Palestine Polytechnic University

Deanship of Graduate Studies and Scientific Research

Master of Mechatronics Engineering

Single Board Computer ROS-Based Tennis Balls Collecting Mobile Robot

Submitted By

Mohammed Omar Al-Qaisi

Thesis submitted in partial fulfillment of requirements of the degree

Master of Mechatronics Engineering

January, 2022

In partial fulfillment of the requirements for the degree of Master in Mechatronics Engineering.

Graduate Advisor Committee:

Prof.: Karim Tahboub

(Supervisor), Palestine Polytechnic University.

Signature: ____________ Date: 25.1.2023

Dr.: Iyad Hashlamoun

(Internal committee member), Palestine Polytechnic University.

Signature: ____________ Date: 26.1.23

Dr.: Ahmad Balasi

(External committee member), Birzeit University.

Signature: ____________ Date: 10.03.2023

Thesis Approved by:

Name: Dr. Nafesh Nasereddin

Dean of Graduate Studies and Scientific Research

Palestine Polytechnic University

Signature:……………………………………

Date:……15..02..2023….

ACKNOWLEDGMENTS

First and foremost, I would like to express my heartfelt gratitude to Prof. Dr. Karim Tahboub for his consistent patience, support, encouragement, and inspiring guidance throughout my thesis.

I would also like to express my sincere appreciation to Dr. Ahmad Balasi (Birzeit University) and Dr. Iyad Hashlamoun (Palestine Polytechnic University) who kindly served on committee members for their insightful remarks.

A special word of thanks goes to my friends for their unwavering and unending support throughout my master's studies.

Finally, I could not have finished this thesis without the support of my family, especially my parents, who have been there for me throughout my life.

TABLE OF CONTENTS

ACKNOWLEDGMENTS 1

TABLE OF CONTENTS 2

LIST OF FIGURES 4

LIST OF TABLES 6

LIST OF ABBREVIATIONS 7

THESIS ABSTRACT 9

1 Introduction 10

1.1 Introduction 10

1.1.1 Problem Statement 11

1.1.2 Motivation 11

1.1.3 Objectives 12

1.1.4 Research Methodology 12

1.2 Previous Studies 14

1.2.1 ROS Based Mobile Robot 14

1.2.2 Tennis Balls Collecting Mobile Robot 21

1.2.3 Summary 22

1.3 Background 24

1.3.1 Robot Operating System 24

1.3.2 Path Planning and Trajectory Generation 29

1.3.3 A* Search Algorithm and Path Generation 31

1.3.4 Vision and Objects Detection 33

1.4 Thesis Organization 40

2 Robot Design and Analysis 41

2.1 Kinematics 44

2.2 Dynamics 47

2.3 Robot Sensors and Actuators 50

3 Robot Vision and Motion 53

3.1 Tennis Mobile Robot Vision 55

3.2 Motion Planning 57
3.3 Path Planning and Trajectory Generation 66
3.4 Motion Control 71
4 ROS Design and Implementation 73
4.1 Robot System ROS Design 73
4.1.1 System Nodes 74
4.1.2 System Topics 79
4.2 Implementations 80
4.2.1 Hardware Implementation 80
4.2.2 Real Time System Considerations 83
4.3 Constraints and Solutions 84
5 Results and Discussion 87
5.1 Object Detection and Classifications using YOLO Algorithm 87
5.2 Depth Data 90
5.3 Workspace Mapping 91
5.4 Path Finding with A* Search Algorithm 94
5.5 Path Planning between Wye Points Through Cartesian Polynomials 95
5.6 Robot Trajectories 97
5.7 Motion Control 100
6 Conclusion and Future Research Topic 101
6.1 Conclusion 101
6.2 Future Work 102
References 103

