z_{1:t}, x_{1:t}) \tag{3.1}$$

Where m is the map, $z_{1:t}$ is the set of measurements up to time t, and $x_{1:t}$ is the path of the robot, that is, the sequence of all its poses.[24]

The most common domain of occupancy grid maps are 2D floorplan maps, which describe a 2D slice of a 3D world. For a robot navigating on a flat surface, 2D maps are usually sufficient. Also, occupancy grid techniques can be generalized to 3D representations, but at significant computational expenses.[24]

Since the map is discretized over a finite number of cells, we can describe that in the following equation[24]:

$$m = \sum_i m_i \tag{3.2}$$

Where m_i denotes the grid cell with index i.

Each m_i contains a binary occupancy value, which specifies whether a cell is free or occupied. We are using ‘0’ for free cells and ‘1’ for occupied ones.[24]

The standard approach for occupancy grid mapping breaks down the problem of the whole map into a collection of binary cell problems with static states, each problem can be represented by equation (3.1) as follows[24]:

$$p(m_i \mid z_{1:t}, x_{1:t}) \tag{3.3}$$

The posterior probability over the whole map can be approximated as the product of the posterior probability of its cells[24]:

$$p(m \mid z_{1:t}, x_{1:t}) \cong \prod_i p(m_i \mid z_{1:t}, x_{1:t}) \tag{3.4}$$

To avoid numerical instabilities for probabilities near zero or one, occupancy grid mapping algorithm uses the log-odds representation of occupancy[24]:

$$l_{t,i} = \log \frac{p(m_i \mid z_{1:t}, x_{1:t})}{1 - p(m_i \mid z_{1:t}, x_{1:t})} \tag{3.5}$$

The probabilities are easily recovered from the log-odds ratio[24]:

$$p(m_i \mid z_{1:t}, x_{1:t}) = 1 - \frac{1}{1 + e^{l_{t,i}}} \tag{3.6}$$

Using Bayes rule on equation (3.5), and after simplifying we get:

$$l_{t,i} = \log \frac{p(m_i \mid z_{1:t}, x_{1:t})}{1 - p(m_i \mid z_{1:t}, x_{1:t})} = \log \frac{p(z_{1:t}, x_{1:t} \mid m_i = 1)}{p(z_{1:t}, x_{1:t} \mid m_i = 0)} + \log \frac{p(m_i = 1)}{p(m_i = 0)} \tag{3.7}$$

Which can be represented as:

$$\log odd^{+} = \log odd_{meas} + \log odd^{-} \tag{3.8}$$

Where odd^{+} is the new value of odds, odd^{-} is the previous value of odds, and odd_{meas} is the update in the odds based on the current measurement.

Note that the expression $p(m_i)$ or $p(m_i = 1)$ refers to the probability that a grid cell is occupied.[24]

In ROS, occupancy grid mapping can be used using the command ‘gmapping’.[25]

Auto Mapping

Instead of moving the robot manually inside an environment, ROS gives the ability to let the robot explore the environment by itself using the Rapidly-Exploring Random Tree algorithm (RRT), and by using gmapping with it, the robot gets the ability to map the environment automatically without any assessments.

3.3 Localization

Localization is the process of establishing correspondence between the map coordinate system and the robot’s local coordinate system. It is the main perceptual problem in robotics, since all robotics tasks require knowledge of the location of the robots and the objects being manipulated, although not necessarily within a global map.[24]

Mobile robot localization (or sometimes called position estimation or position tracking) is the problem of determining the pose of the robot relative to a given map of the environment. Mobile robot localization is partial problem of the general localization problem.[24]

Localization could be considered as a problem of coordinate transformation, since maps are described in a global coordination system, which is independent of the pose of the robot. Knowing this coordinate transformation is a necessary prerequisite for robot navigation, because it enables the robot to express the location of objects of interests within its own coordinate frame. This coordinate transformation can be determined by knowing the pose $\boldsymbol{x}_t = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$ of the robot (position in x, y and orientation around z), which is sufficient assuming it is expressed in the same coordinate frame as the map.[24]

The main problem in getting the pose is that most sensors do not give a noise-free measurement, also sometimes the pose cannot be sensed directly. Because of that, the robot has to integrate data over time to determine its pose, since a single sensor measurement is usually insufficient to determine the pose.[24]

Localization problems are not at the same level of difficulty, they depend on the type of knowledge that is available initially and the run time. There are mainly three types of localization problems with an increasing level of difficulty[24]:

- **Position tracking.** Position tracking assumes that the initial pose of the robot is known, and that the pose error is small because the effect of such noise is usually small, so localizing the robot can be achieved by accommodating the noise in the robot’s motion. The pose uncertainty is often approximated by a unimodal distribution such as the Gaussian distribution. Since the uncertainty is local and confined to region near the robot’s true pose, the position tracking problem is considered a local problem.
- **Global localization.** In this problem, the initial pose of the robot is unknown. The robot is placed somewhere in the environment, but is lacks information of where it is. Some

approaches to global localization does not assume boundedness of the pose error. Global localization is more difficult than position tracking, and unimodal probability distribution is usually inappropriate. In fact, global localization subsumes the position tracking problem.

- **Kidnapped robot problem.** In this problem, the robot can get kidnapped and teleported to some other location during operation. This problem is a variant of the global localization problem, but it is even more difficult because in global localization, the robot knows that it doesn't know where it is, but in the kidnapped robot problem, the robot might believe it knows where it is while it does not. One might argue that robots are rarely kidnapped in practice; however, the practical importance of this problem lies in the fact that most localization algorithms cannot be guaranteed never to fail. This problem gives the robot the ability to recover from failures which is essential for truly autonomous robots. Localization algorithms can be tested by kidnapping to measure its ability to recover from global localization failures.

The above problems can be solved using Monte Carlo Localization algorithm.

3.3.1 Monte Carlo Localization (MCL)

Monte Carlo localization algorithm is arguably the most popular localization approach to date. It uses particle filters to estimate posterior probabilities over robot poses.[24]

This localization algorithm is applicable to both local and global localization problems. MCL is easy to implement, and tends to work well across a broad range of localization problems, which explains why it has become one of the most popular localization algorithms in robotics.[24]

The basic MCL algorithm represents the belief $bel(x_t)$ [1] by a set of M particles $X_t = \left\{x_t^{[1]},\ x_t^{[2]},\ \ldots\ ,x_t^{[M]}\right\}$. The initial belief $bel(x_0)$ is obtained by randomly generating M such particles from the prior distribution $p(x_0)$, and then assigning the uniform importance factor to each particle. The algorithm then samples one particle at a time from the motion model, using particles from present belief as starting points. The measurement model is then applied to determine the importance weight of each particle.[24]

Figure 3.1 shows a one-dimensional hallway example of MCL. First, a set of pose particles is drawn randomly and uniformly over the entire pose space as initialization for the algorithm as shown in Figure 3.1a. As the robot senses the door, MCL assigns importance factors to each particle. The resulting particle set is shown in Figure 3.1b. The height in this figure represents the importance weight for each particle. Note that the set of particles in Figure 3.1a and Figure 3.1b are identical, the only thing modified by the measurement update are the importance weights.[24]

Figure 3.1c shows the particle set after resampling and after incorporating the robot motion. This phase results in a new set of particles which has a uniform importance weights, but with an increased number of particles near the three most probable places. With a new measurement, new importance weights are assigned non-uniformly to the particle set as shown in Figure 3.1d. You can see now that most of the cumulative probability mass is centered on the second door, which is the most likely location.[24]

[1] $bel(x_t) = p(x|z_{1:t})$ for binary static states.

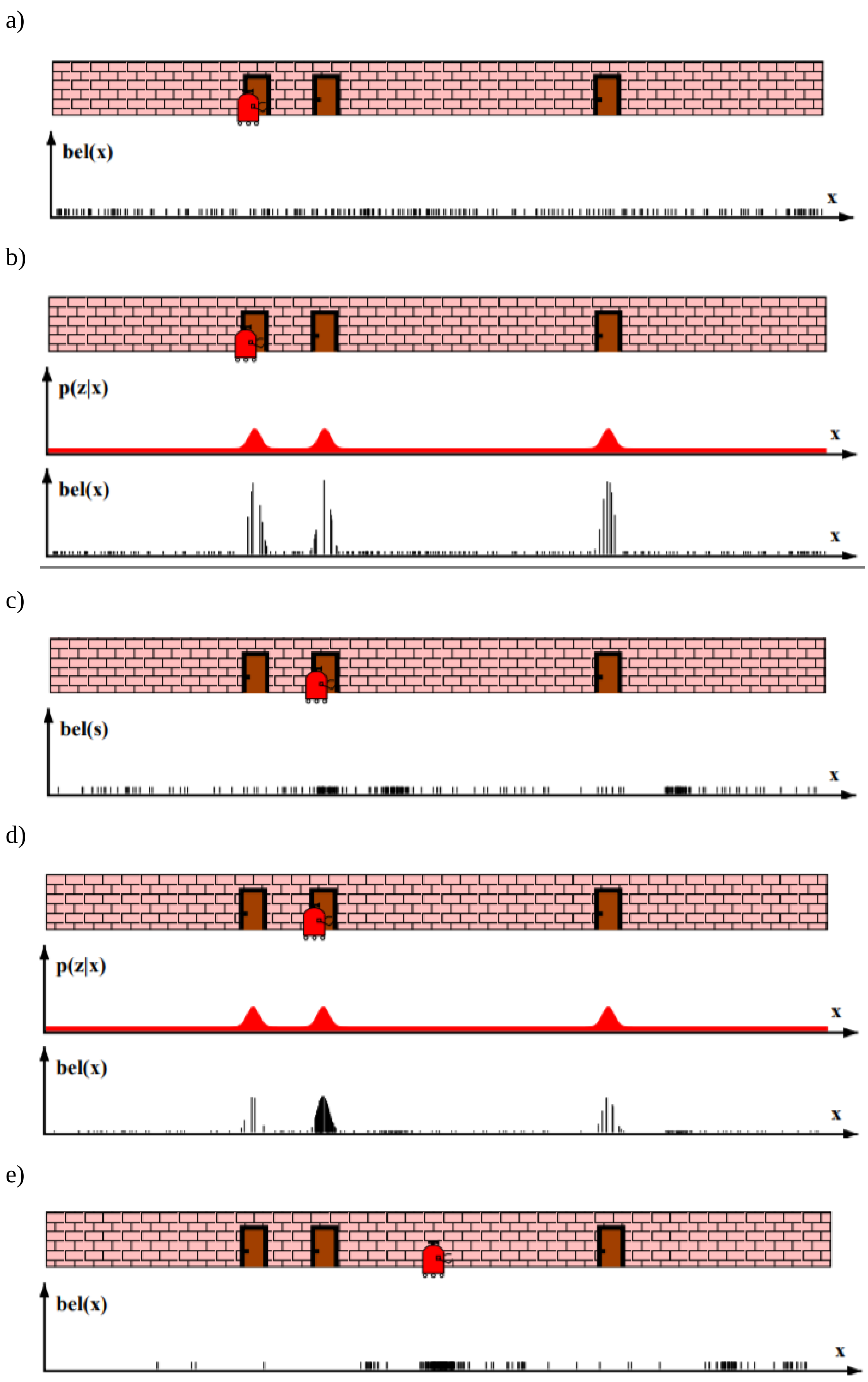


Figure 3.1 One-dimensional hallway example of MCL

The whole cycle is then repeated, when the robot moves, another sampling step happens, and a new particle set is generated, see Figure 3.1e.[24]

Properties of MCL

MCL can approximate almost any distribution of practical importance. It is not bound to a limited parametric subset of distributions. Increasing the total number of particles increases the accuracy of the approximation. The number of particles M is a parameter that enables the user to trade off the accuracy of the computation and the computational resources necessary to run MCL.[24]

A common strategy for setting M is to keep sampling until the next pair u_t and z_t has arrived. In this way, the implementation is adaptive with regards to the computational resources, the faster the processor, the better the localization algorithm. However, care has to be taken that the number of particles remains high enough to avoid filter divergence.[24]

A final advantage of MCL pertains to the non-parametric nature of the approximation. MCL can represent complex multi-modal probability distributions, and blend them seamlessly with focused Gaussian-style distributions.[24]

3.3.2 Recovery from failures

In its present form, MCL solves the global localization problem but cannot recover from robot kidnapping, or global localization failures. In places other than the most likely pose, the particles eventually disappear. At this point, the particles converge near a single pose, and the algorithm is unable to recover if this pose happens to be incorrect.[24]

MCL, like any other stochastic algorithms, may accidentally discard all particles near the correct pose during the resampling step. This is a significant problem, and it gets more significant if the number of particles is small or if the particles are spread over a large volume.[24]

Fortunately, this problem can be solved by a rather simple heuristic. The idea of this heuristic is to add random particles to the particle sets. This injection of random particles can be justified mathematically by assuming that there is a small probability that the robot might get kidnapped, so a fraction of random states is generated in the motion model. Even if the robot does not get kidnapped, an additional level of robustness is added because of these random particles.[24]

This version of MCL with the ability to solve the kidnapped robot problem is called the Augmented or the Adaptive Monte Carlo Localization algorithm (AMCL).[24]

In ROS, we can use the AMCL algorithm using the node (amcl). We can also include the launch file (amcl_diff.launch) to start the node with a series of configured parameters. This configuration is the default and the minimum setting needed to make it work. The contents of the (amcl_diff.launch) launch file which shows the configured parameters with their default values are appended.[26]

For more information on the AMCL algorithm with ROS, refer to [20].

3.4 Optimal Path

In graph theory, the optimal path (shortest path) problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.[27]

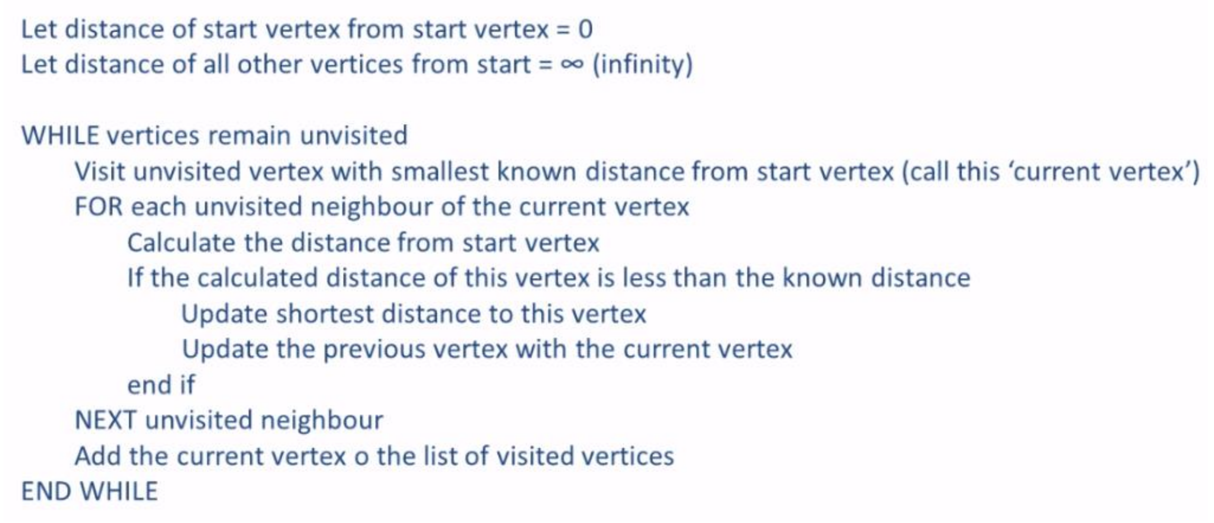
In this section, we are presenting an algorithm that can solve the optimal path problem, which is Dijkstra's Algorithm.

Dijkstra's Algorithm

Dijkstra's algorithm is an algorithm, which was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later, for finding the shortest paths between nodes in a graph, which may represent, for example, road networks.

The algorithm exists in many variants. Dijkstra's original algorithm found the shortest path between two given nodes, but a more common variant produces a shortest-path tree by finding the shortest paths to all nodes in the graph from one fixed source node.[28]

A pseudo code for the algorithm is shown in Figure 3.2.



```
Let distance of start vertex from start vertex = 0
Let distance of all other vertices from start = ∞ (infinity)

WHILE vertices remain unvisited
    Visit unvisited vertex with smallest known distance from start vertex (call this ‘current vertex’)
    FOR each unvisited neighbour of the current vertex
        Calculate the distance from start vertex
        If the calculated distance of this vertex is less than the known distance
            Update shortest distance to this vertex
            Update the previous vertex with the current vertex
        end if
    NEXT unvisited neighbour
    Add the current vertex o the list of visited vertices
END WHILE
```

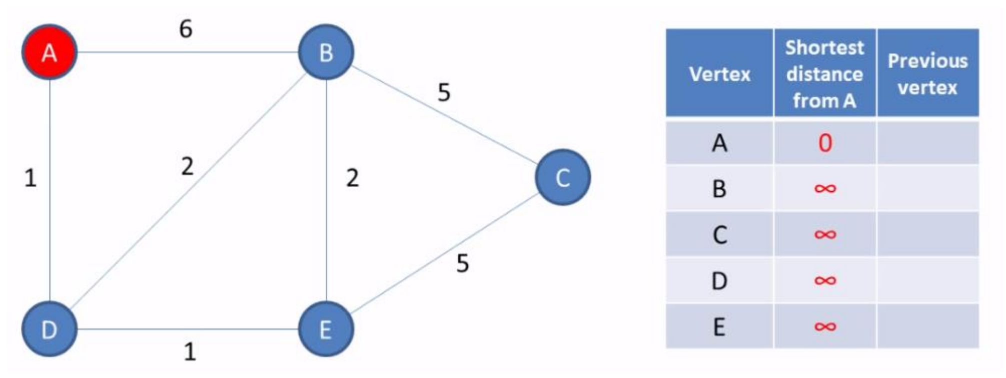
Figure 3.2 Pseudo code for Dijkstra's algorithm.[29]

Consider the following example shown in Figure 3.3, which illustrates the Dijkstra's Algorithm. Our objective is to find the shortest path from A to every other vertex. The table in the figure shows the shortest distance from A to every other vertex, and also it shows the shortest sequence of vertices from A to every other vertex, i.e. the shortest path.

Dijkstra's Algorithm works as follows, we use two lists, ‘visited’ (initially empty) and ‘unvisited’ (initially contains all vertices) to keep track of the vertices we have visited and the ones we haven’t visited yet.

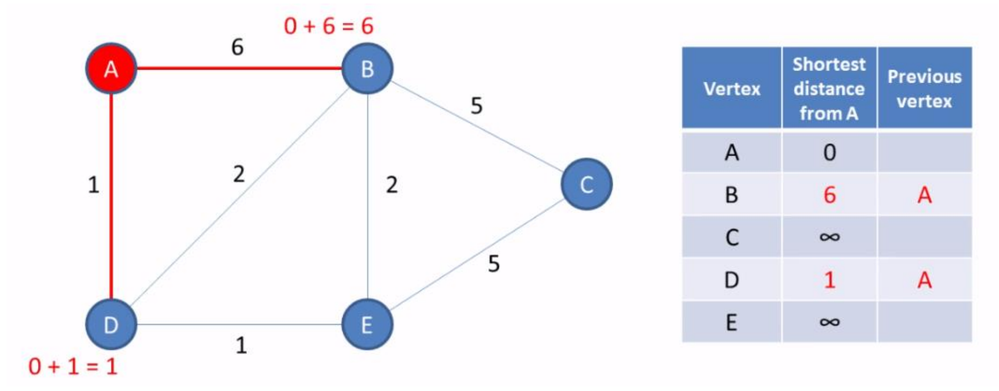
Initially, we set the distance from A to A to zero, and since the distances from A to all other vertices are unknown, we set them to a very high value (infinity), as shown in Figure 3.3a.

According to the algorithm in Figure 3.2, we visit the unvisited vertex with the smallest known distance from the start vertex, in this case, it’s A itself. For the current vertex, calculate the distance of each neighbor from the start vertex. In this case, the neighbors if A are B and D, and the distance from start to each one of them is 6 and 1 respectively. Now, if the calculated a)



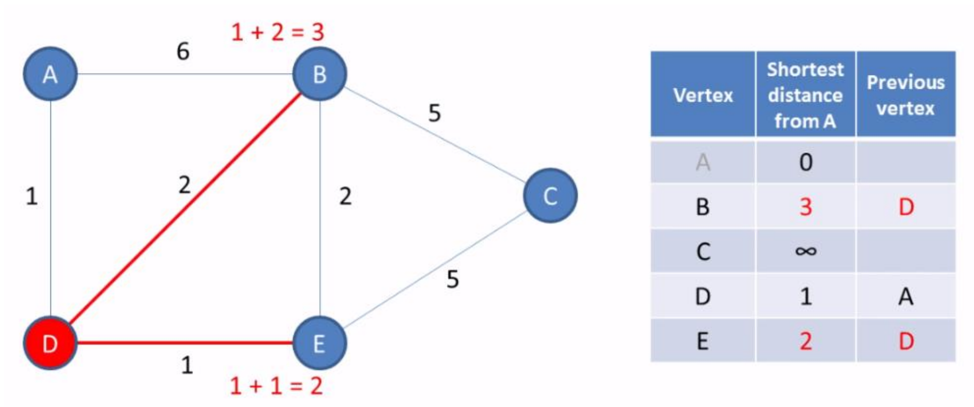
Vertex	Shortest distance from A	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	

b)



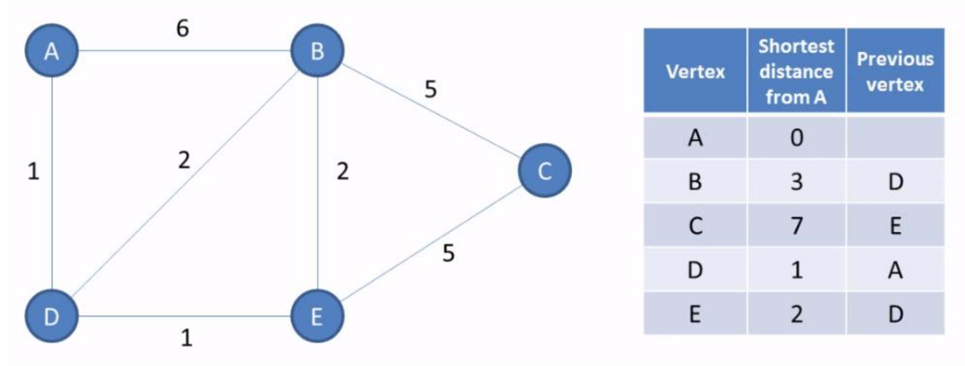
Vertex	Shortest distance from A	Previous vertex
A	0	
B	6	A
C	∞	
D	1	A
E	∞	

c)



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	∞	
D	1	A
E	2	D

d)



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	7	E
D	1	A
E	2	D

Figure 3.3 A simple example illustrating Dijkstra's Algorithm

distance is less than the current known distance, we update the shortest distance and the previous vertex with the current vertex. Since the previous distance value is larger for both B and D, we update the table as shown in Figure 3.3b.

Next we remove the current vertex (A) from the list of unvisited vertices, and we add it to the list of visited vertices, so we don’t visit it again.

The algorithms then repeats itself, we visit D since it has the current shortest distance between the unvisited vertices, and the same process is done as shown in Figure 3.3c.

After visiting all vertices, the table becomes as shown in Figure 3.3d. It shows the shortest distances between A and all other vertices, and it shows the previous vertex lead to the shortest distance.

Chapter 4 **Object Detection**

4.1 Introduction

For the final part of our project, Recognize and localize a specific object, Preparing to pick up it and bring it to a particular location within the map, Computer vision was chosen to detect the object by using You Only Look Once (YOLO) algorithm from images which will be taken by the camera.

The task was separated into two small sections:

- Detect object and move towards it.
- Perform the object's pose estimation with respect to the camera's coordinate frame.

Anaconda [30, 31] was used to detect the object with OpenCV [32] (Open Source Computer Vision Library), and NumPy library which is an open source library adding support for multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays [33].

4.2 YOLOv4 Custom Training

4.2.1 Introduction

The YOLO v4 release lists three authors: Alexey Bochkovskiy, the Russian developer who built the YOLO Windows version, Chien-Yao Wang, and Hong-Yuan Mark Liao.

Most of the modern accurate models require many GPUs for training with a large mini-batch size, and doing this with one GPU makes the training really slow and impractical. YOLO v4 addresses this issue by making an object detector which can be trained on a single GPU with a smaller mini-batch size. It is also YOLO v4 achieves state-of-the-art results at a real time speed on the MS COCO dataset with 43.5 % AP running at 65 FPS on a Tesla V100 GPU.

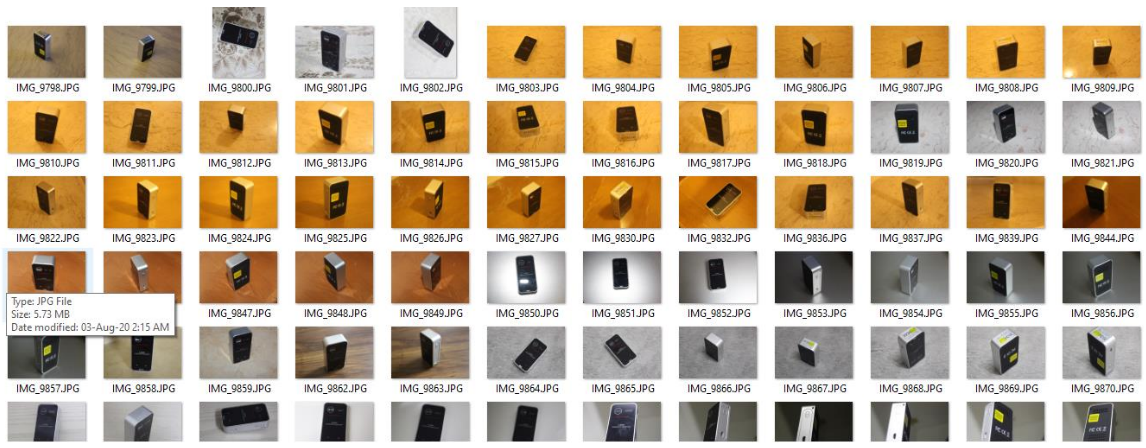
4.2.2 Preparing Darknet

There are very few implementations of the YOLO algorithm that exists on the web. The Darknet is one such open-source neural network framework written in C and CUDA and serves as the basis of YOLO. It is fast, easy to install, and supports CPU and GPU computation. Darknet is used as the framework for training YOLO, meaning it sets the architecture of the network. The first author of Darknet is the author of YOLO itself (J Redmon).

Download a darknet file for Yolo v4 from Alexey's GitHub account by Git tool, Git is an open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It was originally developed by Linus Torvalds in 2005 to ease the development of the Linux Kernel [34].

4.2.3 Collecting Data

1,200 images were taken of the target in different lights, angles and positions to increase accuracy as shown in Figure 4.1, 80% of them used for training and 20% as validation set for testing to evaluate the performance of the model.



Type: JPG File
Size: 5.73 MB
Date modified: 03-Aug-20 2:15 AM

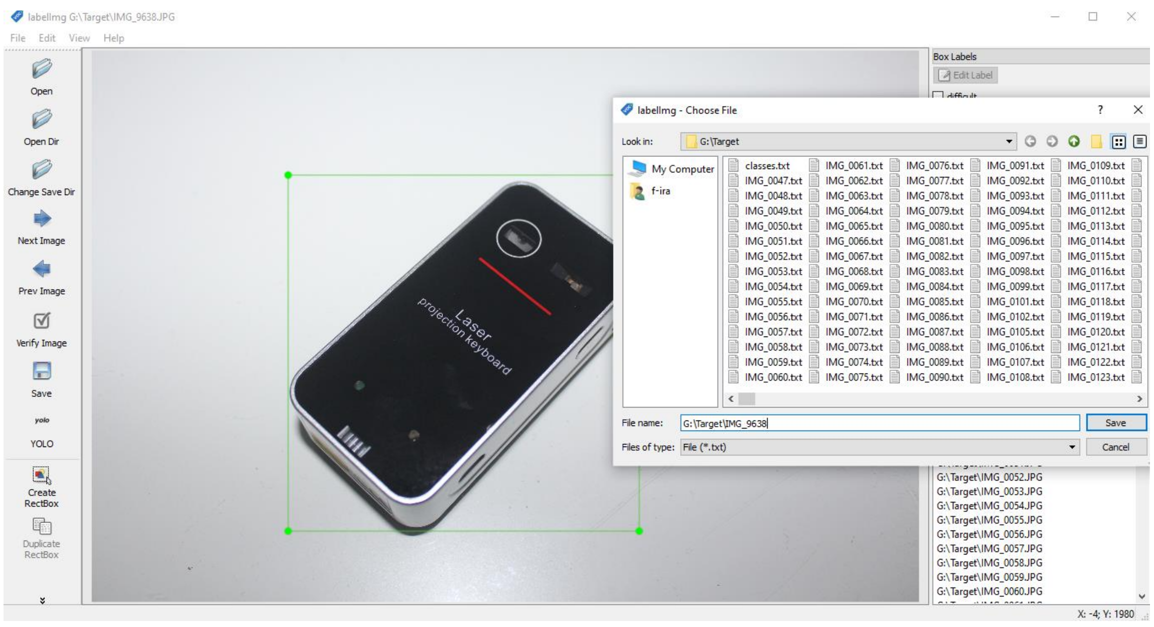
Figure 4.1 Images of Our Target

4.2.4 Labeling

One of the most important part in the data science process is data pre-processing which makes data scientists put lots of time and effort into. Particularly, in the context of machine learning, data pre-processing requires a step of labeling. This is the step to detect and label data samples into multi-classifications so that the labeled data can be used in the further machine learning process.

Labeling the Target in the images by labeling tool which is a graphical image annotation tool which provides images with bounding boxes after labeling and it supports YOLO format [34], with this tool, a circumferential rectangle is placed around the object manually image by image Such as in Figure 4.2.

The object has been named Target, then the label is saved as a text file containing the coordinates of the four corners of the rectangle.



File name: G:\Target\IMG_9638

Files of type: File (*.txt)

Figure 4.2 Labelling the Target

4.2.5 Google Colab

Google Colaboratory, or "Colab" is a widely popular cloud service for machine learning that features free access to GPU and TPU computing.

With Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just a few lines of code. Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware that works with Ubuntu 18.04.3 operating system, including GPUs and TPUs, regardless of the power of your machine. All you need is a browser.

As of October 13, 2018, Google Colab provides a single 12GB NVIDIA Tesla K80 GPU that can be used up to 12 hours continuously, and 68GB storage disk with 12.72GB of RAM, Recently, Colab also started offering free TPU.

There are several benefits of using Colab over using your own local machines. Some of the benefits of Colab are:

- It's not required to do an environment setup, because it comes with important packages pre-installed and ready to use.
- Provides browser-based Jupyter Notebooks.
- Free Cloud service with free GPU
- Store Notebooks on Google Drive
- Importing Notebooks from Github
- Document code with Markdown
- Load Data from Drive

- Colab is used extensively in the machine learning community with applications including:
- Getting started with TensorFlow
- Developing and training neural networks
- Disseminating AI research
- Creating tutorials

The darknet file is compressed after adding the data file to it and uploading it to Google Drive, then a Google custom file is created.

4.2.6 Start Training

First, we have to connect the custom file with Google servers, and change the hardware accelerator of runtime to GPU to use the google colab in a GPU mode, after that mount Google Drive files to google colab.

Before we start the model training process, at the beginning we have to Setup custom darknet environment by following these steps by using some Linux commands, since the server is running on Linux operating system:

- Update repository list
- Extract the uploaded darknet file into the runtime
- Go to darknet directory
- Change permission of darknet of folder to executable
- Compile darknet framework
- Remove default backup folder of darknet if exist
- Create asymbolic link to save the wight directly into google drive backup to get a copy of models on our Google Drive for fear of losing data or a sudden internet interruption
- Start training with arguments

The training process includes continuous several hours depending on the amount of data present, this chart in Figure 4.3 shows the overall training loss over time during learning, and validation mean average precision (mAP) against iteration number for the training, pitted against iterations. As seen in the graph the average loss flattens out past the 1200 mark, and In Figure 4.4, the outcome of learning process.

After completing the training process, a weight file is downloaded from backup file in google drive, and the configuration file is used with it to detect the object.

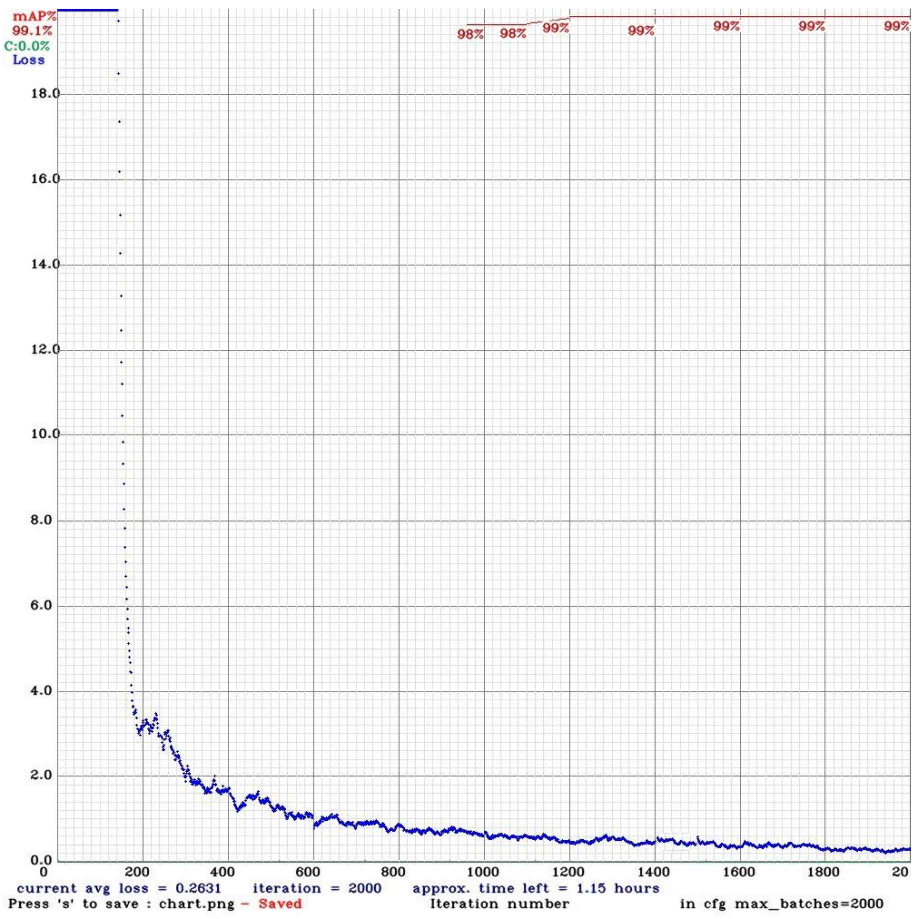


Figure 4.3 Plot of Average Training Loss and Overall Validation mAP Against Number of Iterations

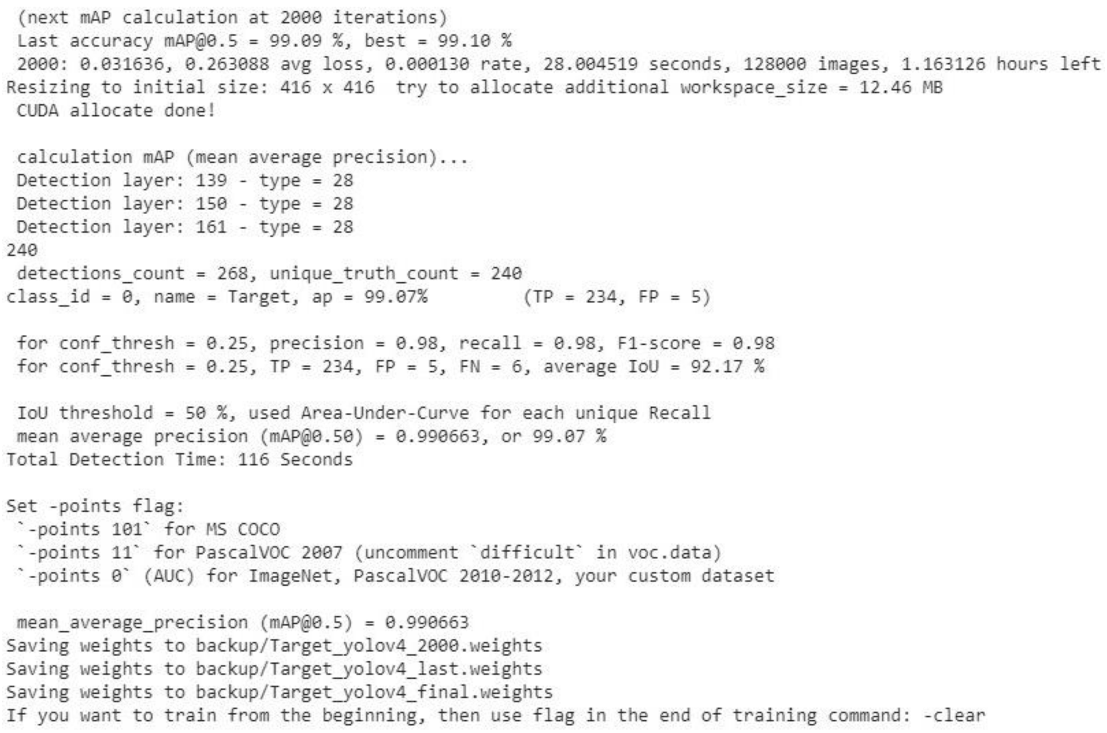
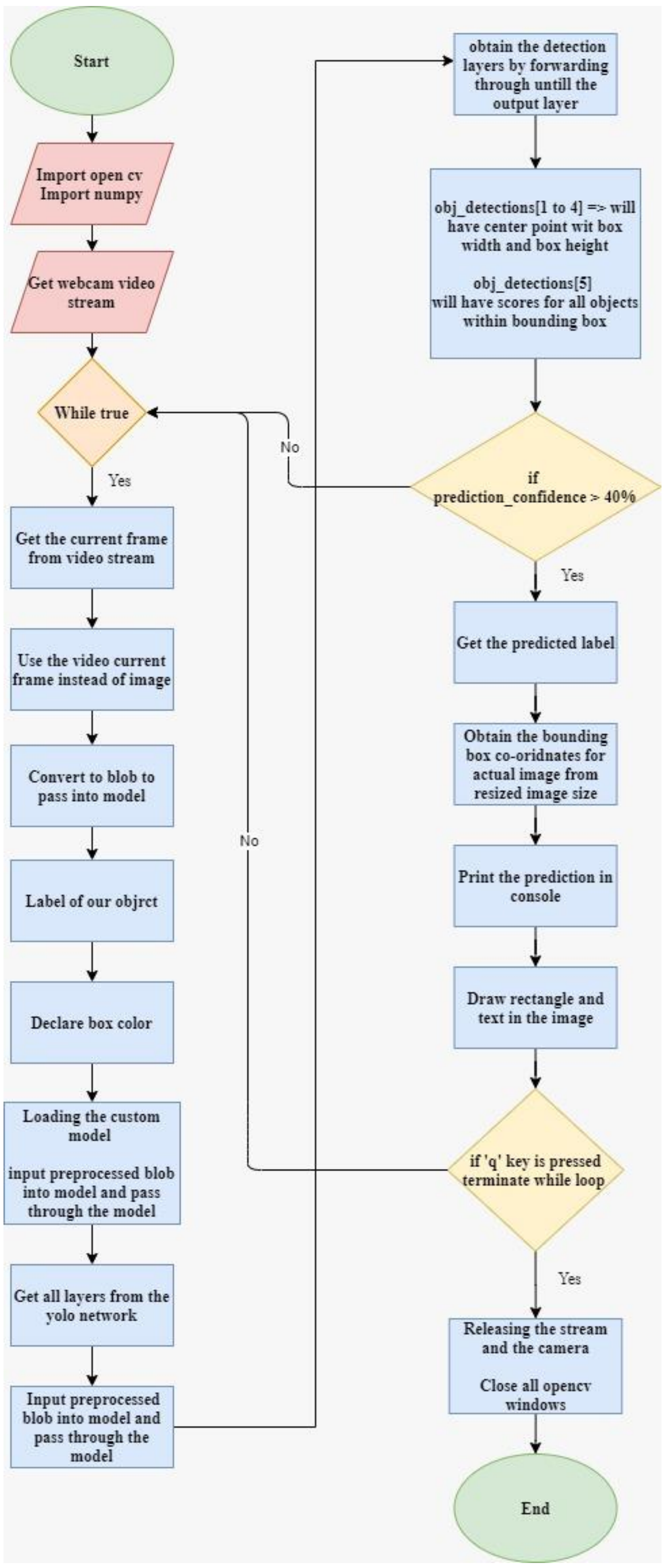

```
 (next mAP calculation at 2000 iterations)
 Last accuracy mAP@0.5 = 99.09 %, best = 99.10 %
 2000: 0.031636, 0.263088 avg loss, 0.000130 rate, 28.004519 seconds, 128000 images, 1.163126 hours left
Resizing to initial size: 416 x 416  try to allocate additional workspace_size = 12.46 MB
 CUDA allocate done!

 calculation mAP (mean average precision)...
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
240
 detections_count = 268, unique_truth_count = 240
class_id = 0, name = Target, ap = 99.07%         (TP = 234, FP = 5)

 for conf_thresh = 0.25, precision = 0.98, recall = 0.98, F1-score = 0.98
 for conf_thresh = 0.25, TP = 234, FP = 5, FN = 6, average IoU = 92.17 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.990663, or 99.07 %
Total Detection Time: 116 Seconds

Set -points flag:
 `-points 101` for MS COCO
 `-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
 `-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset

 mean_average_precision (mAP@0.5) = 0.990663
Saving weights to backup/Target_yolov4_2000.weights
Saving weights to backup/Target_yolov4_last.weights
Saving weights to backup/Target_yolov4_final.weights
If you want to train from the beginning, then use flag in the end of training command: -clear
```

Figure 4.4 Outcome of Learning Process

4.3 Object Detection

Object identification with anaconda is done by the flowchart below in Figure 4.5.



Start
Import open cv
Import numpy
Get webcam video stream
While true
Yes
Get the current frame from video stream
Use the video current frame instead of image
Convert to blob to pass into model
Label of our objrct
Declare box color
Loading the custom model
input preprocessed blob into model and pass through the model
Get all layers from the yolo network
Input preprocessed blob into model and pass through the model
obtain the detection layers by forwarding through untill the output layer
obj_detections[1 to 4] => will have center point wit box width and box height
obj_detections[5] will have scores for all objects within bounding box
if prediction_confidence > 40%
No
Yes
Get the predicted label
Obtain the bounding box co-oridnates for actual image from resized image size
Print the prediction in console
Draw rectangle and text in the image
if 'q' key is pressed terminate while loop
No
Yes
Releasing the stream and the camera
Close all opencv windows
End

Figure 4.5 Object Detection Flowchart

Chapter 5 **Experiments and Results**

5.1 Introduction

This chapter concludes the experements done using Gazebo simulator for the different parts of this project and shows the yeilded results from these experements.

We have done experements on 10 randomly generated mazes with a constrained size between 2×2 and 10×10 cells wise, and the cell being a 2m×2m square.

5.2 Mapping

We built the 10 mazes on Gazebo, then the robot mapped each of them using the auto mapping method explained in section 3.2. Figure 5.1 is a block diagram generated by ROS using the command (rqt_graph), that shows all the nodes and topics running by ROS while mapping the environments. The nodes and topics that are most related to our project are highlighted in red.