LIST OF FIGURES

Figure 1.1: The whole scheme of the proposed system.. 15
Figure 1.2: AMR schematic diagram.. 15
Figure 1.3: System framework diagram.. 16
Figure 1.4: Human following Mobile robot.. 17
Figure 1.5: System hardware diagram.. 17
Figure 1.6: Physical setup and block diagram of the system.. 18
Figure 1.7: System schematic diagram.. 18
Figure 1.8: System block diagram.. 19
Figure 1.9: Turtlebot mobile robot used raspberry pi as the main processing unit.. 19
Figure 1.10: System architecture.. 20
Figure 1.11: System block diagram.. 20
Figure 1.12: System hardware architecture.. 21
Figure 1.13: Collecting mechanism.. 21
Figure 1.14: Overview of the system implementation.. 22
Figure 1.15: 2D grid map with g cost and h cost allocated for each cell. 32
Figure 1.16: Blue cells represent the shortest path found by the A* algorithm. 32
Figure 1.17: Traditional object detection steps. 33
Figure 1.18: Examples of one and two stage detectors. 34
Figure 1.19: R-CNN object detection overview. 34
Figure 1.20: Output image from Yolo detector.. 36
Figure 1.21: input image divided into S x S grid.. 37
Figure 1.22: Predicted bounding box.. 38
Figure 1.23: YOLO algorithm detection steps.. 39
Figure 1.24: Depth Image (Right) in comparison with the original image (Left). 40
Figure 2.1: Tennis-Balls mobile robot. (1: Tennis ball sack, 2: collecting mechanism).. 42
Figure 2.2: System block diagram. 43
Figure 2.3: Geometry of differential drive robot. 44
Figure 2.4: Components of joint actuating system. 50
Figure 3.1: Tennis robot main algorithm. 53
Figure 3.2: Detailed block diagram. 54
Figure 3.3: YOLO algorithm. 55
Figure 3.4: Robot Workspace. 58
Figure 3.5: Robot field of view. 59
Figure 3.6: Local cost matrix generation Algorithm. 61
Figure 3.7: Local cost matrix of the workspace map. 62
Figure 3.8: Distance cost matrix of the workspace map. 63
Figure 3.9: Total cost matrix which represent the value of f(v) for each cell. 63
Figure 3.10: A* search directions. 64
Figure 3.11: A* algorithm flowchart. 64
Figure 3.12: Free obstacles path (sequence of green cells) as generated by the A* algorithm. 65

Figure 3.13: A* Path converted to an arc path. 66
Figure 3.14: Algorithm for Path planning between each adjacent cells in the A* path generated. 70
Figure 3.15: General scheme for the wheel space control. 71
Figure 4.1: ROS system design. Ovals represent the system nodes while rectangles represent the system topics. 74
Figure 4.2: Camera Node Algorithm. 75
Figure 4.3: Image setting configuration parameters. 75
Figure 4.4: Snippet code from YOLO xml launch file. 76
Figure 4.5: Snippet code for defining the addresses. 77
Figure 4.6: $I2C$ initialization command. 77
Figure 4.7: Configuring the DC motor controller to run in constant speed mode. 77
Figure 4.8: Transmitting velocity values to the controller. 78
Figure 4.9: Robot hardware block diagram. 81
Figure 4.10: RPI4B Hardware overview.. 82
Figure 4.11: $I2C$ Raspberry Pi4 detection map. 85
Figure 4.12: $I2C$ detection map after opening $0x01$ address. 86
Figure 5.1: An input image (left) is applied to the YOLO algorithm and an output image (right) with each detected object. 88
Figure 5.2: Example of a generated data from YOLO algorithm for a detected object (Bottle). 88
Figure 5.3: YOLO output image with a tennis ball detection. 89
Figure 5.4: Detected objects classes and their coordinates in the image shown in figure 5.3. 89
Figure 5.5: Depth Image (Right) in comparison with the original image (Left). 90
Figure 5.6: Illustration of objects location within the robot workspace. 91
Figure 5.7: Local cost matrix (left) and the distance cost matrix (Right). 92
Figure 5.8: Total cost matrix as calculated by the ROS node. 93
Figure 5.9: Total cost matrix with the free-obstacles path (Blue cells). 94
Figure 5.10: Interpolation between each two adjacent cells. $Ki = Kf = 0.2$. 96
Figure 5.11: Total A* path after arcing. 97
Figure 5.12: Total velocity profile. 98
Figure 5.13: Right wheel angular velocity. 99
Figure 5.14: Left wheel angular velocity. 99
Figure 5.15: Step response of the closed loop system. 100