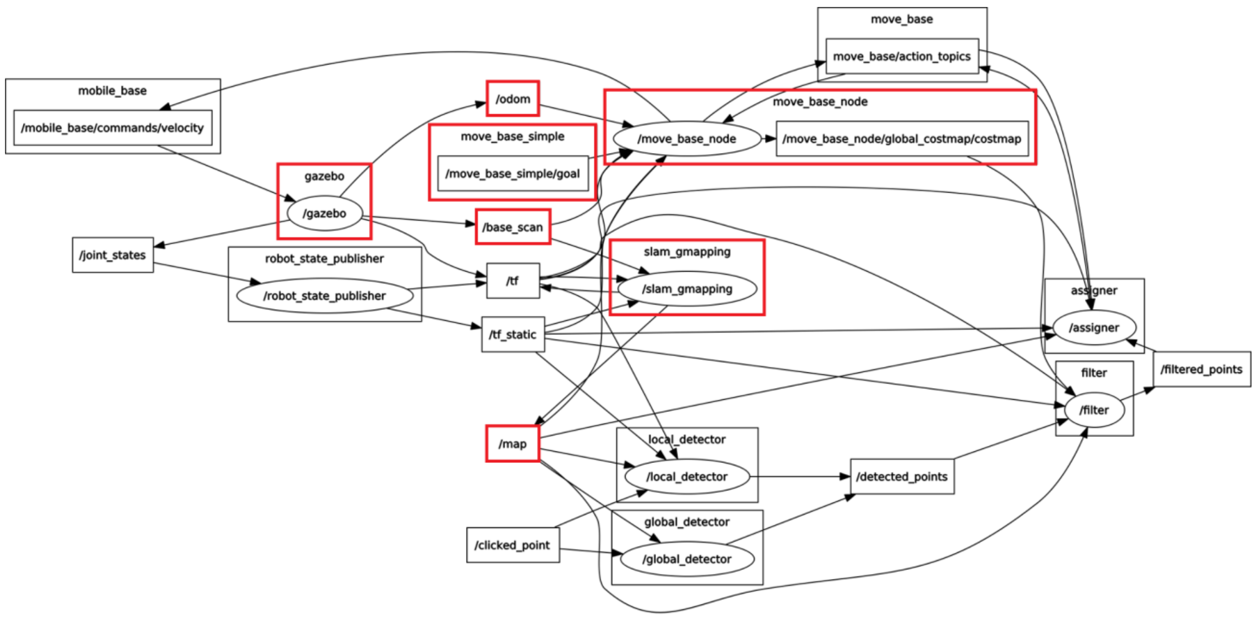
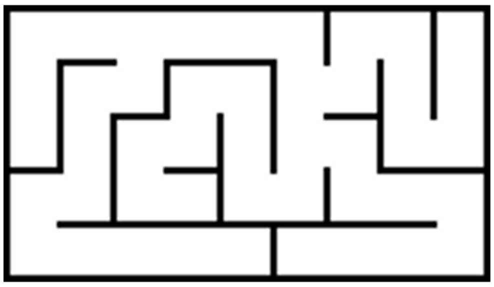
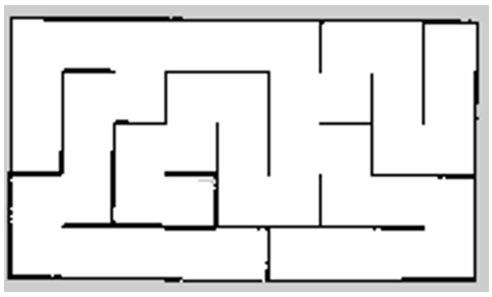
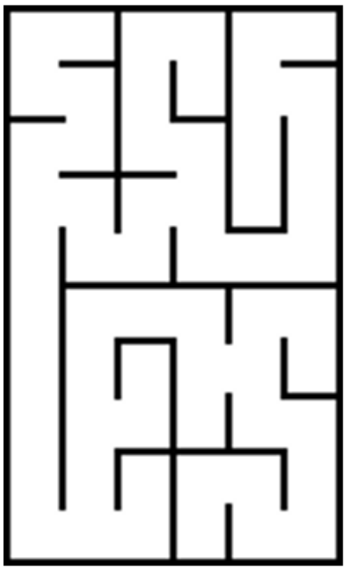
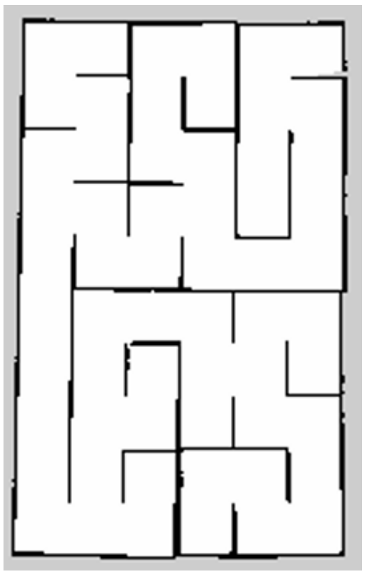
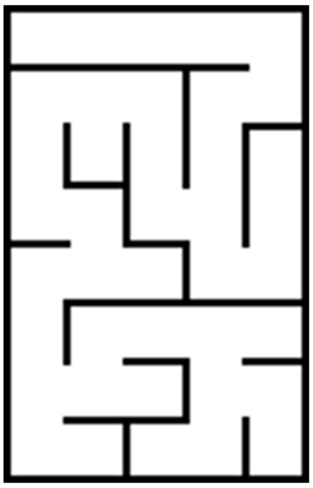
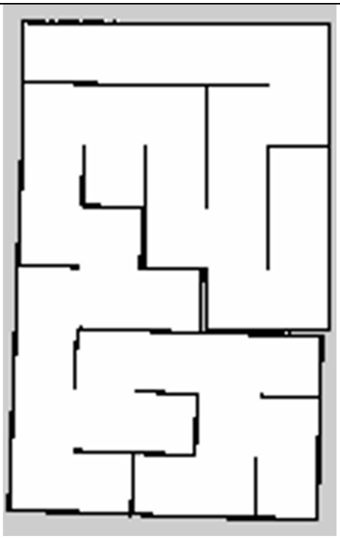
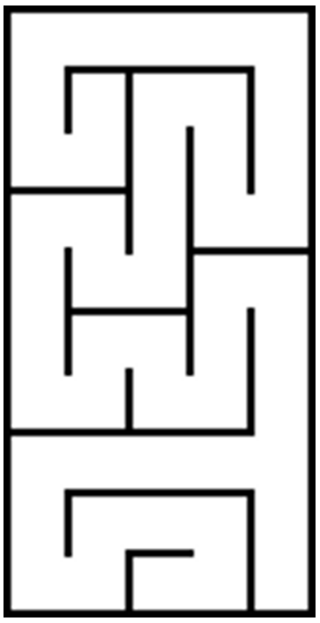
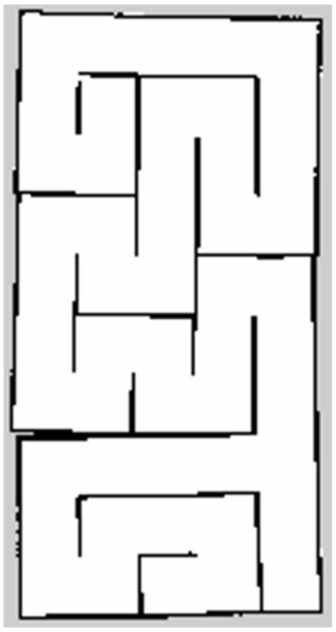
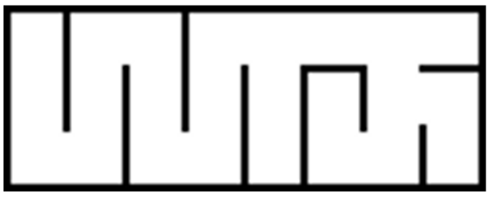
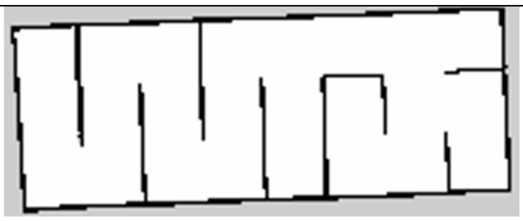
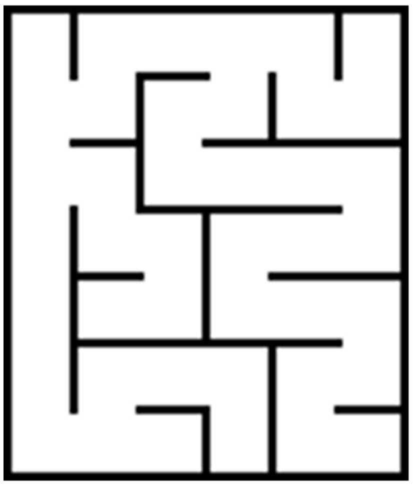
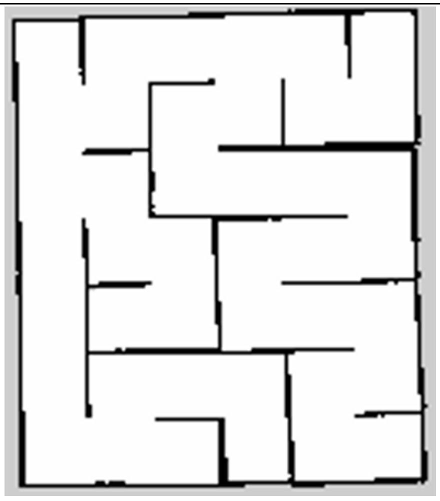


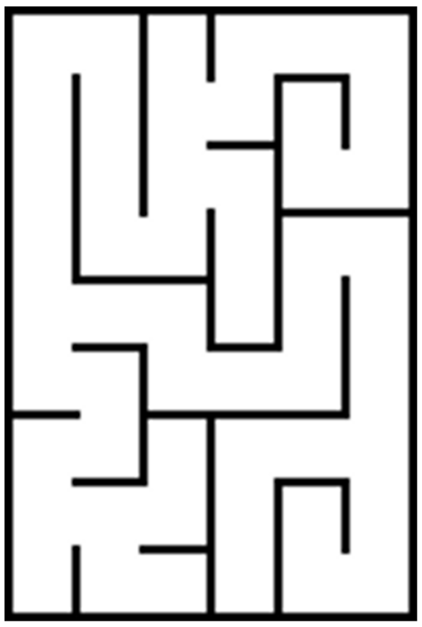
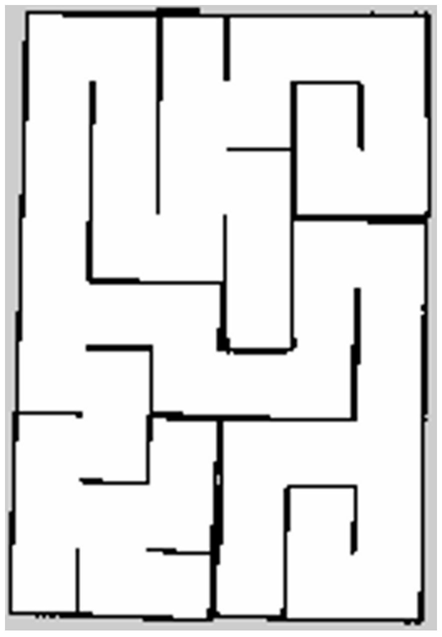
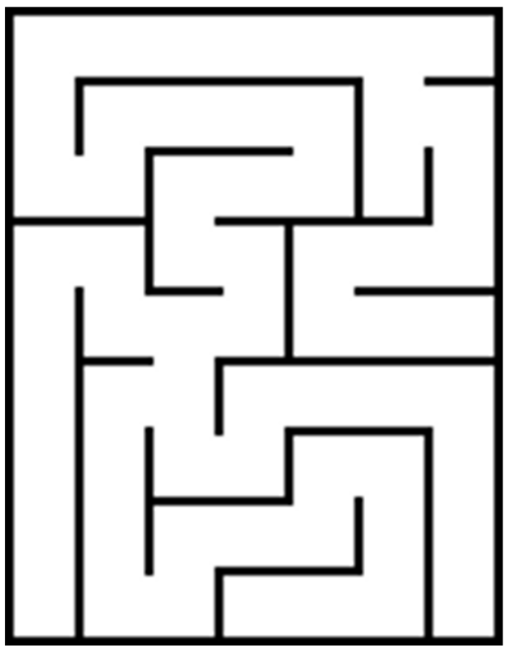
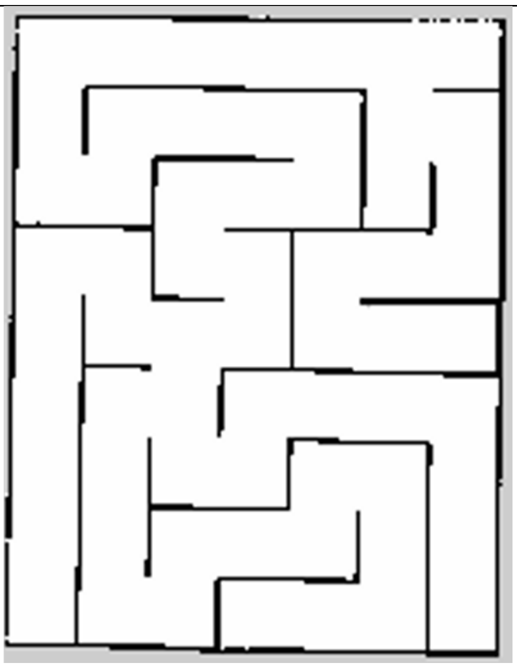
Figure 5.1 A Block Diagram of All the Nodes and Topics Running by ROS During the Mapping Phase

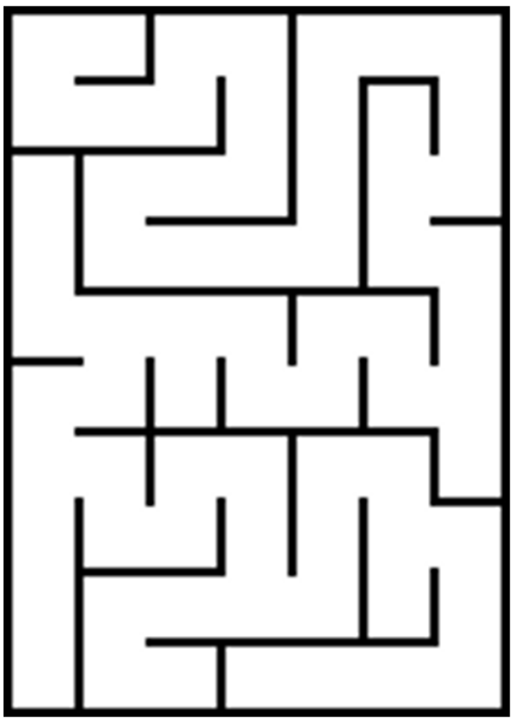
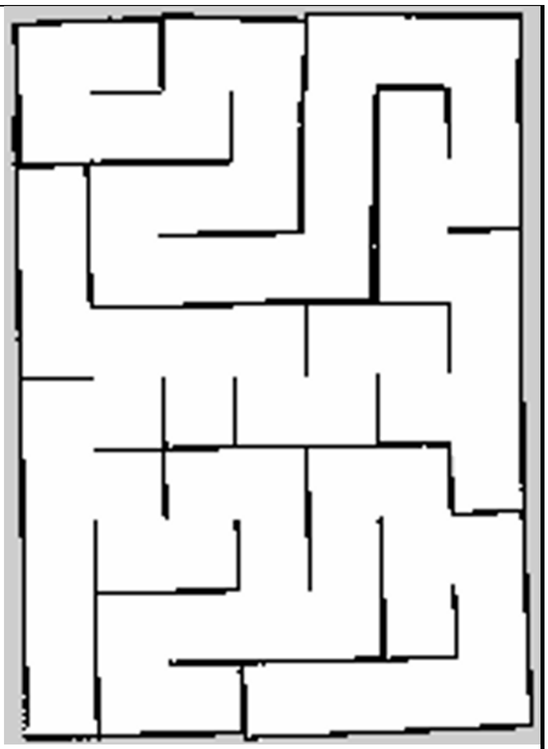
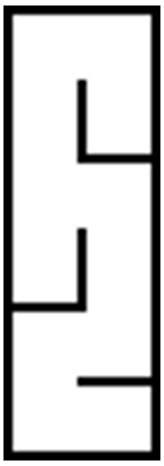
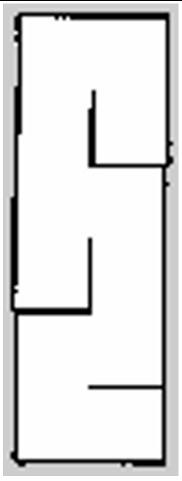
The mapping results are represented in Table 5.1.

Table 5.1 The Experimented Mazes and the Resulting Maps

Maze Number	Reference Maze	Resulting Map
1		
2		
3		

4		
5		
6		

7		
8		

9		
10		

The above results verify that the mapping subsystem works as intended.

5.3 Localization and Navigation

For each one of the generated mazes, we did 10 localization experements for a total of 100 experements where, for each experement, we placed the robot in a random initial pose inside the maze, and gave it a random goal pose, while providing the respective map.

Figure 5.2 is a block diagram generated by ROS using the command (rqt_graph), that shows all the nodes and topics running by ROS during the localization and navigation phase. The nodes and topics that are most related to our project are highlighted in red.

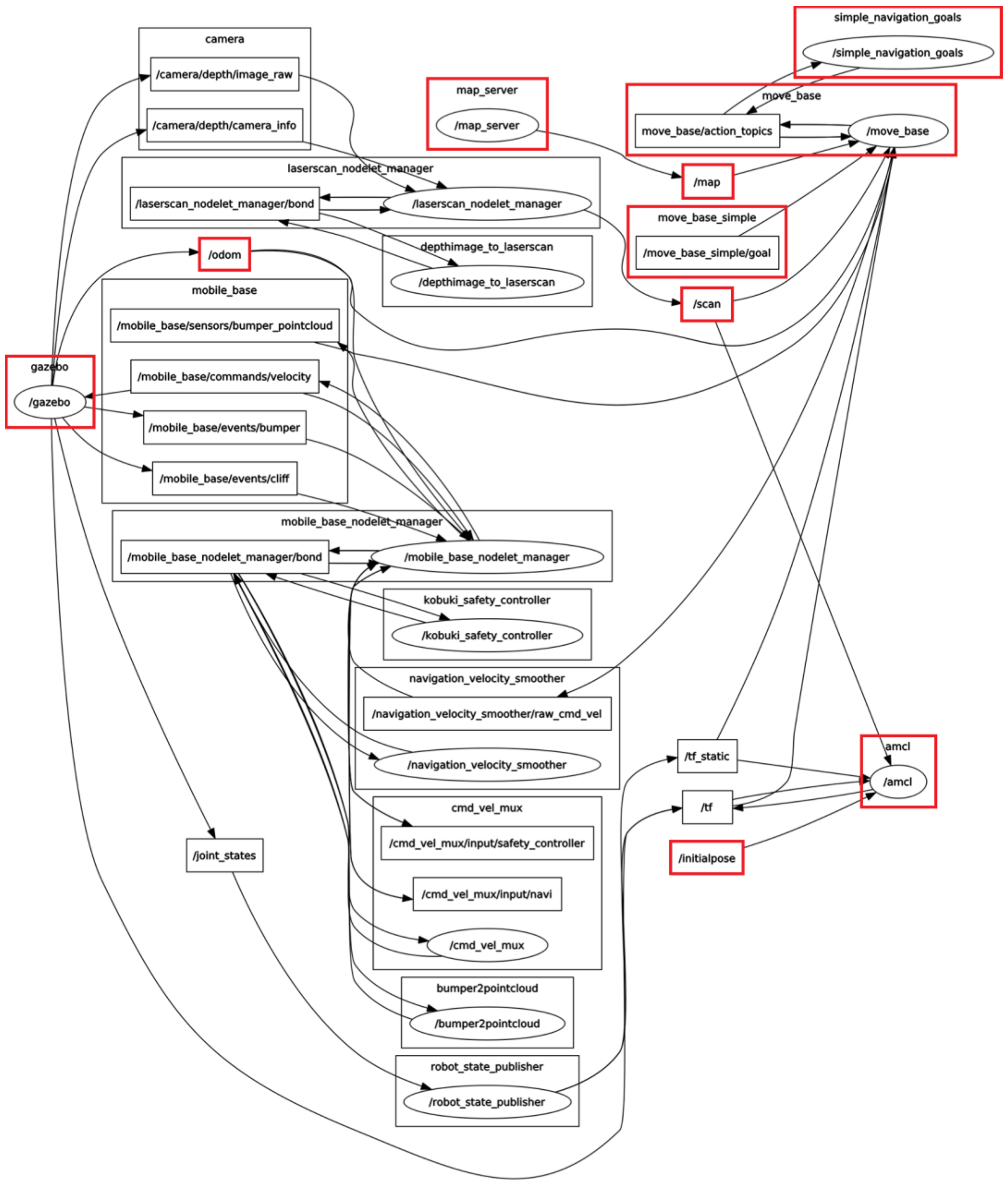

camera
/camera/depth/image_raw
/camera/depth/camera_info
map_server
/map_server
simple_navigation_goals
/simple_navigation_goals
move_base
move_base/action_topics
/move_base
laserscan_nodelet_manager
/laserscan_nodelet_manager/bond
/laserscan_nodelet_manager
/map
move_base_simple
/move_base_simple/goal
depthimage_to_laserscan
/depthimage_to_laserscan
/odom
/scan
mobile_base
/mobile_base/sensors/bumper_pointcloud
/mobile_base/commands/velocity
/mobile_base/events/bumper
/mobile_base/events/cliff
gazebo
/gazebo
mobile_base_nodelet_manager
/mobile_base_nodelet_manager/bond
/mobile_base_nodelet_manager
kobuki_safety_controller
/kobuki_safety_controller
navigation_velocity_smoother
/navigation_velocity_smoother/raw_cmd_vel
/navigation_velocity_smoother
/tf_static
/tf
amcl
/amcl
/initialpose
/joint_states
cmd_vel_mux
/cmd_vel_mux/input/safety_controller
/cmd_vel_mux/input/navi
/cmd_vel_mux
bumper2pointcloud
/bumper2pointcloud
robot_state_publisher
/robot_state_publisher

Figure 5.2 A Block Diagram of All the Nodes and Topics Running by ROS During the Localization and Navigation Phase

In each experement the robot navigates while trying to localize itself until it stops with one of the following three cases:

Case 1: The robot succeeds to localize itself and reaches the goal

Case 2: The robot fails to localize itself and gives up

Case 3: The robot localizes itself in a wrong position and thinks it succeeded

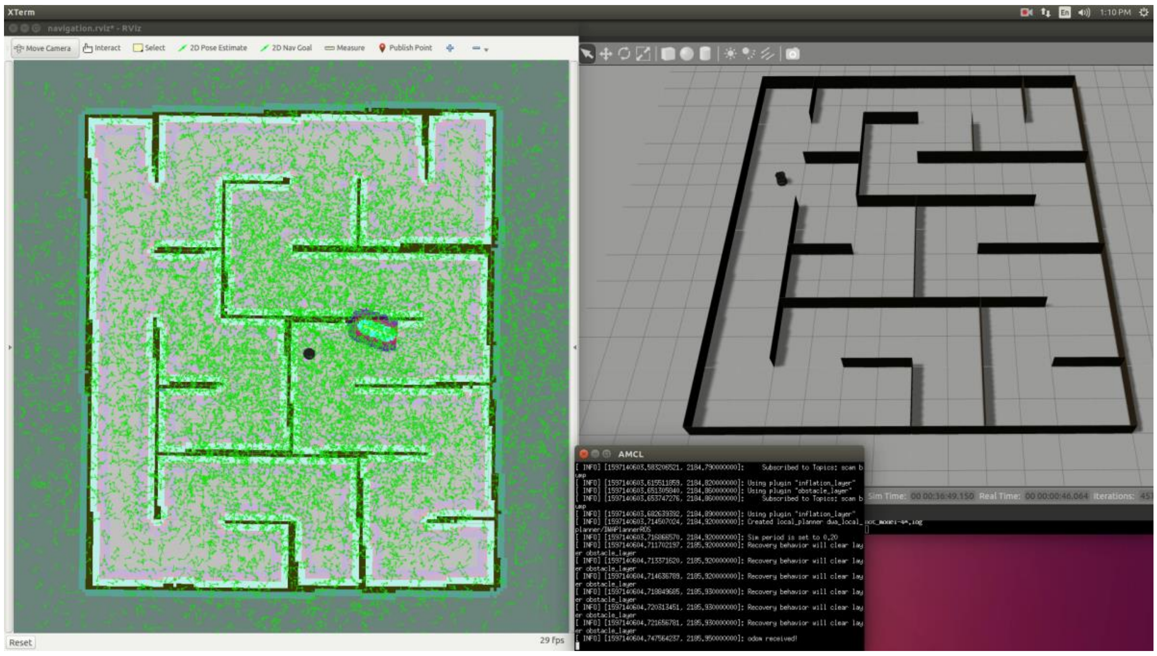
For each experiment, we recorded the case and measured the time of localization and navigation for the successful ones, the results are represented in Table 5.2.

Table 5.2 Localization and Navigation Experemental Results

Maze / Case	1	2	3	4	5	6	7	8	9	10	Total
Case 1	7	6	5	7	5	7	6	3	8	8	62
Case 2	0	2	1	2	0	0	1	1	2	1	10
Case 3	3	2	4	1	5	3	3	6	0	1	28
Success Rate	70%	60%	50%	70%	50%	70%	60%	30%	80%	80%	62%
Avg Time (sec)	170	87	185	205	58	138	135	282	144	61	139

Below is a timestamp representation of a successful localization attempt by the robot on maze #6 with T being the time when the goal pose was sent. On the figures, the right window shows the simulated environment by Gazebo, and the left window is Rvis showing multiple things including the map, a virual robot navigating inside it, a highlight of what the robot is scanning, and the particles of the particle filter.

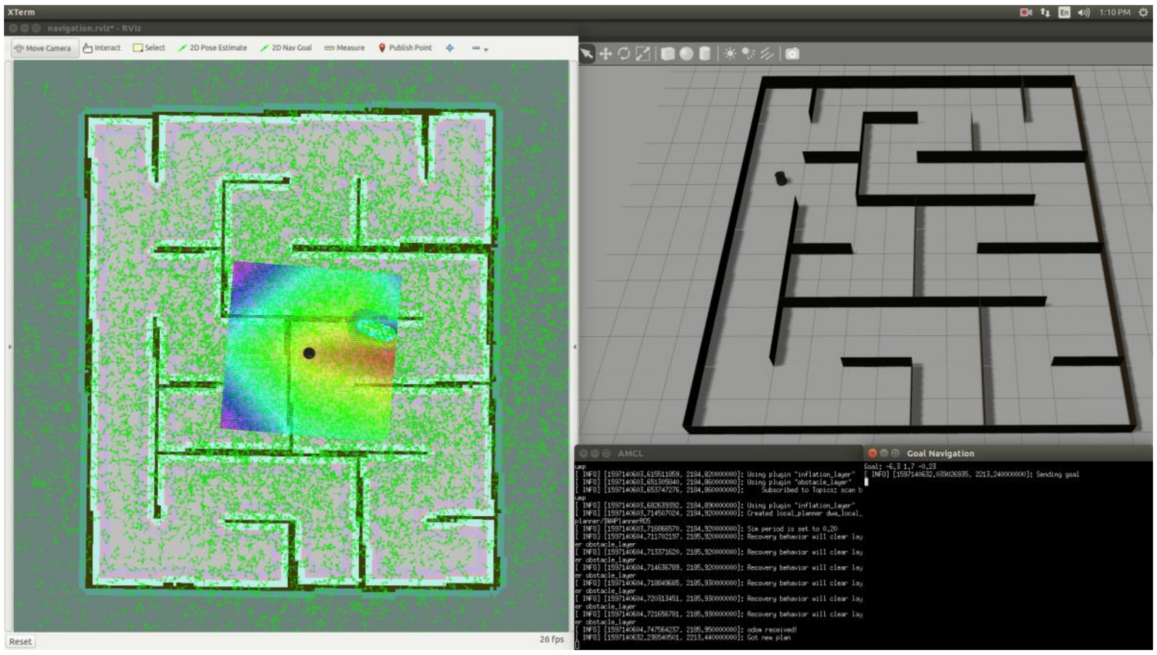
- At T-4, the goal pose has not been sent to the robot yet, the robot is placed in a random pose, the initial pose of the virtual robot is as close to the center of the map as possible, and the particles are distributed across the whole map as shown in Figure 5.3.



```
[ INFO] [1597140603.583206521, 2184.790000000]:     Subscribed to Topics: scan b
ump
[ INFO] [1597140603.615511859, 2184.820000000]: Using plugin "inflation_layer"
[ INFO] [1597140603.651305840, 2184.860000000]: Using plugin "obstacle_layer"
[ INFO] [1597140603.653747276, 2184.860000000]:     Subscribed to Topics: scan b
ump
[ INFO] [1597140603.682639392, 2184.890000000]: Using plugin "inflation_layer"
[ INFO] [1597140603.714507024, 2184.920000000]: Created local_planner dwa_local_
planner/DWAPlannerROS
[ INFO] [1597140603.716866570, 2184.920000000]: Sim period is set to 0.20
[ INFO] [1597140604.711702197, 2185.920000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.713371620, 2185.920000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.714636789, 2185.920000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.718849685, 2185.930000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.720313451, 2185.930000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.721656781, 2185.930000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.747564237, 2185.950000000]: odom received!
```

Figure 5.3 Localization Status at T-4

- At T, the goal pose has just been sent, the global plan has been set, and the robot is about to start navigating as shown in Figure 5.4.

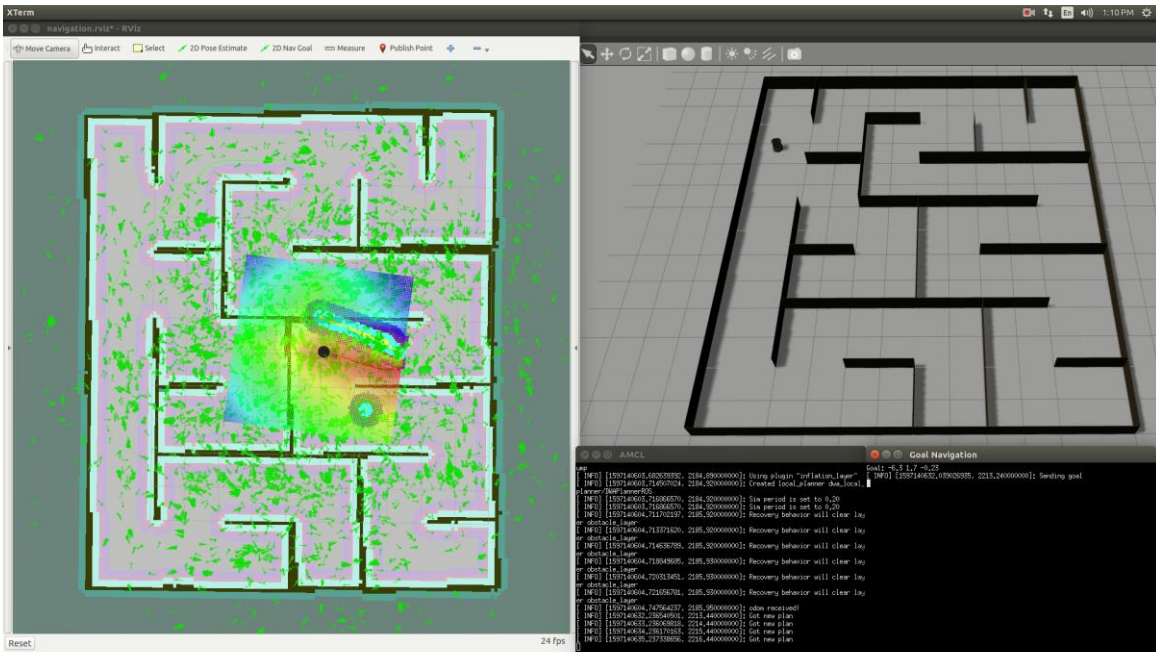


```
ump
[ INFO] [1597140603.615511859, 2184.820000000]: Using plugin "inflation_layer"
[ INFO] [1597140603.651305840, 2184.860000000]: Using plugin "obstacle_layer"
[ INFO] [1597140603.653747276, 2184.860000000]:     Subscribed to Topics: scan b
ump
[ INFO] [1597140603.682639392, 2184.890000000]: Using plugin "inflation_layer"
[ INFO] [1597140603.714507024, 2184.920000000]: Created local_planner dwa_local_
planner/DWAPlannerROS
[ INFO] [1597140603.716866570, 2184.920000000]: Sim period is set to 0.20
[ INFO] [1597140604.711702197, 2185.920000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.713371620, 2185.920000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.714636789, 2185.920000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.718849685, 2185.930000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.720313451, 2185.930000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.721656781, 2185.930000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.747564237, 2185.950000000]: odom received!
[ INFO] [1597140632.236540501, 2213.440000000]: Got new plan
```

```
Goal: -6.3 1.7 -0.23
[ INFO] [1597140632.039026935, 2213.240000000]: Sending goal
```

Figure 5.4 Localization Status at T

- At T+4, the robot has moved his first steps and the particles are starting to converge as shown in Figure 5.5.

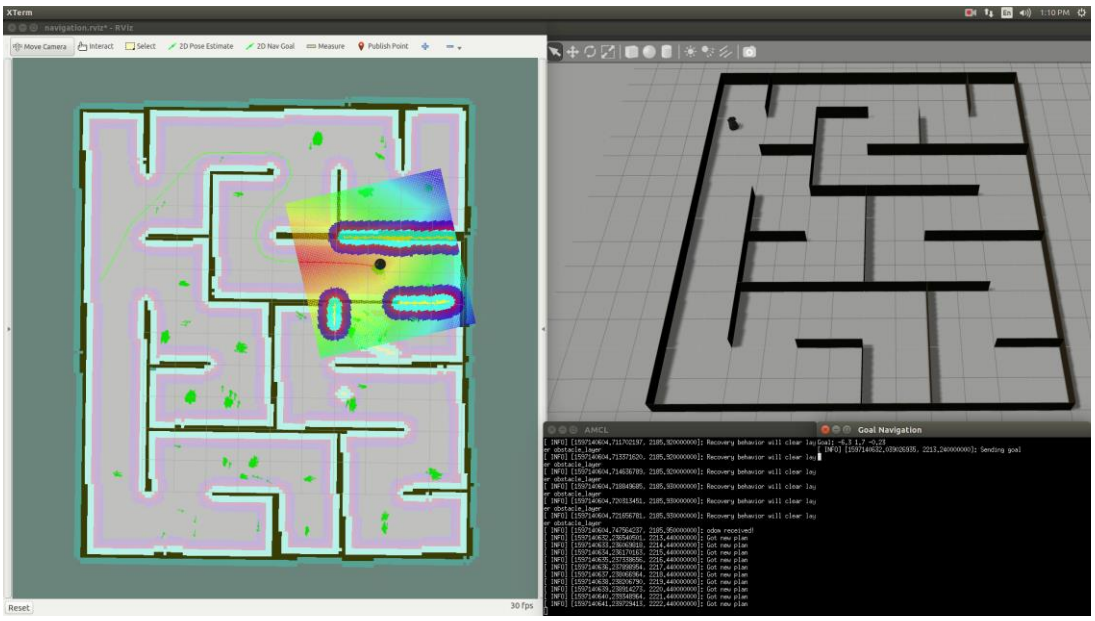


```
ump
[ INFO] [1597140603.682639392, 2184.890000000]: Using plugin "inflation_layer"
[ INFO] [1597140603.714507024, 2184.920000000]: Created local_planner dwa_local_
planner/DWAPlannerROS
[ INFO] [1597140603.716866570, 2184.920000000]: Sim period is set to 0.20
[ INFO] [1597140603.716866570, 2184.920000000]: Sim period is set to 0.20
[ INFO] [1597140604.711702197, 2185.920000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.713371620, 2185.920000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.714636789, 2185.920000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.718849685, 2185.930000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.720313451, 2185.930000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.721656781, 2185.930000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604.747564237, 2185.950000000]: odom received!
[ INFO] [1597140632.236540501, 2213.440000000]: Got new plan
[ INFO] [1597140633.236069818, 2214.440000000]: Got new plan
[ INFO] [1597140634.236170163, 2215.440000000]: Got new plan
[ INFO] [1597140635.237338656, 2216.440000000]: Got new plan
```

```
Goal: -6.3 1.7 -0.23
[ INFO] [1597140632.039026935, 2213.240000000]: Sending goal
```

Figure 5.5 Localization Status at T+4

- At T+10, as the robot continues to move, the particles are converging more and more in specific places as shown in Figure 5.6. The virtual robot changes position to try to loacalize itself in the most probable position, but notice that there are no particles in the correct position.

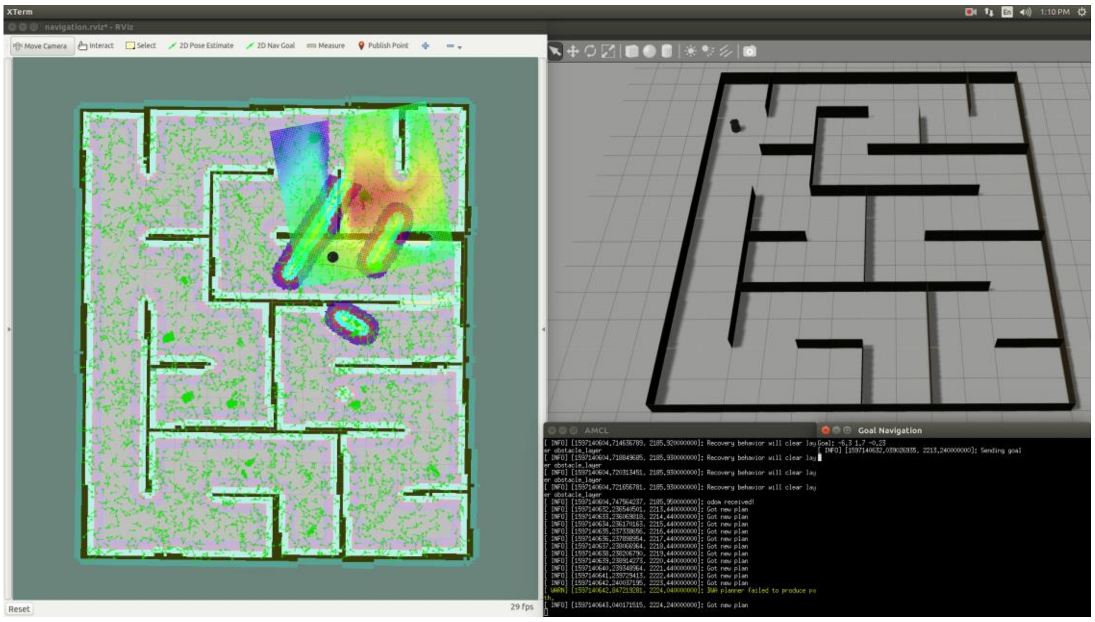


```
[ INFO] [1597140604,711702197, 2185,920000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604,713371620, 2185,920000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604,714636789, 2185,920000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604,718849685, 2185,930000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604,720313451, 2185,930000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604,721656781, 2185,930000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604,747564237, 2185,950000000]: odom received!
[ INFO] [1597140632,236540501, 2213,440000000]: Got new plan
[ INFO] [1597140633,236069818, 2214,440000000]: Got new plan
[ INFO] [1597140634,236170163, 2215,440000000]: Got new plan
[ INFO] [1597140635,237338656, 2216,440000000]: Got new plan
[ INFO] [1597140636,237898954, 2217,440000000]: Got new plan
[ INFO] [1597140637,238066964, 2218,440000000]: Got new plan
[ INFO] [1597140638,238206790, 2219,440000000]: Got new plan
[ INFO] [1597140639,238914273, 2220,440000000]: Got new plan
[ INFO] [1597140640,239348964, 2221,440000000]: Got new plan
[ INFO] [1597140641,239729413, 2222,440000000]: Got new plan
```

```
Goal: -6,3 1,7 -0,23
[ INFO] [1597140632,039026935, 2213,240000000]: Sending goal
```

Figure 5.6 Localization Status at T+10

- At T+11, as shown in Figure 5.7, a portion of the particles are respread around the map by the AMCL algorithm as an attempt to recover from a possible localization failure (which actually happened in this example) as explained in section 3.3.2.

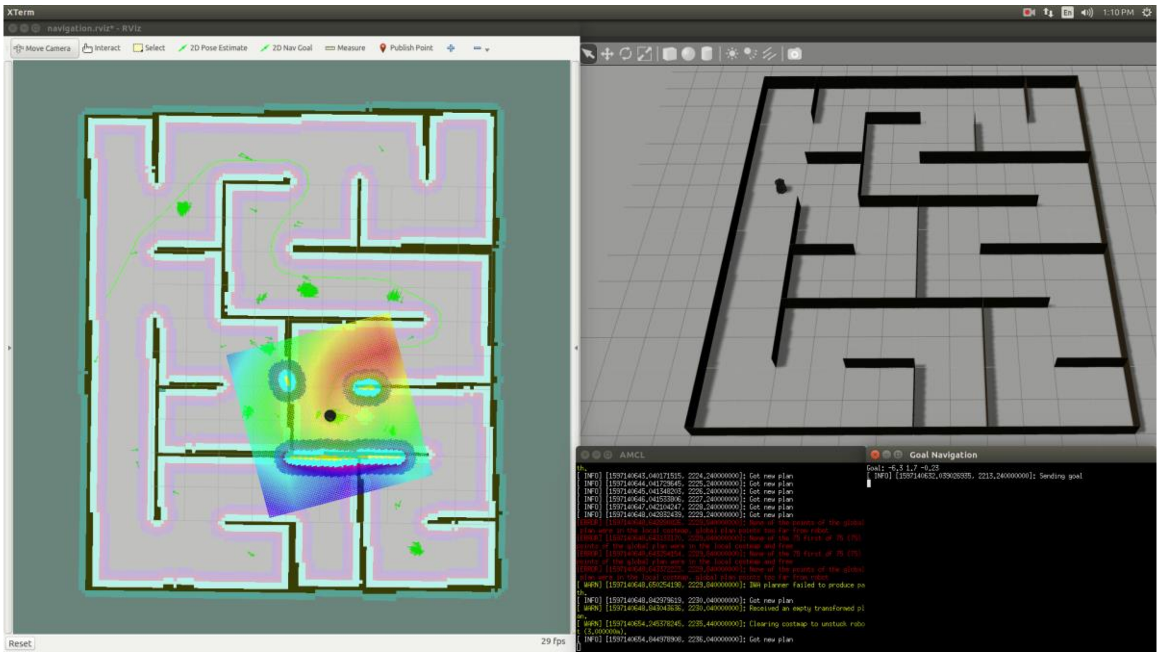


```
[ INFO] [1597140604,714636789, 2185,920000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604,718849685, 2185,930000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604,720313451, 2185,930000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604,721656781, 2185,930000000]: Recovery behavior will clear lay
er obstacle_layer
[ INFO] [1597140604,747564237, 2185,950000000]: odom received!
[ INFO] [1597140632,236540501, 2213,440000000]: Got new plan
[ INFO] [1597140633,236069818, 2214,440000000]: Got new plan
[ INFO] [1597140634,236170163, 2215,440000000]: Got new plan
[ INFO] [1597140635,237338656, 2216,440000000]: Got new plan
[ INFO] [1597140636,237898954, 2217,440000000]: Got new plan
[ INFO] [1597140637,238066964, 2218,440000000]: Got new plan
[ INFO] [1597140638,238206790, 2219,440000000]: Got new plan
[ INFO] [1597140639,238914273, 2220,440000000]: Got new plan
[ INFO] [1597140640,239348964, 2221,440000000]: Got new plan
[ INFO] [1597140641,239729413, 2222,440000000]: Got new plan
[ INFO] [1597140642,240037195, 2223,440000000]: Got new plan
[ WARN] [1597140642,847219281, 2224,040000000]: DWA planner failed to produce pa
th.
[ INFO] [1597140643,040171515, 2224,240000000]: Got new plan
```

```
Goal: -6,3 1,7 -0,23
[ INFO] [1597140632,039026935, 2213,240000000]: Sending goal
```

Figure 5.7 Localization Status at T+11

- At T+23, the particles converged again, but still not in the correct position. Look at Figure 5.8.

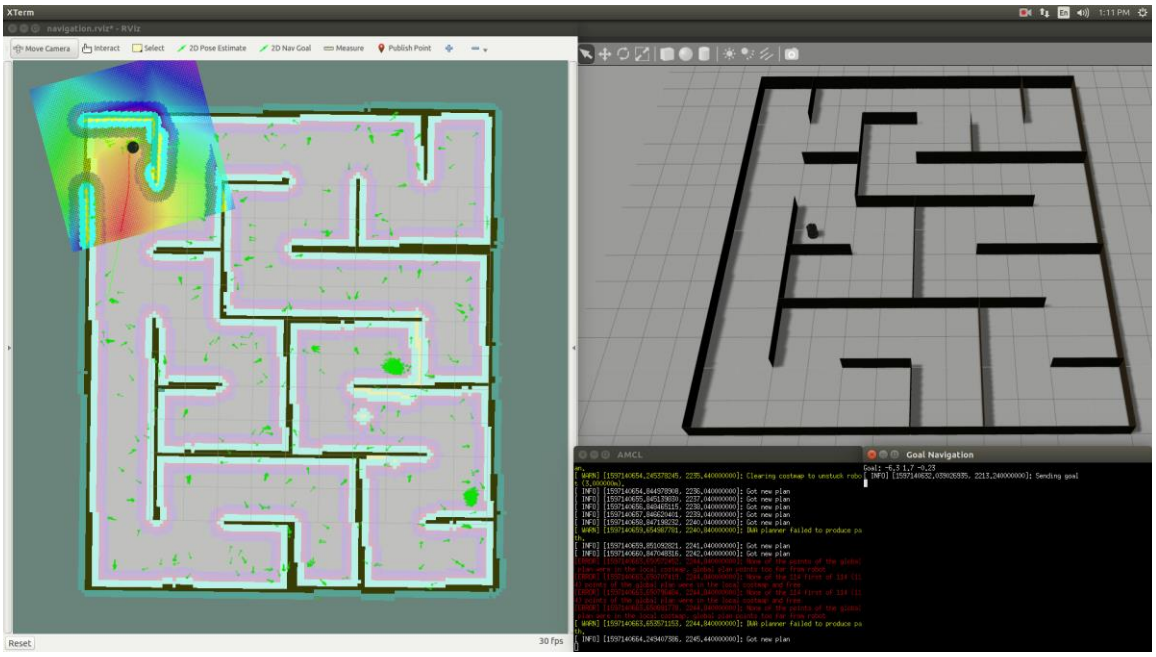


```
th.
[ INFO] [1597140643.040171515, 2224.240000000]: Got new plan
[ INFO] [1597140644.041729645, 2225.240000000]: Got new plan
[ INFO] [1597140645.041348203, 2226.240000000]: Got new plan
[ INFO] [1597140646.041533806, 2227.240000000]: Got new plan
[ INFO] [1597140647.042104247, 2228.240000000]: Got new plan
[ INFO] [1597140648.042832439, 2229.240000000]: Got new plan
[ERROR] [1597140648.[illegible], 2229.640000000]: None of the points of the global
 plan were in the local costmap, global plan points too far from robot
[ERROR] [1597140648.[illegible], 2229.640000000]: None of the 75 first of 75 (75)
points of the global plan were in the local costmap and free
[ERROR] [1597140648.[illegible], 2229.640000000]: None of the 75 first of 75 (75)
points of the global plan were in the local costmap and free
[ERROR] [1597140648.[illegible], 2229.640000000]: None of the points of the global
 plan were in the local costmap, global plan points too far from robot
[ WARN] [1597140648.650254198, 2229.640000000]: DWA planner failed to produce pa
th.
[ INFO] [1597140648.842979619, 2230.040000000]: Got new plan
[ WARN] [1597140648.843043636, 2230.040000000]: Received an empty transformed pl
an.
[ WARN] [1597140654.245378245, 2235.440000000]: Clearing costmap to unstuck robo
t (3.000000m).
[ INFO] [1597140654.844978908, 2236.040000000]: Got new plan
```

```
Goal: -6,3 1.7 -0.23
[ INFO] [1597140632.035026335, 2213.240000000]: Sending goal
```

Figure 5.8 Localization Status at T+23

- At T+33, another respreading action is attempted, the virtual robot keeps trying the positions where the particles converge, but still not correct as shown in Figure 5.9.