LIST OF TABLES

Table 1.1: Previous Studies Comparison table…………………………………………………………………………………..23

Table 4.1: Nodes Execution Times………………………………………………………………………………………………..83

LIST OF ABBREVIATIONS

ROS	**Robot Operating System**
PC	**Personal Computer**
CPU	**Central Processing Unit**
GPU	**Graphical Processing Unit**
SBC	**Single Board Computer**
PID	**Proportional-Plus-Integral-Plus-Derivative**
PWM	**Pulse Width Modulation**
SQL	**Structured Query Language**
GPIO	**General Purpose Input Output**
USB	**Universal Serial Bus**
I^2C	**Inter Integrated Circuit**
OS	**Operating System**
GHz	**Giga Hertz**
MHz	**Mega Hertz**
GSM	**Global System for Mobile Communications**
SMS	**Short Message Service**
RGB-D	**Red Green Blue-Depth**
2D	**Two-Dimensional**
3D	**Three-Dimensional**
FPGA	**Field Programmable Gate Array**
SoC	**System on a Chip**
ARM	**Advanced RISC Machine**
MCU	**Microcontroller Unit**
IOU	**Intersection Over Union**
YOLO	**You Only Look Once**
R-CNN	**Region-based convolutional neural network**

SVM	**Support Vector Machine**
CNN	**Convolutional Neural Network**
RPN	**Region Proposal Network**
TWD	**Two Wheeled Differential**
SDK	**Software Development Kit**
rad	**Radian**
Sec.	**Seconds**
NUC	**Next Unit of Computing**
LiDAR	**Light Detection and Ranging**
ZigBee	**Zonal Intercommunication Global Standard**
API	**Application Programming Interface**
SLAM	**Simultaneous Localization and Mapping**
RPI4B	**Raspberry Pi 4 Model B**

THESIS ABSTRACT

With the recent increases of mobile robot deployment that rely on robot operating system (ROS), new challenges have emerged as a result of the hardware requirements imposed by ROS on the host computer. Installing ROS on a mobile robot requires the target robot to be equipped with a full personal computer (PC) with specific specifications. However, deploying ROS on such PC's will introduce new issues such as increased size, weight, cost, and power consumption.

This research presents the development and implementation of a fully integrated standalone tennis balls collecting mobile robot using ROS. The operating system is deployed on a compact, low-cost, low power consumption, light weight and embedded single board computer (Raspberry Pi 4). The robot goal is to assist playground attendees by collecting scattered tennis balls. This is accomplished by integrating and implementing a miniature series of algorithms that construct the robot tasks. These algorithms are used to detect objects, classify them, plan optimal paths, and avoid obstacles. During the implementation process, a significant challenge arose in the form of a high computational load on the main processing unit (CPU). The vision detection algorithm is to blame for this. This was resolved by using a lighter version of the algorithm, which reduced the computational load.

The proposed method was investigated in this work. The results show that a single board computer (Raspberry Pi 4) can complete the required objectives and run the algorithms within acceptable constraints. The vision algorithms performed as expected, detecting all of the objects in the robot workspace. However, the Raspberry Pi requires a longer execution time than a standard PC to perform vision tasks. The extra time is due to the Raspberry Pi's hardware resource limitations, as well as the limitation on utilizing hardware acceleration abilities. Keep in mind that hardware acceleration employs the graphical processing unit to address vision algorithms in order to shorten execution time. Furthermore, the A* algorithm was used to help the robot find the shortest obstacle-free path. Other algorithms are in charge of formulating the wheel's trajectory and control law. All of the robot algorithms were coded to use fewer computational resources, resulting in less extra execution time. As a result, the robot is able to complete the tasks in a reasonable amount of time. Finally, the proposed low-cost solution was shown to be capable of running ROS-based mobile robot algorithms.

1

Introduction

1.1 Introduction

Recently, mobile robot technology has advanced dramatically, allowing robots to function autonomously in a variety of roles. Such robots are mobile robots that can perform multiple tasks at the same time. The required robot should execute multiple algorithms in parallel in order to perform the desired tasks, such as object detection and picking up, effective and intelligent collaboration with humans, mapping its surroundings and motion planning, and so on. The development of these robots presents a variety of challenges, resulting in complex designs in robot software and hardware.

Traditional robots had an on-board software system that was installed on a dedicated computer and was used to operate a specific hardware device using hardware-specific firmware developed by a company. This system could only use the manufacturer's services, resulting in a lack of hardware and software modularity and a lengthy development time for robotics applications. As a result, many standardized robot software platforms are introduced in robotics research to accelerate robot development by maximizing code reuse. As a result, many standardized robot software platforms are introduced to accelerate robot development in the same way that a smart phone is developed. The smart phone development system enables the creation of new applications with little knowledge of the existing hardware.