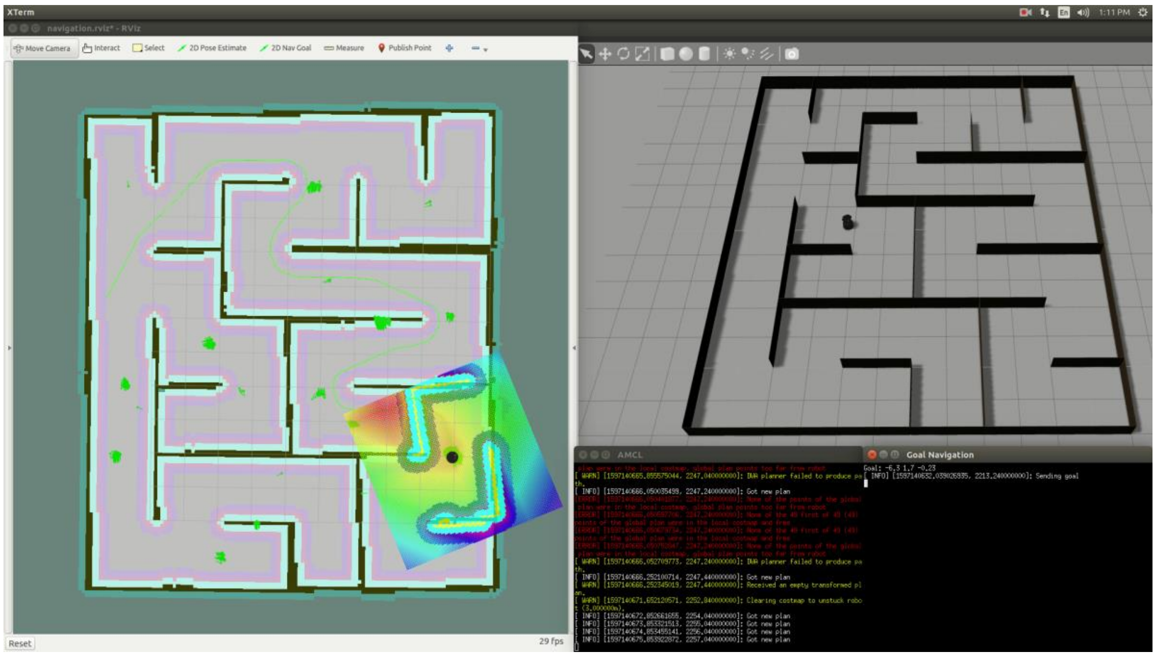


```
an.
[ WARN] [1597140654.245378245, 2235.440000000]: Clearing costmap to unstuck robo
t (3.000000m).
[ INFO] [1597140654.844978908, 2236.040000000]: Got new plan
[ INFO] [1597140655.845139830, 2237.040000000]: Got new plan
[ INFO] [1597140656.849465115, 2238.040000000]: Got new plan
[ INFO] [1597140657.846620401, 2239.040000000]: Got new plan
[ INFO] [1597140658.847198232, 2240.040000000]: Got new plan
[ WARN] [1597140659.654987781, 2240.840000000]: DWA planner failed to produce pa
th.
[ INFO] [1597140659.851092821, 2241.040000000]: Got new plan
[ INFO] [1597140660.847048316, 2242.040000000]: Got new plan
[ERROR] [1597140663.[illegible], 2244.840000000]: None of the points of the global
 plan were in the local costmap, global plan points too far from robot
[ERROR] [1597140663.[illegible], 2244.840000000]: None of the 114 first of 114 (11
4) points of the global plan were in the local costmap and free
[ERROR] [1597140663.[illegible], 2244.840000000]: None of the 114 first of 114 (11
4) points of the global plan were in the local costmap and free
[ERROR] [1597140663.[illegible], 2244.840000000]: None of the points of the global
 plan were in the local costmap, global plan points too far from robot
[ WARN] [1597140663.653571153, 2244.840000000]: DWA planner failed to produce pa
th.
[ INFO] [1597140664.249407386, 2245.440000000]: Got new plan
```

```
Goal: -6,3 1.7 -0.23
[ INFO] [1597140632.035026335, 2213.240000000]: Sending goal
```

Figure 5.9 Localization Status at T+33

- At T+44, some particles converged in the correct place for the first time as seen in Figure 5.10.

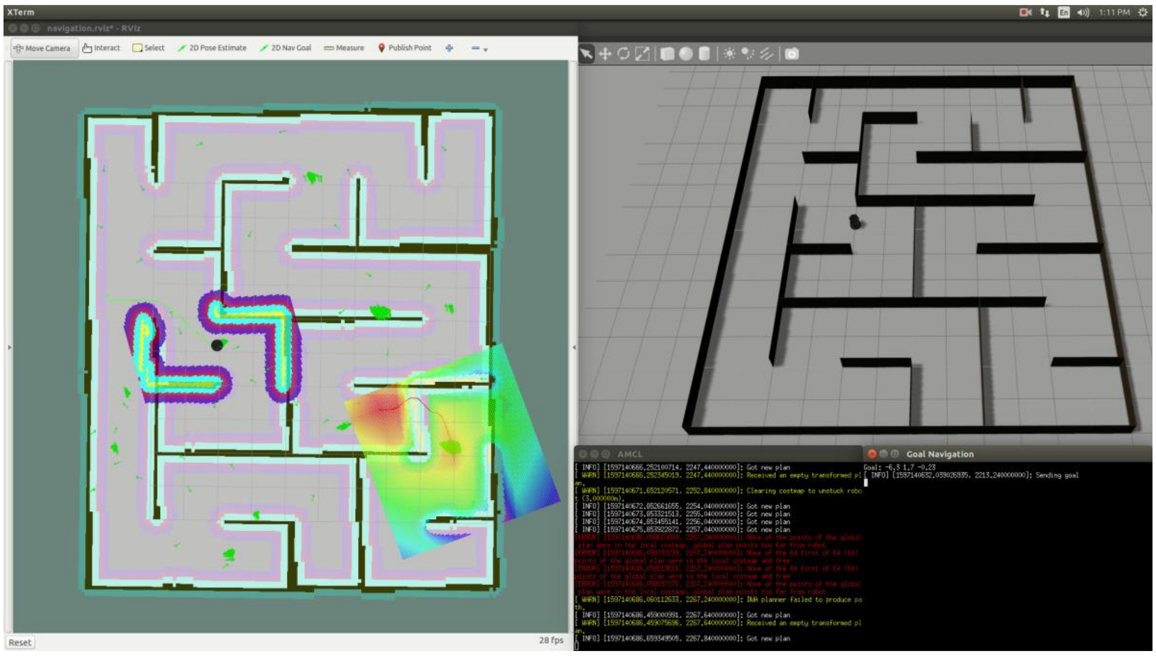


```
plan were in the local costmap, global plan points too far from robot
[ WARN] [1597140665.855575044, 2247.040000000]: DWA planner failed to produce pa
th.
[ INFO] [1597140666.050035499, 2247.240000000]: Got new plan
[ERROR] [illegible]: None of the points of the global
 plan were in the local costmap, global plan points too far from robot
[ERROR] [illegible]: None of the 48 first of 48 (48)
points of the global plan were in the local costmap and free
[ERROR] [illegible]: None of the 48 first of 48 (48)
points of the global plan were in the local costmap and free
[ERROR] [illegible]: None of the points of the global
 plan were in the local costmap, global plan points too far from robot
[ WARN] [1597140666.052709773, 2247.240000000]: DWA planner failed to produce pa
th.
[ INFO] [1597140666.252100714, 2247.440000000]: Got new plan
[ WARN] [1597140666.252345019, 2247.440000000]: Received an empty transformed pl
an.
[ WARN] [1597140671.652120571, 2252.840000000]: Clearing costmap to unstuck robo
t (3.000000m).
[ INFO] [1597140672.852661655, 2254.040000000]: Got new plan
[ INFO] [1597140673.853321513, 2255.040000000]: Got new plan
[ INFO] [1597140674.853455141, 2256.040000000]: Got new plan
[ INFO] [1597140675.853922872, 2257.040000000]: Got new plan
```

```
Goal: -6.3 1.7 -0.23
[ INFO] [1597140632.039026935, 2213.240000000]: Sending goal
```

Figure 5.10 Localization Status at T+44

- At T+55, the virtual robot is finally placed in the correct position. Look at Figure 5.11.

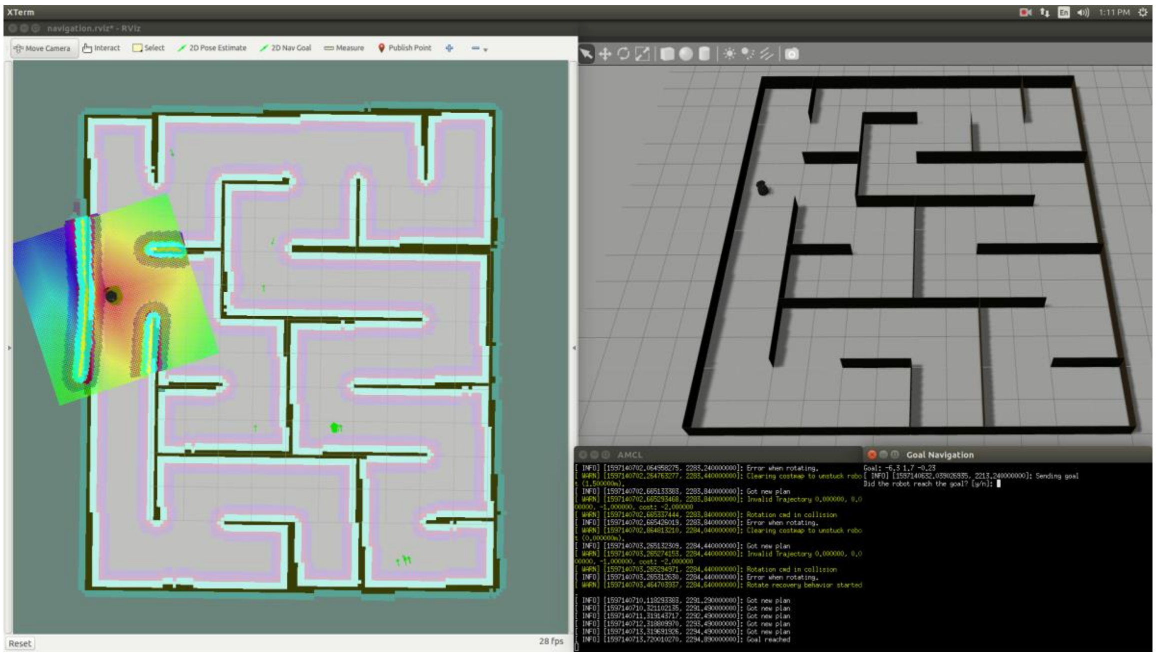


```
[ INFO] [1597140666.252100714, 2247.440000000]: Got new plan
[ WARN] [1597140666.252345019, 2247.440000000]: Received an empty transformed pl
an.
[ WARN] [1597140671.652120571, 2252.840000000]: Clearing costmap to unstuck robo
t (3.000000m).
[ INFO] [1597140672.852661655, 2254.040000000]: Got new plan
[ INFO] [1597140673.853321513, 2255.040000000]: Got new plan
[ INFO] [1597140674.853455141, 2256.040000000]: Got new plan
[ INFO] [1597140675.853922872, 2257.040000000]: Got new plan
[ERROR] [illegible]: None of the points of the global
 plan were in the local costmap, global plan points too far from robot
[ERROR] [illegible]: None of the 64 first of 64 (64)
points of the global plan were in the local costmap and free
[ERROR] [illegible]: None of the 64 first of 64 (64)
points of the global plan were in the local costmap and free
[ERROR] [illegible]: None of the points of the global
 plan were in the local costmap, global plan points too far from robot
[ WARN] [1597140686.060112633, 2267.240000000]: DWA planner failed to produce pa
th.
[ INFO] [1597140686.459000991, 2267.640000000]: Got new plan
[ WARN] [1597140686.459075696, 2267.640000000]: Received an empty transformed pl
an.
[ INFO] [1597140686.659349505, 2267.840000000]: Got new plan
```

```
Goal: -6.3 1.7 -0.23
[ INFO] [1597140632.039026935, 2213.240000000]: Sending goal
```

Figure 5.11 Localization Status at T+55

- At T+82, the robot navigates straight to the goal pose and stops for a successful localization and navigation attempt, as shown in Figure 5.12.



```
[ INFO] [1597140702.064958275, 2283.240000000]: Error when rotating.
[ WARN] [1597140702.264763277, 2283.440000000]: Clearing costmap to unstuck robo
t (1.500000m).
[ INFO] [1597140702.665133383, 2283.840000000]: Got new plan
[ WARN] [1597140702.665293468, 2283.840000000]: Invalid Trajectory 0.000000, 0.0
00000, -1.000000, cost: -2.000000
[ WARN] [1597140702.665337444, 2283.840000000]: Rotation cmd in collision
[ INFO] [1597140702.665426019, 2283.840000000]: Error when rotating.
[ WARN] [1597140702.864813210, 2284.040000000]: Clearing costmap to unstuck robo
t (0.000000m).
[ INFO] [1597140703.265132309, 2284.440000000]: Got new plan
[ WARN] [1597140703.265274153, 2284.440000000]: Invalid Trajectory 0.000000, 0.0
00000, -1.000000, cost: -2.000000
[ WARN] [1597140703.265294971, 2284.440000000]: Rotation cmd in collision
[ INFO] [1597140703.265312630, 2284.440000000]: Error when rotating.
[ WARN] [1597140703.464703937, 2284.640000000]: Rotate recovery behavior started
[
[ INFO] [1597140710.118293383, 2291.290000000]: Got new plan
[ INFO] [1597140710.321102135, 2291.490000000]: Got new plan
[ INFO] [1597140711.319143717, 2292.490000000]: Got new plan
[ INFO] [1597140712.318809970, 2293.490000000]: Got new plan
[ INFO] [1597140713.319691926, 2294.490000000]: Got new plan
[ INFO] [1597140713.720010270, 2294.890000000]: Goal reached
```

```
Goal: -6.3 1.7 -0.23
[ INFO] [1597140632.039026935, 2213.240000000]: Sending goal
Did the robot reach the goal? [y/n]:
```

Figure 5.12 Localization Status at T+82

The above results verify that the localization and the optimal path subsystems work as intended.

5.4 Object Detection

Random images of the object to be detected were taken from various positions, lights and shooting angles. The prediction confidence of object recognition was as follows in the Figure 5.13 - Figure 5.20.

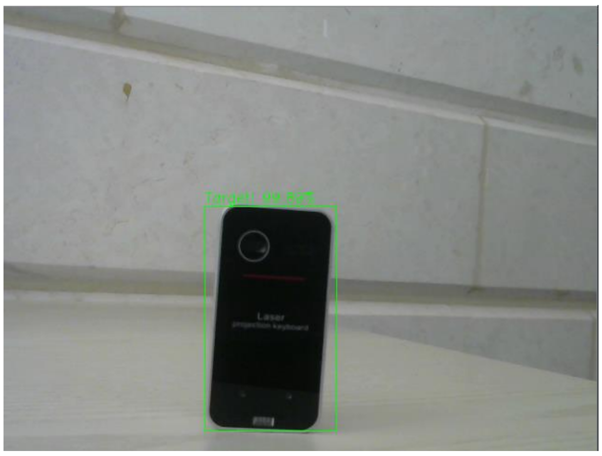


Figure 5.13 99.69% Prediction Confidence

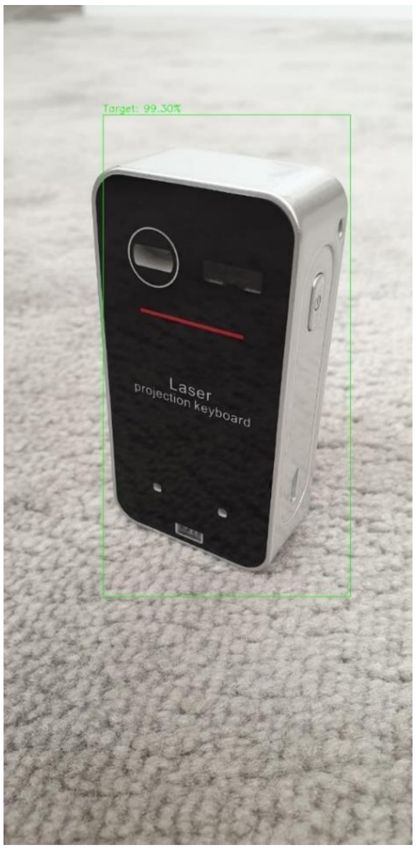


Figure 5.14 99.30% Prediction Confidence

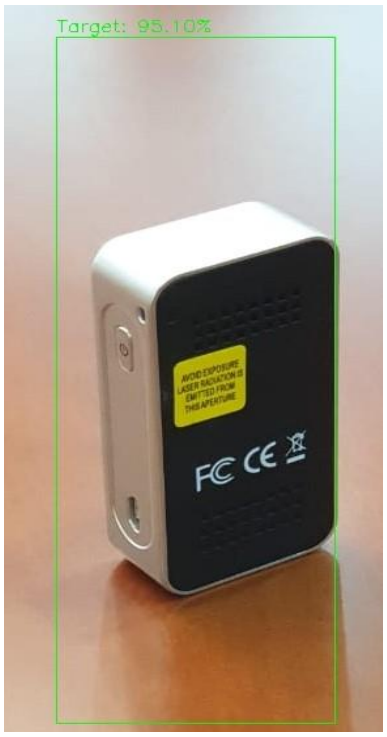


Figure 5.15 95.10% Prediction Confidence

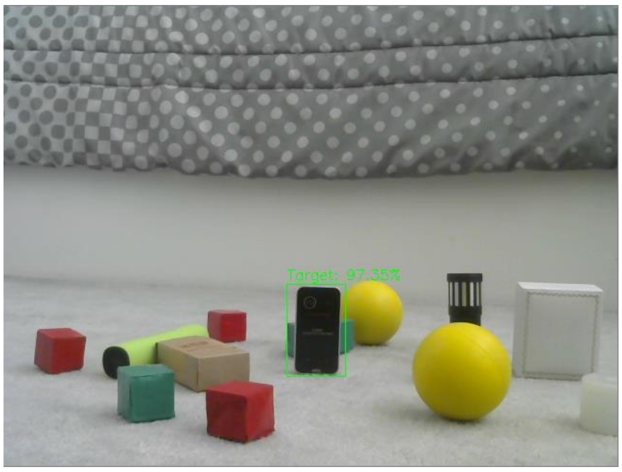


Figure 5.16 97.35% Prediction Confidence

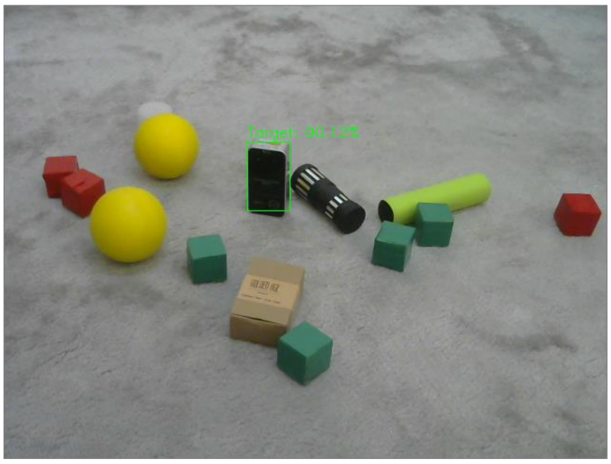


Figure 5.17 90.12% Prediction Confidence

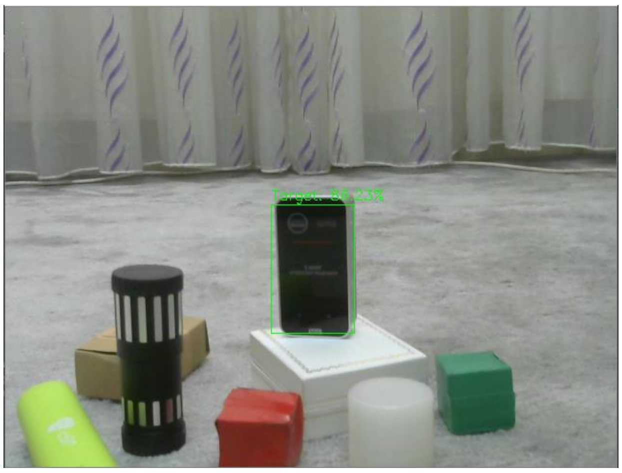


Figure 5.18 86.23% Prediction Confidence

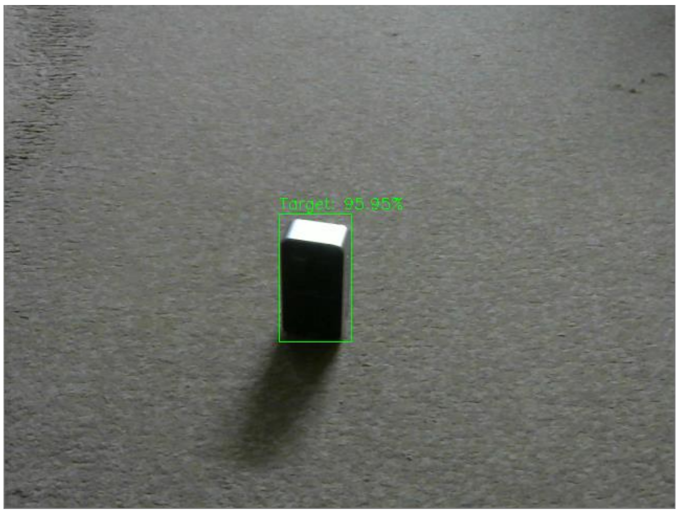


Figure 5.19 95.95% Prediction Confidence

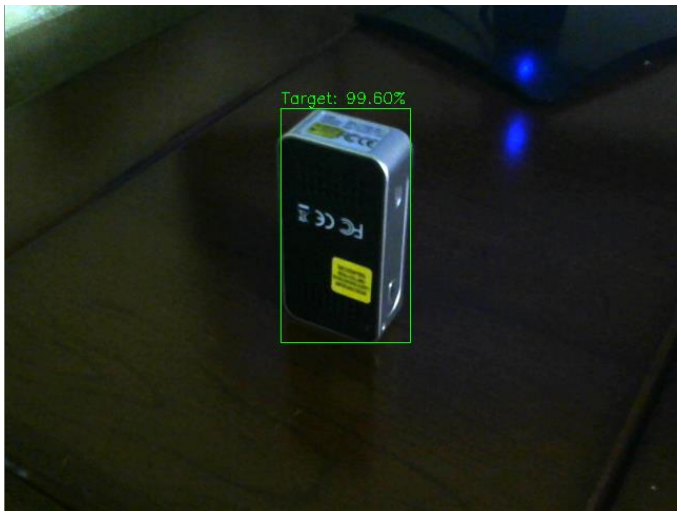


Figure 5.20 99.60% Prediction Confidence

The above results verify that the object detection subsystem works as intended.

5.5 Integration of Subsystems

After verifying that each subsystem achieves its goal individually, this section shows how those subsystems integrate with each other to achieve the goals of the project.

In the mapping phase, the Rapidly-Exploring Random Tree algorithm searches for the nearest undiscovered area in the environment, and creates a goal pose for the robot. Dijkstra’s algorithm then finds the optimal path for that goal and creates a global plan. The robot then moves towards the goal through that plan. While the robot moves, the LIDAR sensor scans the surroundings and the encoder measures the displacement continuously and they both send the data to the Occupancy Grid Mapping algorithm which creates a map for the environment. This cycle is repeated until the whole environment is discovered. The map then can be saved to be used in the localization phase.

In the localization phase, the robot is usually placed in an initial pose arbitrary. The user must provide the map of the environment (which was created earlier) and must specify an approximate position of the target object before the localization is started. Initially, the robot assumes it is in the center of the map, the particles of the particle filter are spread across the map, and an initial global plan is created by Dijkstra’s algorithm towards the target object. The robot starts moving, the LIDAR sensor starts scanning the environment and the encoder starts measuring the displacement, and the data is sent to the Adaptive Monte Carlo Localization algorithm (AMCL). As the robot moves, the particles converge in some places by the AMCL depending on the gathered data. The robot tries to localize itself in the places where the particles are converged, and each time, Dijkstra’s algorithm creates a new global plan. This cycle is repeated until the robot localizes itself in the correct place and navigates towards the target object.

When the robot gets close to the target object, the object detection phase starts. The camera starts capturing frames of the place where the object is approximately at. These frames are analyzed and compared to a custom model for the object which was built previously using the You Only Look Once algorithm (YOLO). If the prediction confidence of the target object at any frame is above 40%, then the object is considered to be detected, and the robot is then ready for further interactions with the object.

The validation of the system can be achieved after experimental results of the integrated system is done.

Chapter 6 **Conclusion and Future Work**

6.1 Conclusion

In this graduation project, we made an intelligent mobile robot to map mazes and then localize itself inside them. The robot also can detect and localize a certain object within its environment. The Mapping is done using the Occupancy Grid Mapping algorithm and is automated using the Rapidly-Exploring Random Tree algorithm, for localization, the robot uses the Adaptive Monte Carlo Localization algorithm (AMCL), Dijkstra's algorithm is used for planning the optimal path, and You Only Look Once (YOLO) is the algorithm used to create a model to recognize and detect the object. Gazebo, a simulation program, is used to navigate a Turtlebot robot inside virtual maze environments. The connections between the virtual robot and the algorithms mentioned earlier was done through Robot Operating System (ROS).

One hundred localization experements were done across 10 mazes. We were able to calculate the success rate and the average time of the successful attempts of localization and navigation using the data collected.

6.2 Future Work

Some suggestions for future work include modifying the localization algorithm to gather more data before judging when the particles are hardly converged in more than one position to reduce localization failure, mounting a robotic arm on the robot to interact with objects, automatic docking for recharging when the battery is low, and 3-D scanning of the environments for more detailed maps.

Appendix A Kinematics and Dynamics of the Mobile Robot

A.1 Introduction

In this chapter, the kinematics and dynamics of a differential drive mobile robot are derived, and the differences between the two models and limitations of the kinematic model are explained.

A.2 Non-Holonomic motion

Wheels are the most common mechanism to achieve locomotion in mobile robots. Any wheeled vehicle is subject to kinematic constraints that reduce its local mobility, for instance, a car can reach any final configuration in its plane, but it cannot move sideways. Depending on the goal configuration, it requires to perform a series of maneuvers (such as parallel parking) to reach the desired state. [35]

So, the holonomic and non-holonomic systems have to be defined. Consider a mechanical system whose configuration $q \epsilon C$ is described by a vector of generalized coordinates, where C is the configuration space of the proposed system and coincides with R^n. For considered system, a constraint is called Kinematic when it only involves generalized coordinates (q) and velocities ($\dot{q}$).[35]

A nonholonomic constraint is called a Pfaffian constraint if it is linear in $\dot{\boldsymbol{q}}$, that is, if it can be expressed in the form:

$$\mu_i^T (q)\dot{q} = 0 \tag{A.1}$$

$$i = 1, \dots , j$$

Where μ_i are linearly independent row vectors and $\boldsymbol{q} = \left[q_1, q_2, \ldots, q_n\right]^T$.

In compact matrix form the above j Pfaffian constraints can be written as:

$$M^T (q)\dot{q} = 0 \tag{A.2}$$

$$M(q) = \begin{bmatrix} \mu_1(q) \\ \mu_2(q) \\ \cdot \\ \mu_j(q) \end{bmatrix} \tag{A.3}$$

The nonholonomic constraint encountered in mobile robotics is the motion constraint of a disk that rolls on a plane without slipping (Figure A.1). The no-slipping condition does not allow the generalized velocities $\dot{x}$, $\dot{y}$ and $\dot{\phi}$ to take arbitrary values.

Let r be the disk radius , θ is wheel angle of rotation which is shown in Figure A.1. Due to the no-slipping condition the generalized coordinates are constrained by the following equations:

$$\dot{x} = r\dot{\theta}\cos\phi \tag{A.4}$$

$$\dot{y} = r\dot{\theta}\sin\phi \tag{A.5}$$

Which are not integrable. These constraints express the condition that the velocity vector of the disk center lies in the midplane of the disk. Eliminating the velocity in Eq. (A.4, A.5) gives:

$$v = r\dot{\theta} = \frac{\dot{x}}{\cos\phi} = \frac{\dot{y}}{\sin\phi} \tag{A.6}$$

Or

$$\dot{x}\sin\phi - \dot{y}\cos\phi = 0 \tag{A.7}$$

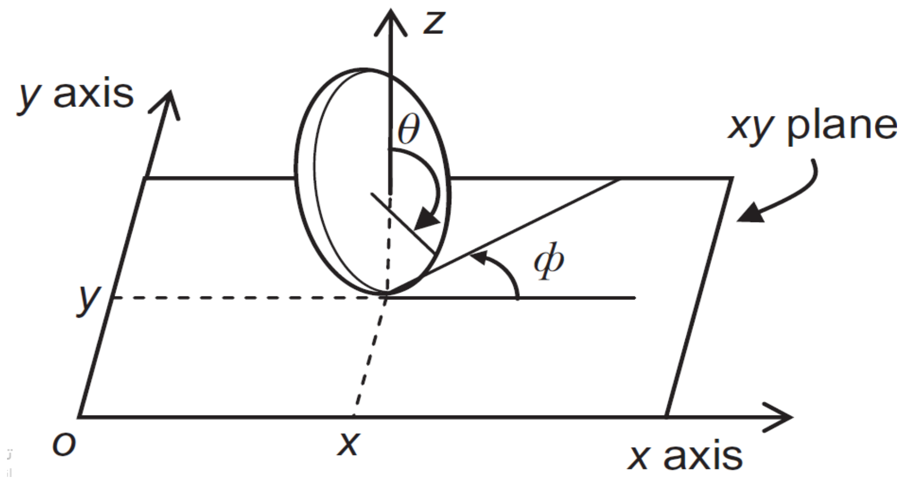


Figure A.1 The generalized coordinates x, y, and φ.

A.3 Kinematics model

Differential drive mobile robot (DDMR) is considered as unicycle vehicle that has a single orientable wheel, its configuration is completely described by q.

Kinematic modeling is the study of the motion of mechanical systems without considering the forces that affect the motion, so the goal is to represent the robot velocities as a function of the driving wheels velocities along with the geometric parameters of the robot.[35]

Indoor and other mobile robots use the differential drive locomotion type. The Pioneer DDMR is an example of differential drive WMR. The geometry and kinematic parameters of this robot are shown in Figure A.2. The pose (position/orientation) vector of the DDMR and its speed are respectively:[35]

$$\mathbf{p} = \begin{bmatrix} x_Q \\ y_Q \\ \phi \end{bmatrix} \tag{A.8}$$

$$\dot{\mathbf{p}} = \begin{bmatrix} \dot{x}_Q \\ \dot{y}_Q \\ \dot{\phi} \end{bmatrix} \tag{A.9}$$

The angular positions and speeds of the left and right wheels are $\{\theta_l, \dot{\theta}_l\}$, $\{\theta_r, \dot{\theta}_r\}$ respectively:

$$\mathbf{q} = \begin{bmatrix} \theta_r \\ \theta_l \end{bmatrix} \tag{A.10}$$

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{\theta}_r \\ \dot{\theta}_l \end{bmatrix} \tag{A.11}$$

The following assumptions are made:

- Wheels are rolling without slippage.
- The guidance (steering) axis is perpendicular to the plane O_{xy}.
- The point Q which is shown in Figure A.2 coincides with the center of gravity G, that is $\|\vec{GQ}\| = 0$.

Let v_r and v_l which are shown in Figure A.2 be the linear velocity of the right and left wheel respectively, and v_Q the velocity of the wheel midpoint Q of the DDMR. Then, from Figure A.2 we get:

$$v_r = v_Q + a\dot{\phi} \tag{A.12}$$

$$v_l = v_Q - a\dot{\phi} \tag{A.13}$$

Adding and subtracting v_r and v_l we get:

$$v_Q = \frac{1}{2}(v_r + v_l) \tag{A.14}$$

$$2a\dot{\phi} = v_r - v_l \tag{A.15}$$

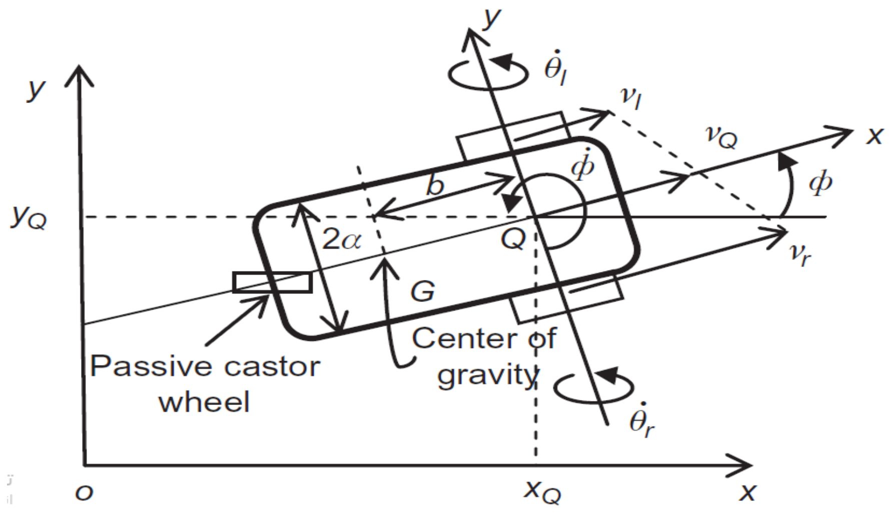


Figure A.2 Geometry of differential drive WMR [35]

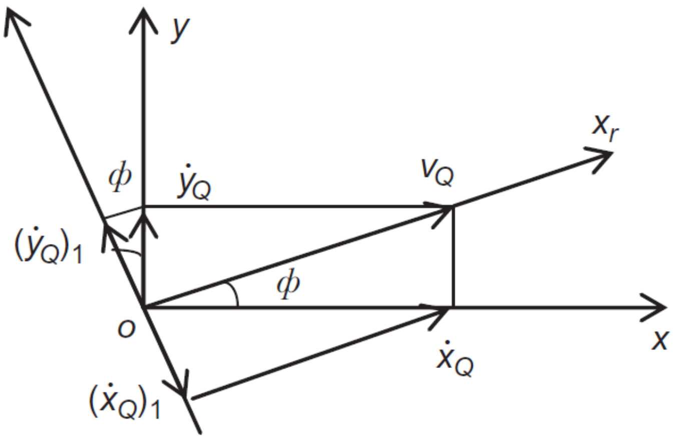


Figure A.3 Diagram illustrating the nonholonomic constraint [35]

Where, due to the nonslip assumption, we have:

$$v_r = r\dot{\theta}_r,,,v_l = r\dot{\theta}_l \tag{A.16}$$

As in Figure A.3, x_Q and y_Q are given by:

$$\dot{x}_Q = v_Q \cos\phi \tag{A.17}$$
$$\dot{y}_Q = v_Q \cos\phi \tag{A.18}$$

And so the kinematic model of this DDMR is described by the following relations:

$$\dot{x}_Q = \frac{r}{2}\left(\dot{\theta}_r \cos\phi + \dot{\theta}_l \cos\phi\right) \tag{A.19}$$
$$\dot{y}_Q = \frac{r}{2}\left(\dot{\theta}_r \sin\phi + \dot{\theta}_l \sin\phi\right) \tag{A.20}$$

$$\dot{\phi} = \frac{r}{2a}\left(\dot{\theta}_r - \dot{\theta}_l\right) \tag{A.21}$$

From the kinematic model ((4.19),(4.20),(4.21)) can be written in the driftless affine form:

$$\dot{\mathbf{p}} = \begin{bmatrix} (r/2)\cos\phi \\ (r/2)\sin\phi \\ r/2a \end{bmatrix} \dot{\theta}_r + \begin{bmatrix} (r/2)\cos\phi \\ (r/2)\sin\phi \\ -r/2a \end{bmatrix} \dot{\theta}_l \tag{A.22}$$

Or

$$\dot{\mathbf{p}} = \mathbf{J}\dot{\mathbf{q}} \tag{A.23}$$

And J is the DDMR's Jacobian:

$$\mathbf{J} = \begin{bmatrix} (r/2)\cos\phi & (r/2)\cos\phi \\ (r/2)\sin\phi & (r/2)\sin\phi \\ r/2a & -r/2a \end{bmatrix} \tag{A.24}$$

Here, the two 3-dimensional vector fields are:

$$\mathbf{g_1} = \begin{bmatrix} (r/2)\cos\phi \\ (r/2)\sin\phi \\ r/2a \end{bmatrix} \tag{A.25}$$

$$\mathbf{g_2} = \begin{bmatrix} (r/2)\cos\phi \\ (r/2)\sin\phi \\ -r/2a \end{bmatrix} \tag{A.26}$$

The field g1 allows the rotation of the right wheel, and g2 allows the rotation of the left wheel.

Now, in view of the assumed constraints (pure rolling with no lateral slip) the linear and translational velocity of the robot frame (centered at Q) can be expressed as:

$$\begin{bmatrix} \dot{x}_Q \\ \dot{y}_Q \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \cos\phi & 0 \\ \sin\phi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_Q \\ w \end{bmatrix} \tag{A.27}$$

Where v_Q and ω are the linear and angular velocities of the DDMR respectively. The linear velocity of the DDMR in the Robot Frame is therefore the average of the linear velocities of the two wheels:

$$v_Q = \frac{r\left(\dot{\theta}_r + \dot{\theta}_l\right)}{2} \tag{A.28}$$

And the angular velocity of the DDMR about the normal (z-axis) is:

$$w = \frac{r\left(\dot{\theta}_r - \dot{\theta}_l\right)}{2a} \tag{A.29}$$

Equations (A.28 and A.29) can be inverted to get the inverse velocity equations as:

$$\begin{bmatrix} \dot{\theta}_r \\ \dot{\theta}_l \end{bmatrix} = \frac{1}{r}\begin{bmatrix} 1 & a \\ 1 & -a \end{bmatrix}\begin{bmatrix} v_Q \\ w \end{bmatrix} \tag{A.30}$$

A.4 Dynamic model

The robot kinematics do not represent the actual inputs of the robot motors (i.e. forces and/or torques). In other words, we consider that the motors give enough torques to drive the robot when dealing just with a kinematic model. The dynamic model of the DDMR is essential for simulation, analysis of robot motion, and for the design of motion control algorithm.

A non-holonomic DDMR with n generalized coordinates (q_1, q_2, … , q_n) and subject to m constraints can be described by the following equation of motion:

$$\mathbf{B}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{F_v}\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} - \mathbf{M}^{\mathbf{T}}(\mathbf{q})\mathbf{h} \tag{A.31}$$

$$\boldsymbol{\tau} = \mathbf{T}(\mathbf{q})\mathbf{v} \tag{A.32}$$

Where,

B(q) an n × n symmetric positive definite inertia matrix,

C(q, $\dot{\mathbf{q}}$) is the centripetal and Coriolis matrix,

g(q) is the gravitational vector,

T(q) is the input matrix,

v is the input vector,

M$^{\mathbf{T}}$(q) is the matrix associated with the kinematic constraints,

h is the Lagrange multipliers vector,

$\mathbf{F_v}$ denotes the (n×n) diagonal matrix of viscous friction coefficient,

and $\boldsymbol{\tau}$ is the actuation torques.

There are two methods to derive the dynamic model for the DDMR. Newton-Euler method that describes the system in terms of all the forces and momentum acting on the system based of direct interpretations of Newton's Second Law of Motion. And the other one Lagrange method that derive the equations of motion by considering the kinetic and potential energies of the given system. Lagrange method is chosen to derive the system model.

A.4.1 **Lagrange dynamic approach**

The Lagrangian L of the mechanical system as the difference between its kinetic and potential energy as follows:

$$L = T - V \tag{A.33}$$

Where, L is the Lagrange function, T, and V are kinetic and potential energies respectively.

The Lagrange equation can be written in the following form:

$$\frac{d}{dt}\left(\frac{\delta L}{\delta \dot{q}_i}\right) + \frac{\delta L}{\delta q_i} = F - M^T(q)h \tag{A.34}$$

Where,

q_i is the generalized coordinates,

F is the generalized force vector,

M is the constraints matrix,

and h is the vector of Lagrange multipliers associated with the constraints.

The first step in deriving the dynamic model using the Lagrange approach is to find the kinetic and potential energies, since the differential drive mobile robot is moving in the X-Y plane, the potential energy is considered to be zero.

We chose to make the dynamic model of DDMR along with the two motors:

$$\mathbf{q} = \begin{bmatrix} x \\ y \\ \phi \end{bmatrix} \tag{A.35}$$

$\mathbf{q}$ is the vector of generalized variables (linear, angular)

Let be the mass of DDMR, m_{mr}, m_{ml} be the masses of the motors I_{mr}, I_{ml} be the moments of inertia with respect to the axes of the two motors and Let $m_{mr} = m_{ml}$ and $I_{mr} = I_{ml}$.

With the chosen coordinate frames which are shown in Figure A.2 , computation of the Jacobians in kinematic model:

For the motors:

Linear and translational velocity:

$$J_{pmr} = \begin{bmatrix} \frac{r}{2}\cos\phi & \frac{r}{2}\cos\phi \\ \frac{r}{2}\sin\phi & \frac{r}{2}\sin\phi \\ 0 & 0 \end{bmatrix} \tag{A.36}$$

Angular velocity:

$$J_{omr} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{rK_R}{2a} & \frac{-rK_R}{2a} \end{bmatrix} = J_{oml} \tag{A.37}$$

Where K_R is the gear ratio of Motor. K_R is the same for right and left motor.

For the DDMR:

Linear and translational velocity:

$$J_p = \begin{bmatrix} \frac{r}{2}\cos\phi & \frac{r}{2}\cos\phi \\ \frac{r}{2}\sin\phi & \frac{r}{2}\sin\phi \\ 0 & 0 \end{bmatrix} \tag{A.38}$$

Angular velocity:

$$J_o = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{r}{2a} & \frac{-r}{2a} \end{bmatrix} \tag{A.39}$$

We can get the inertia matrix using the equation:

$$B(q) = \left(mJ_P^T J_P + IJ_o^T J_O \right) + \sum_{i=1}^{n} (m_{mi} J_{Pmi}^T J_{pmi} \, \mathrm{I}_{mi} \, J_{omi}^T J_{omi}) \tag{A.40}$$

Which leads to the following:

$$B(q) = \begin{bmatrix} \frac{r^2}{2}\left(m_m + \frac{m}{2} + \frac{I_m}{a^2} K_R^2 + \frac{I}{2a^2} \right) & \frac{r^2}{2}\left(m_m + \frac{m}{2} - \frac{I_m}{a^2} K_R^2 - \frac{I}{2a^2} \right) \\ \frac{r^2}{2}\left(m_m + \frac{m}{2} - \frac{I_m}{a^2} K_R^2 - \frac{I}{2a^2} \right) & \frac{r^2}{2}\left(m_m + \frac{m}{2} + \frac{I_m}{a^2} K_R^2 + \frac{I}{2a^2} \right) \end{bmatrix} \tag{A.41}$$

This implies **C(q)**=0, i.e. there are no contributions of centrifugal and Coriolis forces. **g(q)**=0, since the differential drive mobile robot is moving in the X-Y plane.

The viscous friction coefficients for each motor D_{mr}, respectively, so the matrix of viscous friction coefficient is:

$$F_V = \begin{bmatrix} D_{mr} & o \\ o & D_{ml} \end{bmatrix} \tag{A.42}$$

For the equation of motion:

$$\mathbf{B}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{F_v}\dot{\mathbf{q}} = \boldsymbol{\tau} \tag{A.43}$$

Solving for $\boldsymbol{\tau}$ we get:

$$\tau_r = \left(\frac{m}{2} + m_m + \frac{I_m}{a^2} K_R^2 + \frac{I}{2a^2} \right) \frac{r^2 \times \ddot{\theta}_r}{2} + \left(\frac{m}{2} + m_m - \frac{I_m}{a^2} K_R^2 - \frac{I}{2a^2} \right) \frac{r^2 \times \ddot{\theta}_l}{2} + D_{mr} \times \dot{\theta}_r \tag{A.44}$$

$$\tau_l = \left(\frac{m}{2} + m_m - \frac{I_m}{a^2} K_R^2 - \frac{I}{2a^2} \right) \frac{r^2 \times \ddot{\theta}_r}{2} + \left(\frac{m}{2} + m_m + \frac{I_m}{a^2} K_R^2 + \frac{I}{2a^2} \right) \frac{r^2 \times \ddot{\theta}_l}{2} + D_{ml} \times \dot{\theta}_l \tag{A.45}$$

From kinematic model:

$$\begin{bmatrix} \dot{\theta}_r \\ \dot{\theta}_l \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & a \\ 1 & -a \end{bmatrix} \begin{bmatrix} v_Q \\ w \end{bmatrix} \tag{A.46}$$

Taking the derivative for both sides, we get:

$$\begin{bmatrix} \ddot{\theta}_r \\ \ddot{\theta}_l \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & a \\ 1 & -a \end{bmatrix} \begin{bmatrix} \dot{v}_Q \\ \dot{w} \end{bmatrix} \tag{A.47}$$

Which implies:

$$\ddot{\theta}_r = \frac{1}{r}\left(\dot{v}_Q + a\dot{w} \right) \tag{A.48}$$

$$\ddot{\theta}_l = \frac{1}{r}\left(\dot{v}_Q - a\dot{w} \right) \tag{A.49}$$

And:

$$\dot{v} = \frac{(\tau_r + \tau_l) - (2D_m / r)v}{(m + 2m_m)r} \tag{A.50}$$

$$\dot{w} = \frac{(\tau_r - \tau_l) - (2aD_m / r)w}{(I + 2I_m K_R^2) r / a} \tag{A.51}$$

A.4.2 **Actuator dynamics**

The DC motors are generally used to drive the wheels of the robot. Since the wheels motor are symmetrical, it is considered to derive the model for one of them. There are two classes of DC motors control which are a filed-current controlled and armature-voltage controlled.

An armature-controlled dc motor shown in Figure A.4Figure A.4 which is the case for our robot system, the armature voltage V_a is used as the control input while keeping the conditions in the field circuit constant. For a permanent-magnet dc motor, we have the following equations for the armature circuit:

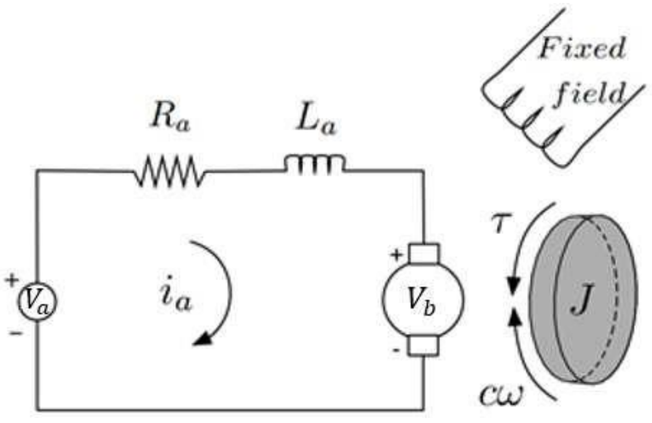


Figure A.4 Circuit equivalent of a DC motor

Based on circuit model provided in Figure (A.4), and considering the back-EMF voltage (v_b), induced by the rotation of armature winding, the voltage relation on the armature will be:

$$v_a = v_b + v_L + v_R \tag{A.52}$$

Where, v_R , v_L are voltages across R_a , L_a respectively.

Back-EMF has a linear relation to the motor angular speed (w_m) through back EMF constant as follow:

$$v_b = k_b w_m \tag{A.53}$$

By substitute equation (A.53) in equation (A.52) and taking the Laplace transform, the following equation is achieved:

$$v_a = R_a i_a + L_a \frac{di_a}{dt} + K_b w_m \tag{A.54}$$

We assume L_a equals 0 H.

In an armature-voltage controlled structure, the motor torque is linearly dependent on the armature current by:

$$\tau_m = k_m i_a \tag{A.55}$$

Where,

τ_m is the torque of the motor and km is the motor torque constant.

Then:

$$\tau_m^* = \frac{k_m v_a}{R_a} - \frac{k_b k_m w_m}{R_a} \tag{A.56}$$

But $w_m = \dot{\theta}^*$, so:

$$\tau_r^* = \frac{k_m v_{ar}}{R_a} - \frac{k_b k_m \dot{\theta}_r^*}{R_a} \tag{A.57}$$

$$\tau_l^* = \frac{k_m v_{al}}{R_a} - \frac{k_b k_m \dot{\theta}_l^*}{R_a} \tag{A.58}$$

But these torques of motors without look at the gear ratio, because of that:

$$\tau_l = K_R \tau_l^* \tag{A.59}$$

$$\tau_r = K_R \tau_r^* \tag{A.60}$$

And the same thing for the rotation speed of motors:

$$\dot{\theta}_r = \dot{\theta}_r^* \div K_r \tag{A.61}$$

$$\dot{\theta}_l = \dot{\theta}_l^* \div K_r \tag{A.62}$$

Then:

$$\ddot{\phi} = \frac{(\tau_r - \tau_l) - (2aD_m / r)\dot{\phi}}{(I + 2I_m K_R^2) r / a} \tag{A.63}$$

Will be:

$$\ddot{\phi} = \frac{-(2k_b aC_3 + arC_4)}{rC_2}\dot{\phi} + \frac{C_3 K_R}{C_2} v_{ar} - \frac{C_3 K_R}{C_2} v_{al} \tag{A.64}$$

And:

$$\dot{v} = \frac{(\tau_r + \tau_l) - (2D_m / r)v}{(m + 2m_m)r} \tag{A.65}$$

Will be:

$$\ddot{d} = \frac{C_3 K_R}{C_1} v_{ar} + \frac{C_3 K_R}{C_1} v_{al} - \frac{(2k_b C_3 + rC_4)}{rC_1} \dot{d} \tag{A.66}$$

Where $v = \dot{d}$ and $w = \dot{\phi}$.

These is the value of constants C1, C2, C3 and C4:

$$C_1 = (m + 2m_m)r \tag{A.67}$$

$$C_2 = \left(I + 2I_m K_R^2\right) r / a \tag{A.68}$$

$$C_3 = \frac{k_m}{R_a} \tag{A.69}$$

$$C_4 = 2D_m / r \tag{A.70}$$

Table A.1 shows the system parameters.

Table A.1 System parameters

Parameter	Description	Nominal Value
R_a	Armature Inductance	1.5506 ohm
r	Wheel Radius	0.038 m
m	DDWR mass	10 Kg
m_m	Motor mass	0.2 Kg
I	DDWR inertia	2 Kg.m2
I_m	Motor inertia	0.002 Kg.m2
a	Half of distance between wheels	0.112 m
k_b	Back emf Constant	0.0012 V/rpm
k_m	Torque Constant	0.005 N. m/Amp
L_a	Armature Inductance	0 H
K_R	Gear Ratio	40
D_m	viscous friction coefficients	0.12*10-6

Appendix B Robot Operating System (ROS)

B.1 What is ROS?

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. ROS is similar in some aspects to “robot frameworks”, such as Player, YARP, Orocos, CARMEN, Orca, MOOS, and Microsoft Robotics Studio.[36]

The ROS runtime "graph" is a peer-to-peer network of processes (potentially distributed across machines) that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server.[36]

ROS is not a realtime framework, though it is possible to integrate ROS with realtime code. The Willow Garage PR2 robot uses a system called pr2_etherCAT, which transports ROS messages in and out of a realtime process. ROS also has seamless integration with the Orocos Real-time Toolkit.[36]

B.2 How does ROS work?

According to the ROS wiki page [36], ROS has three levels of concepts: the Filesystem level, the Computation Graph level, and the Community level. These levels and concepts are summarized below.

B.2.1 ROS Filesystem Level

The filesystem level concepts mainly cover ROS resources that you encounter on disk, such as:

- **Packages:** Packages are the main unit for organizing software in ROS. A package may contain ROS runtime processes (nodes), a ROS-dependent library, datasets, configuration files, or anything else that is usefully organized together. Packages are the most atomic build item and release item in ROS. Meaning that the most granular thing you can build and release is a package.
- **Metapackages:** Metapackages are specialized Packages which only serve to represent a group of related other packages. Most commonly metapackages are used as a backwards compatible place holder for converted (rosbuild Stacks).
- **Package Manifests:** Manifests (package.xml) provide metadata about a package, including its name, version, description, license information, dependencies, and other meta information like exported packages.
- **Repositories:** A collection of packages which share a common VCS[7] system. Packages which share a VCS share the same version and can be released together using the catkin release automation tool (bloom). Often these repositories will map to converted (rosbuild Stacks). Repositories can also contain only one package.

[7] VCS: Version control system

- Message (msg) types: Message descriptions, stored in (my_package/msg/MyMessageType.msg), define the data structures for messages sent in ROS.
- Service (srv) types: Service descriptions, stored in (my_package/srv/MyServiceType.srv), define the request and response data structures for services in ROS.

B.2.2 **ROS Computation Graph Level**

The Computation Graph is the peer-to-peer network of ROS processes that are processing data together. The basic Computation Graph concepts of ROS are nodes, Master, Parameter Server, messages, services, topics, and bags, all of which provide data to the Graph in different ways.[36]

- **Nodes:** Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale; a robot control system usually comprises many nodes. For example, one node controls a laser range-finder, one node controls the wheel motors, one node performs localization, one node performs path planning, another one provides a graphical view of the system, and so on. A ROS node is written with the use of a ROS client library, such as (roscpp) or (rospy).
- **Master:** The ROS Master provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.
- **Parameter Server:** The Parameter Server allows data to be stored by key in a central location. It is currently part of the Master.
- **Messages:** Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structs).
- **Topics:** Messages are routed via a transport system with publish / subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each other's existence. The idea is to decouple the production of information from its consumption. Logically, one can think of a topic as a strongly typed message bus. Each bus has a name, and anyone can connect to the bus to send or receive messages as long as they are the right type. Figure B.1 shows how data is exchanged between publishers and subscribers through topics.
- **Services:** The publish / subscribe model is a very flexible communication paradigm, but its many-to-many, one-way transport is not appropriate for request / reply interactions, which are often required in a distributed system. Request / reply is done via services, which are defined by a pair of message structures: one for the request and one for the reply. A providing node offers a service under a name and a

client uses the service by sending the request message and awaiting the reply. ROS client libraries generally present this interaction to the programmer as if it were a remote procedure call.

- **Bags:** Bags are a format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

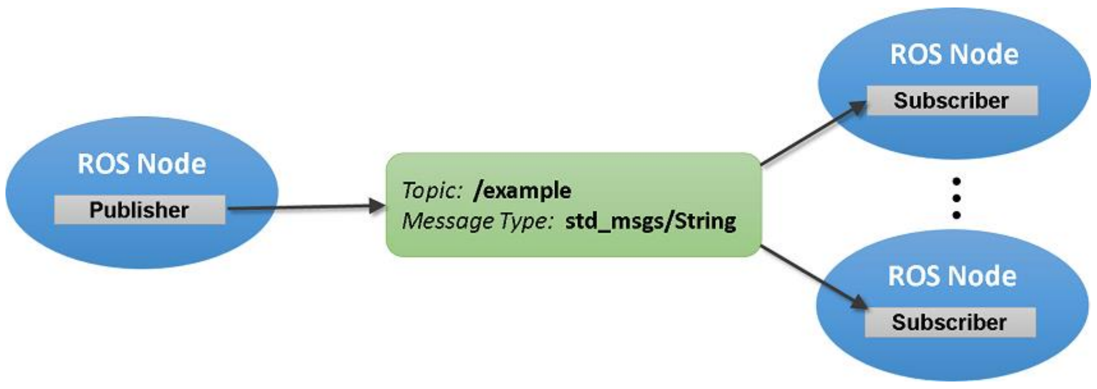


Figure B.1 Data Exchange with ROS Publishers and Subscribers

B.2.3 ROS Community Level

The ROS Community Level concepts are ROS resources that enable separate communities to exchange software and knowledge. These resources include:

- **Distributions:** ROS Distributions are collections of versioned stacks that you can install. Distributions play a similar role to Linux distributions: they make it easier to install a collection of software, and they also maintain consistent versions across a set of software.
- **Repositories:** ROS relies on a federated network of code repositories, where different institutions can develop and release their own robot software components.
- **The ROS Wiki:** The ROS community Wiki is the main forum for documenting information about ROS. Anyone can sign up for an account and contribute their own documentation, provide corrections or updates, write tutorials, and more.
- **Mailing Lists:** The ros-users mailing list is the primary communication channel about new updates to ROS, as well as a forum to ask questions about ROS software.
- **ROS Answers:** A Q&A site for answering your ROS-related questions.
- **Blog:** The (ros.org) Blog provides regular updates, including photos and videos. [36]

B.3 ROS Capabilities

Although ROS requires a computer’s operating system to run, it can perform many functions of an operating system, because of that, ROS is sometimes called a meta operating system. One of the main purposes of ROS is to provide communication between the user, the computer’s operating system, and equipment external to the computer including robots and sensors.[37]

Hardware abstraction is one of the best benefits of ROS, because it gives the user the ability to control robots without the need of knowing all the details about them. For example, one can control the movement of a robot’s arm by issuing a ROS command, or using some scripts written by the designers of the robot.[37]

ROS also provides some simulation options to give the ability to design and simulate your own robot, like Rviz and Gazebo.[37]

B.3.1 **Robots using ROS**

The huge advantages and capabilities of ROS have driven so many companies to develop and manufacture ROS-enabled robots. Here is a list of some of them:

- ABB, Adept, Fanuc, Motoman, and Universal Robots are supported by ROS-Industrial.
- Baxter at Rethink Robotics, Inc.
- HERB developed at Carnegie Mellon University in Intel's personal robotics program.
- Husky A200 robot developed (and integrated into ROS) by Clearpath Robotics.
- PR1 personal robot developed in Ken Salisbury's lab at Stanford.
- PR2 personal robot being developed at Willow Garage.
- Raven II Surgical Robotic Research Platform.
- Shadow Robot Hand, A fully dexterous humanoid hand.
- STAIR I and II robots developed in Andrew Ng's lab at Stanford.
- SummitXL: Mobile robot developed by Robotnik, an engineering company specialized in mobile robots, robotic arms, and industrial solutions with ROS architecture.
- Nao humanoid: University of Freiburg's Humanoid Robots Lab developed a ROS integration for the Nao humanoid based on an initial port by Brown University.
- UBR1 developed by Unbounded Robotics, a spin-off of Willow Garage.
- ROSbot: autonomous robot platform by Husarion.
- Webots: robot simulator integrating a complete ROS programming interface.[38]

For more information on robots using ros, you can refer to [39].

B.3.1.1 TurtleBot 2

The TurtleBot 2 is one of the non-holonomic robots. Holonomic refers to the relationship between controllable and total degrees of freedom of a robot. If the controllable degree of freedom is equal to total degrees of freedom, then the robot is said to be Holonomic, and if the controllable degree of freedom is less than the total degrees of freedom, then it is known as non-holonomic drive.[40]

TurtleBot 2 is differential mobile robot, this means it has two active wheels, and two passive wheel works as a support.

This mobile robot officially proposed by Willow Garage to develop in the operating system dedicated to robotics: ROS. It is equipped with a Kinect sensor, a Netbook, trays for the installation of these two components and a Kobuki base as shown in Figure B.2.[41]

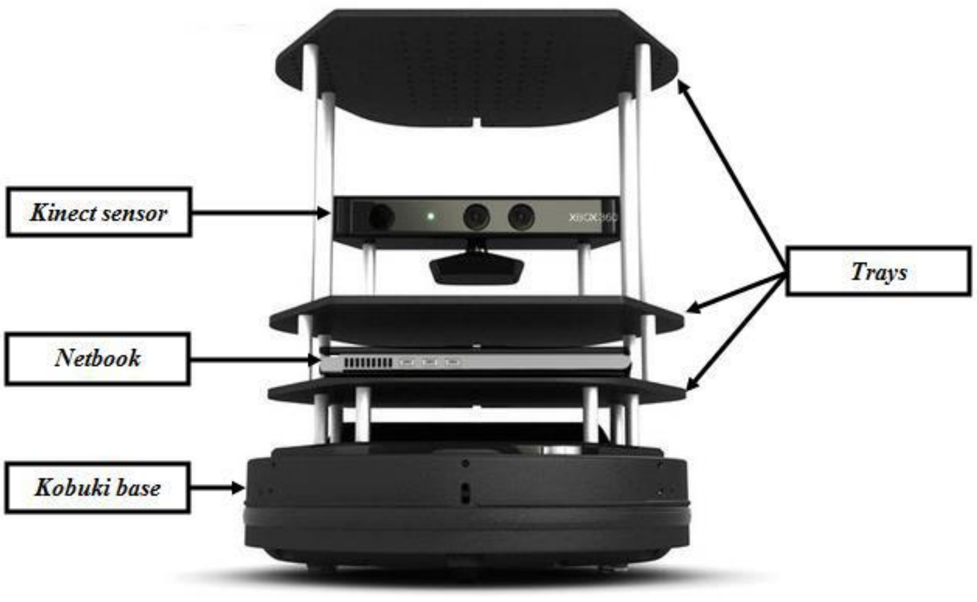
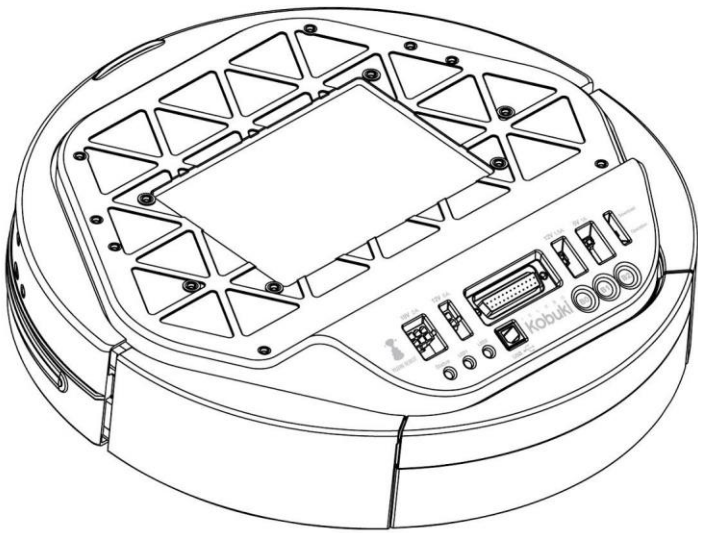


Figure B.2 TurtleBot 2

Figure B.3 shows the anatomy of Kobuki base.

a)



b)

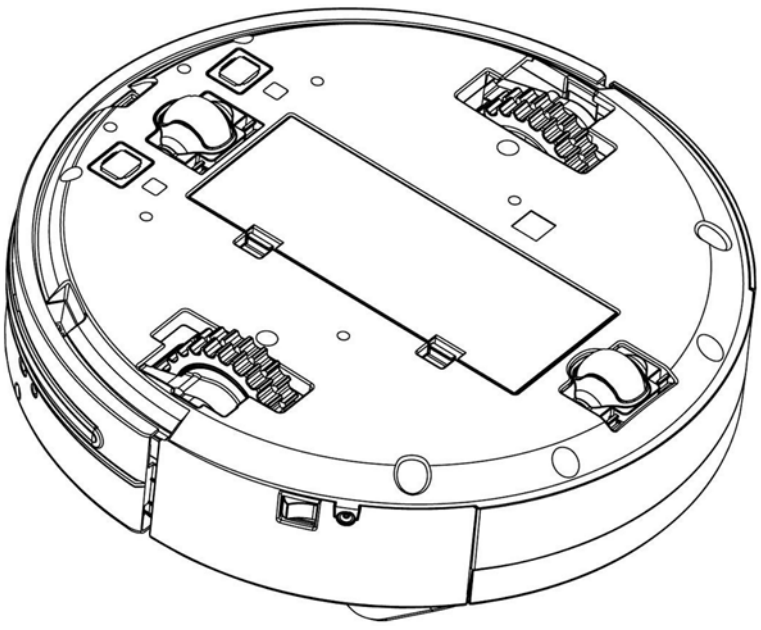


Figure B.3 Anatomy of Kabuki. a) Top view. b) Bottom view.

Figure B.4 shows the main components of Kobuki base.

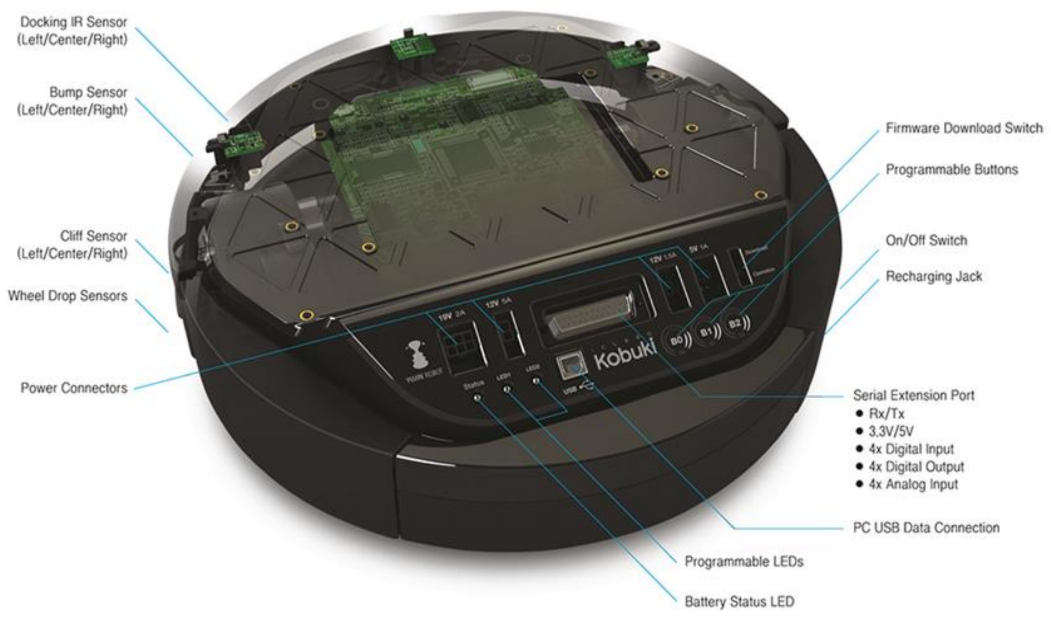


Figure B.4 Kobuki base components

Specifications

- Max. mass of payload, mL = 5 Kg.
- Total mass of robot, m = 5 Kg.
- Dimensions 354 x 354 x 420 mm.
- Radius of wheel, r = 0.038 m.
- Maximum translational velocity, V: 65 cm/s.
- Maximum rotational velocity, W:180 deg/s.
- Threshold Climbing: climbs thresholds of 12 mm or lower.
- Rug Climbing: climbs rugs of 12 mm or lower.
- Ground clearance: 1.5cm.
- Bumpers: left, center, right.[42]

Sensors

1) Cliff Sensor

TurtleBot2 uses 3 Cliff sensors: left, center, right.[42]

A Cliff Sensor is important to have on certain robotic systems to avoid excessive drops that otherwise might damage the robot. TurtleBot2 come with a cliff sensor to help them avoid driving over stairwells or ledges.[43]

A cliff sensor can be mechanical, optical, or even ultrasonic - however they all accomplish the same purpose. The TurtleBot2 uses an optical cliff sensor - which shines an LED onto the ground at an angle that is picked up from a sensor. When the TurtleBot2 comes across a large drop off, the reflected light from the LED is no longer detected into the receiver, and the TurtleBot2 registers the drop off.[43]

Cliff: will not drive off a cliff with a depth greater than 5cm.[42]

2) Wheel drop sensor

TurtleBot2 mobile robot include a wheel drop behavior for detecting floor transitions and drops, as well as gradually sloping obstacles or objects which might not otherwise be detected by bumpers or other obstacle detection sensors.

For example, if the front of the robot rides up on a shallow transition in the floor, or when the front of the robot is lifted by a gradual rise or object on the cleaning surface that does not alert the robot to its presence by triggering a bumper or other sensor, the front wheels of the robot may drop down and thus lose contact with the surface.

3) Encoders: 25700 cps (11.5 ticks/mm)[42]

Encoders are sensors attached to a rotating object (eg. wheels or motors) to measure rotation. By measuring the rotation, the displacement, speed and acceleration of the robot can be determined. These encoders allow location by odometry. The odometry is based on the individual measurement of wheels movements to reconstruct the overall movement of the robot. Starting from a known initial position and integrating the measured displacements, it is then possible to calculate at each instant the current position of the mobile robot.[41]

4) Motor Overload Detection

Disables power on detecting high current (>3A), this is a safety feature in TurtleBot2.[41]

5) gyro sensor

Gyro sensors, also known as angular rate sensors or angular velocity sensors, are devices that sense angular velocity. Gyro sensors sense angular velocity from the Coriolis force applied to a vibrating element. For this reason, the accuracy with which angular velocity is measured differs significantly depending on element material and structural differences. Here, we briefly describe the main types of elements used in vibration gyro sensors.[44]

Gyroscopic sensors used in controlling system allows to monitor exact position of robot. These information can be applied for robot controlling, its autonomous control or its tracking. Inertial navigation is completely autonomous and independent from surroundings, i.e. the system is resistant from external influences as magnetic disturbances, electronically disturbance, signal deformation, etc. For mobile robots to be successful, they have to move safely in environments populated and dynamic. While recent research has led to a variety of localization methods that can track robots well in static environments, we still lack methods that can robustly localize mobile robots in dynamic environments.[45]

Specifications[42]

- 3-Axis Digital Gyroscope.
- Manufacturer : STMicroelectronics.
- Part Name : L3G4200D .
- Measurement Range: ±250 deg/s.
- Yaw axis is factory calibrated within the range of ±20 deg/s to ±100 deg/s.

6) Kinect sensor

Figure B.5 show the main Kinect components. We focus on the following:

a) RGB (Red Green Blue) camera that stores data in three channels with a resolution of 1280x960 pixels. It allows the capture of a color image.[22]

b) Infrared transmitter / receiver (IR) and IR depth sensor. The transmitter emits beams of IR light and the depth sensor reads the IR beams reflected by the obstacles encountered. The reflected beams are converted into depth information thus measuring the distance between the obstacle and the sensor. This technology allows the capture of a depth image.[22]

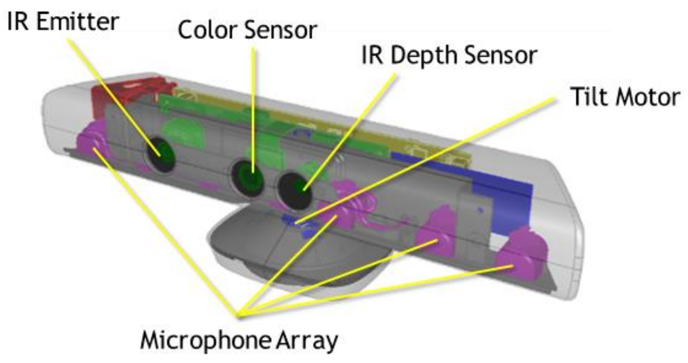


Figure B.5 Kinect components

Table B.1 Some specifications of Kinect

Kinect	Specifications
Viewing angle	Field of View (FoV): 43° vertical x 57° horizontal
Vertical tilt range	±27°
Frame rate (depth and color stream)	30 frames per second (FPS)

The data come from Kinect will send to the laptop by a USB cable which will contain the detection of the object and its location.

7) 2D Laser range finder: model number is URG-04LX-UG01, made by HOKUYO company.

Laser proximity sensors constitute a special case of optical sensors that have a range from centimeters to meters. They are commonly called laser radars (or LIDARs = light direction and ranging sensors). Energy is emitted at impulses. The distance is computed from the time-of-flight. They can also be employed as laser altimeters for obstacle avoidance or for vehicle detection in highways.

The URG-04LX-UG01 scanning laser rangefinder is a low power, small, accurate, high-speed device for obstacle detection. Using the USB interface for power and communication, this unit can obtain measurement data in a 240° field of view up to a distance of 5.6 meters.[46]

We use this LIDAR to make a map for the environment, and after that we will use it for Localization. Figure B.6 shows the LIDAR sensor.

Specification[46]

- Light source: Semiconductor laser diode.
- Scanning range: 0.02 - 5.6m, 240°.
- Measuring accuracy: ±30mm (0.02 - 1m) , ±3% of measurement (1 - 5.6m).
- Angular resolution: 0.352° (360°/1,024 steps).
- Scanning frequency: 10Hz (600rpm).
- Power source: USB bus power.
- Power consumption: 0.5A or less.
- Weight: 160g .
- Size (W X D X H): 50 x 50 x 70 mm.
- 5V operating voltage.

- Resolution: 1 mm.

Figure B.6 LIDAR sensor

Battery and Power System

As we explained before, the TurtleBot2 consists of two motors, Cliff sensors, wheel drop sensor, encoders, gyro sensor, bump sensor and Kinect sensor. All these components will be powered by 2200 mAh Li-Ion battery.

All connectors in the Control Panel are powered by lithium battery to use them in some tasks.

Control Panel [42]

Figure B.7 shows the control panel of Kobuki base.

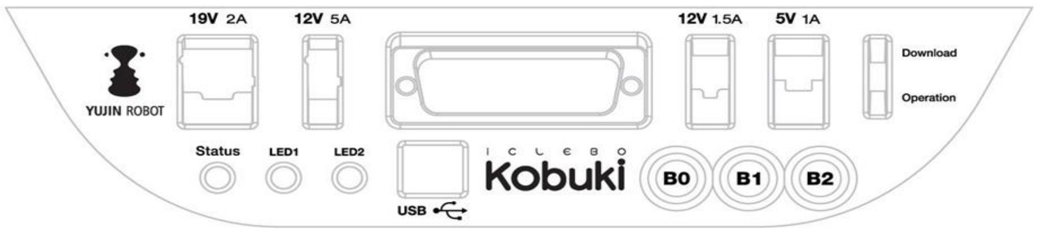


Figure B.7 Control panel for Kobuki

- 19V/2A: Laptop power supply, we can recharge the laptop from this connector (only enabled when robot is recharging).
- 12V/5A: Arm power supply.
- 12V/1.5A: Xbox Kinect power supply.
- 5V/1A: General power supply

The 2D Laser range finder has a different case, which is powered by the laptop through the USB wire.

Power Adapter [42]

Table B.2 shows the input/output specifications of the power adapter.

Table B.2 I/O specifications of power adapter

Input	**Output**
Voltage: 100-240V	Voltage: 19V
Frequency: 50/60Hz	Ampere: 3.16A
Ampere: 1.5A Max	

Appendix C **ROS Adjustable Parameters**

C.1 AMCL Adjustable Parameters

```
<launch>
<node pkg="amcl" type="amcl" name="amcl" output="screen">
<!-- Publish scans from best pose at a max of 10 Hz -->
<param name="odom_model_type" value="diff" />
<param name="odom_alpha5" value="0.1" />
<param name="transform_tolerance" value="0.2" />
<param name="gui_publish_rate" value="10.0" />
<param name="laser_max_beams" value="30" />
<param name="min_particles" value="500" />
<param name="max_particles" value="5000" />
<param name="kld_err" value="0.05" />
<param name="kld_z" value="0.99" />
<param name="odom_alpha1" value="0.2" />
<param name="odom_alpha2" value="0.2" />
<!-- translation std dev, m -->
<param name="odom_alpha3" value="0.8" />
<param name="odom_alpha4" value="0.2" />
<param name="laser_z_hit" value="0.5" />
<param name="laser_z_short" value="0.05" />
<param name="laser_z_max" value="0.05" />
<param name="laser_z_rand" value="0.5" />
<param name="laser_sigma_hit" value="0.2" />
<param name="laser_lambda_short" value="0.1" />
<param name="laser_lambda_short" value="0.1" />
<param name="laser_model_type" value="likelihood_field" />
<!--<param name="laser_model_type" value="beam"/> -->
<param name="laser_likelihood_max_dist" value="2.0" />
<param name="update_min_d" value="0.2" />
<param name="update_min_a" value="0.5" />
<param name="odom_frame_id" value="odom" />
```

```
<param name="resample_interval" value="1" />
<param name="transform_tolerance" value="0.1" />
<param name="recovery_alpha_slow" value="0.0" />
<param name="recovery_alpha_fast" value="0.0" />
</node>
</launch>
```

C.2 gMapping Adjustable Parameters

```
<launch>
  <arg name="scan_topic" default="scan" />
  <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
    <param name="base_frame" value="base_footprint"/>
    <param name="odom_frame" value="odom"/>
    <param name="map_update_interval" value="5.0"/>
    <param name="maxUrange" value="6.0"/>
    <param name="maxRange" value="8.0"/>
    <param name="sigma" value="0.05"/>
    <param name="kernelSize" value="1"/>
    <param name="lstep" value="0.05"/>
    <param name="astep" value="0.05"/>
    <param name="iterations" value="5"/>
    <param name="lsigma" value="0.075"/>
    <param name="ogain" value="3.0"/>
    <param name="lskip" value="0"/>
    <param name="srr" value="0.01"/>
    <param name="srt" value="0.02"/>
    <param name="str" value="0.01"/>
    <param name="stt" value="0.02"/>
    <param name="linearUpdate" value="0.5"/>
    <param name="angularUpdate" value="0.436"/>
    <param name="temporalUpdate" value="-1.0"/>
    <param name="resampleThreshold" value="0.5"/>
    <param name="particles" value="80"/>
    <param name="xmin" value="-50.0"/>
    <param name="ymin" value="-50.0"/>
    <param name="xmax" value="50.0"/>
    <param name="ymax" value="50.0"/>
    <param name="delta" value="0.01"/>
    <param name="llsamplerange" value="0.01"/>
    <param name="llsamplestep" value="0.01"/>
    <param name="lasamplerange" value="0.005"/>
    <param name="lasamplestep" value="0.005"/>
    <remap from="scan" to="$(arg scan_topic)"/>
  </node>
</launch>
```

Appendix D **YOLO Algorithm**

D.1 You Only Look Once (YOLO)

You Only Look Once (YOLO) is a deep neural network that evolutionarily reframed object detection and recognition from RGB images to be a single regression problem directly from the image to object's bounding boxes and class probabilities. YOLO neural network is proved to be very fast, where its base network reaches a processing speed of 45 fps on an Nvidia Titan X GPU. A faster version of the network, which has lower number of layers, reaches a processing speed of 150 fps on the same GPU.

The architecture of YOLO network is inspired by GoogLeNet [47], network that is used for image classification. YOLO network contains 24 convolutional layers, 4 max pooling layers, and 2 fully connected layers, which are architected as shown in these, 30 totally, layers are used in a simple task, which is to detect the objects in a RGB image and recognize their types. YOLO works in a simple way, as shown in Figure D.1, where the input images to the network are resized to 480 x 480, apply the convolutional neural network algorithm on the resized image, and apply a clipping threshold on the output detections' confidence from the network.

The main difference between YOLO and the other approaches for object detection and recognition (i.e. R-CNN) is that YOLO evaluates the whole image in one shot direct to the bounding boxes and class probabilities[48].

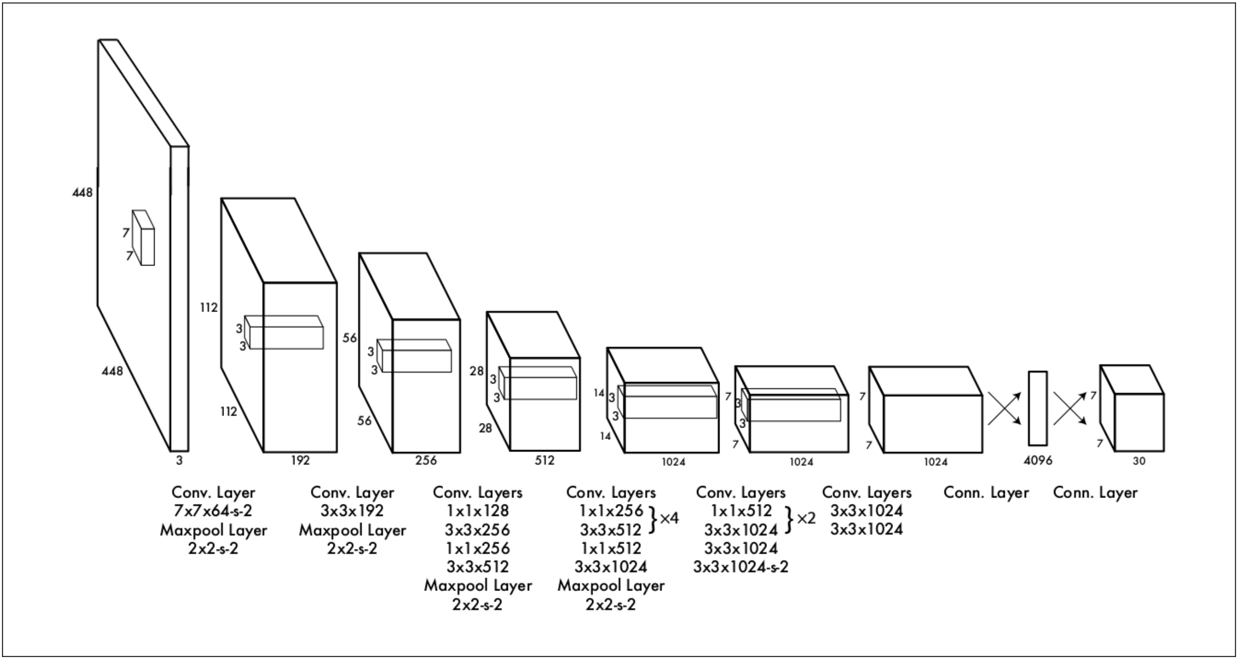


Figure D.1 YOLO Architecture

D.2 How YOLO Works

Here are the fundamental concepts of how YOLO object detection is able to detect an object.

The YOLO detector can predict the class of object, its bounding box, and the probability of the class of object in the bounding box. Each bounding box is having the following parameters:

- The center position of the bounding box in the image (b_x, b_y)
- The width of the box (b_w)
- The height of the box (b_h)
- The class of object (c)

As shown in Figure D.2.

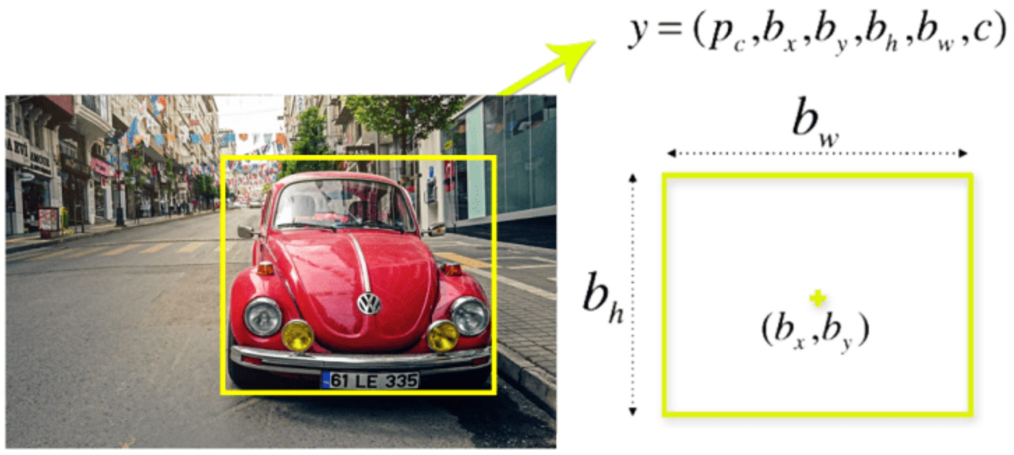


Figure D.2 Parameters of bounding box

Each bounding box is associated with a probability value (pc), it is the probability of a class of object in that bounding box. The YOLO is splitting the image into several cells typically using a 19×19 grid. Each cell is responsible for predicting 5 bounding boxes (there can be one or more objects in a cell). If it does that, it will finally end up in 1805 bounding boxes for one image as shown in Figure D.3.

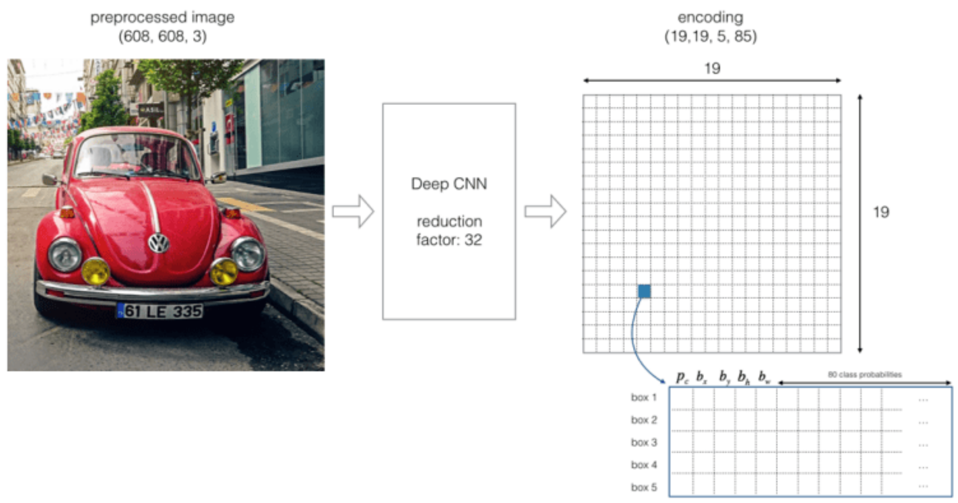


Figure D.3 Splitting the Image by a 19×19 grid

Most of the bounding boxes in the cell may not have an object. The filtration of these bounding boxes is done based on the probability of object class pc. The non-max suppression processes will eliminate the unwanted bounding boxes and only the highest probability bounding boxes will remain As shown in the Figure D.4, it will be used when the image is displayed and the object is recognized[49].

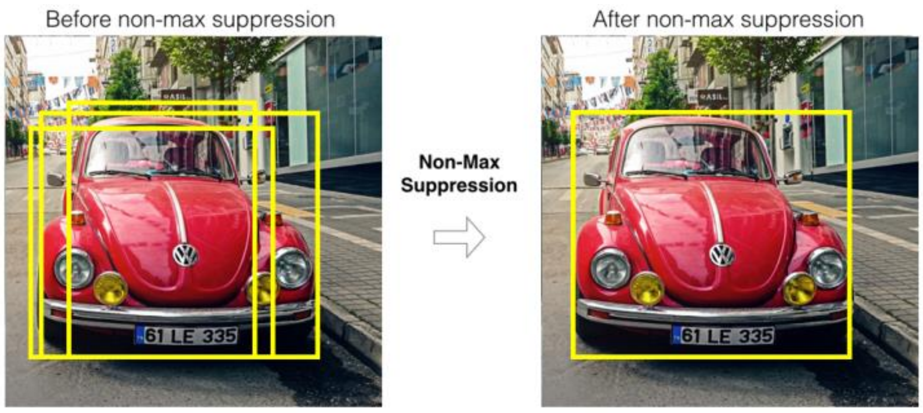


Figure D.4 The Effect of non-max suppression

References

1. Xuexi, Z., et al. *SLAM Algorithm Analysis of Mobile Robot Based on Lidar*. in *2019 Chinese Control Conference (CCC)*. 2019. IEEE.
2. Zarkasi, A., et al. *Implementation of RAM Based Neural Networks on Maze Mapping Algorithms for Wall Follower Robot*. in *Journal of Physics: Conference Series*. 2019. IOP Publishing.
3. **Douglas Henke Dos Reisa, D.W., Marco Antonio De Souza Leite Cuadrosc,** and **a.D.F.T. Gamarraa**, ***Stobot – Storage Robot*. 2019**.
4. Shen, Z., Y. Ma, and Y. Song, *Robust Adaptive Fault-Tolerant Control of Mobile Robots With Varying Center of Mass.* IEEE Transactions on Industrial Electronics, 2018. **65**(3): p. 2419-2428.
5. Liang, X., et al., *Formation Control of Nonholonomic Mobile Robots Without Position and Velocity Measurements.* IEEE Transactions on Robotics, 2018. **34**(2): p. 434-446.
6. Yu, H., et al. *Trajectory tracking control of wheeled mobile robots via fuzzy approach*. in *Proceedings of the 33rd Chinese Control Conference*. 2014. IEEE.
7. Ye, J., *Tracking control of a non-holonomic wheeled mobile robot using improved compound cosine function neural networks.* International Journal of Control, 2014. **88**(2): p. 364-373.
8. Lu, X. and J. Fei. *Velocity tracking control of wheeled mobile robots by fuzzy adaptive iterative learning control*. in *2016 Chinese Control and Decision Conference (CCDC)*. 2016. IEEE.
9. De Luca, A., G. Oriolo, and M. Vendittelli, *Control of wheeled mobile robots: An experimental overview*, in *Ramsete*. 2001, Springer. p. 181-226.
10. Fiorini, P. and Z. Shiller, *Motion Planning in Dynamic Environments Using Velocity Obstacles.* The International Journal of Robotics Research, 1998. **17**(7): p. 760-772.
11. Elfes, A., *Using occupancy grids for mobile robot perception and navigation.* 1989. **22**(6): p. 46-57.
12. Salomon, B., et al. *Interactive navigation in complex environments using path planning*. in *Proceedings of the 2003 symposium on Interactive 3D graphics*. 2003. ACM.
13. Hoffmann, G.M., et al. *Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing*. in *2007 American Control Conference*. 2007. IEEE.
14. Yang, H., et al., *Nonlinear Control for Tracking and Obstacle Avoidance of a Wheeled Mobile Robot With Nonholonomic Constraint.* IEEE Transactions on Control Systems Technology, 2015: p. 1-1.
15. Ahmad Abu Hatab, R.D., *Dynamic Modelling of Differential-Drive Mobile Robots using Lagrange and Newton-Euler Methodologies: A Unified Framework.* Advances in Robotics & Automation, 2013. **02**(02).
16. Diymore. *2WD Robot Smart Car Chassis Kit*. 1992; Available from: https://www.aliexpress.com/item/32667456836.html?spm=a2g0s.9042311.0.0.355e4c4dv5kSyE.
17. ClearPathRobotics, *Turtlebot 2*. 2011.
18. Wikipedia. *Adjacency matrix*. 2018 18 July 2020; Available from: https://en.wikipedia.org/wiki/Adjacency_matrix.
19. Wikipedia. *Stochastic process*. 2019 1 August 2020; Available from: https://en.wikipedia.org/wiki/Stochastic_process.
20. ROS.org, *Adaptive Monte Carlo Localization.* 2019.

21. Biswas, J., M.M.J.R. Veloso, and A. Systems, *Episodic non-Markov localization.* 2017. **87**: p. 162-176.
22. Cong, B.R. and R. Winters, *How Does The Xbox Kinect Work.*
23. Wikipedia. *Robot navigation*. 2019 19 December 2019; Available from: https://en.wikipedia.org/wiki/Robot_navigation.
24. Thrun, S., W. Burgard, and D. Fox, *Probabilistic robotics*. 2005: MIT press.
25. ROS.org. *gmapping*. 2019; Available from: http://wiki.ros.org/gmapping.
26. Mahtani, A., et al., *Effective robotics programming with ROS*. 2016: Packt Publishing Ltd.
27. Wikipedia. *Shortest path problem*. 2019 19 December 2019; Available from: https://en.wikipedia.org/wiki/Shortest_path_problem.
28. Wikipedia. *Dijkstra's algorithm*. 2019 17 December 2019; Available from: https://en.wikipedia.org/wiki/Dijkstra's_algorithm.
29. Science, C. *Dijkstra's Shortest Path Algorithm*. 2016; Available from: https://www.youtube.com/watch?v=pVfj6mxhdMw.
30. https://www.anaconda.com/open-source, *Open Source Community*. 2019.
31. https://web.archive.org/web/20200419034550/https://www.anaconda.com/media-kit/, *About Anaconda*. 2020.
32. https://opencv.org/about/, *About OpenCV*. 2020.
33. https://numpy.org/about/, *About NumPy*. 2020.
34. https://git-scm.com/, *everything-is-local*. 2018.
35. Tzafestas, S.G., *Introduction to mobile robot control*. 2013: Elsevier.
36. ROS.org. *ROS Concepts*. 2018; Available from: http://wiki.ros.org/ROS/Concepts.
37. Fairchild, C. and T.L. Harman, *ROS robotics by example*. 2016: Packt Publishing Ltd.
38. Wikipedia. *Robot Operating System*. 2019; Available from: https://en.wikipedia.org/wiki/Robot_Operating_System.
39. ROS.org. *Robots Using ROS*. Available from: https://robots.ros.org/.
40. PLATFORM, R. *Classification of Robots*. 2018; Available from: http://www.robotplatform.com/knowledge/Classification_of_Robots/Holonomic_and_Non-Holonomic_drive.html.
41. Bensaci, C., Y. Zennir, and D. Pomorski. *Nonlinear Control of a differential wheeled mobile robot in real time-Turtlebot 2*. 2018.
42. Garage, W., *Kobuki - User Guide*. 2017.
43. Robopedia. *Cliff Sensor*. 2017; Available from: http://www.robotappstore.com/Robopedia/Cliff%20Sensor-2.html.
44. Corp, S.E. *Gyro sensors - How they work and what's ahead*. 2019; Available from: https://www5.epsondevice.com/en/information/technical_info/gyro/.
45. Qazizada, M.E. and E.J.P.E. Pivarčiová, *Mobile robot controlling possibilities of inertial navigation system.* 2016. **149**: p. 404-413.
46. Hokuyo, *Hokuyo URG-04LX-UG01 Scanning Laser Rangefinder*. 2018.
47. Szegedy, C., et al. *Going deeper with convolutions*. in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.
48. Redmon, J., et al. *You only look once: Unified, real-time object detection*. in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.
49. https://robocademy.com/2020/05/01/a-gentle-introduction-to-yolo-v4-for-object-detection-in-ubuntu-20-04/, *A Gentle Introduction to YOLO v4 for Object detection in Ubuntu 20.04*. 2020.