A platform like this is the robot operating system (ROS), which enables robotics engineers to develop robotics applications in a short period of time and at a low cost. ROS is a collection of software libraries and tools that aid in the development of robot applications by providing services such as hardware abstraction, low-level device control, implementation of commonly-used functionalities, message passing between processes, and package management. In order to implement the desired algorithms related to specific robot tasks, the ROS platform must be deployed on a general-purpose computer with specific specifications. To run ROS, the computer must have a host operating system (Linux) installed. Traditional computers are large, expensive, and consume a lot of power. Because there are no fully supported ROS compact-embedded single board computers (SBC), these computers are used. These disadvantages will eventually reduce the robot's mobility.

Tennis ball collecting mobile robot is one of the real-life mobile robots that needs to be developed by increasing system integration (software and hardware). This robot is in charge of detecting tennis balls, obstacles, and determining the optimized path to the tennis ball.

1.1.1 Problem Statement

The process of designing a mobile robot necessitates the design of its hardware and software to meet the desired functionality. The software is in charge of controlling the robot actuators based on data from the robot sensors and specific algorithms in order to complete a task. Because the software is dedicated to the current robot hardware, it is difficult to use codes written for another robot, even if they perform the same tasks. As a result, the engineers must write all codes from scratch, not to mention the expertise associated with the computers and controllers used, as well as the programming languages. As a result, development time and cost are not optimized.

As a result, the process of developing optimized, hardware-specific computer architectures for robotics systems can be time consuming and complex, acting as a stumbling block to continued robotics innovation.

Installing a ROS on a mobile robot to make it fully integrated necessitates the target robot having a full computer with specific specifications in order to run the host operating system, such as Linux, which is responsible for running the ROS.

However, putting a PC on a mobile robot will cause issues such as extra weight, size, cost, and power consumption. Which is why minimizing them is regarded as a critical criterion for designing a mobile robot, particularly a tennis collecting mobile robot. As a result, the tennis collecting mobile robot should have an embedded computer (single board computer SBC) that meets the minimum software and hardware requirements of ROS. To be light, small, low-cost, and power-efficient.

1.1.2 Motivation

The main goal of this thesis is to develop a fully integrated, standalone tennis balls collecting mobile robot using ROS on a low-cost, compact, and embedded SBC (Raspberry Pi 4 (RPI4B)). As previously stated, the proposed method will aim to overcome the challenges raised and associated with mobile robot development. These difficulties arose as a result of the wide range of tasks that a mobile robot must perform and were implemented using the traditional robotics platform (Non ROSified software platforms). Traditional platforms are incapable of meeting the minimum requirements for robot hardware modularity. The modularity of the hardware should allow the robot development environment to be fast enough. As a result, a new software platform was created to be run on a full host computer. Although the same software is not yet supported on single board computers. Because of the constraints and limitations associated with its software compatibility with ROS, hardware architecture, central processing unit, and graphical processing unit, integrating ROS on an embedded SBC like Raspberry Pi is a critical challenge.

1.1.3 Objectives

The primary objective of this dissertation is to develop a ROS-based tennis balls collecting mobile robot. The robot is aimed to achieve a series of tasks such as objects detection and path planning. The main goal to be reached by achieving a miniature series of goals, which are:

- Develop a fully integrated standalone Tennis balls collecting mobile robot.
- Develop/adapt algorithms corresponding to the required robot tasks and implement them with ROS.
- Install a SBC (RPI4B) on the robot and deploy ROS on it using a standard operating system (Linux)

1.1.4 Research Methodology

To achieve the study's main goals, an engineering research and development approach will be used, with the following phases:

Phase I: Literature review

Investigate the research and studies that have been conducted in the field of deploying ROS on a single board computer, as well as the capability of implementing a mobile robot tasks algorithm with the proposed hardware. Algorithms for such tasks include object detection and localization, environment mapping, generating an obstacle-free path, and trajectory control.

Phase II: Specifying the Requirements

Determine the robot requirements that must be met in order to meet the main objectives; these requirements must provide a solution for the player to collect the tennis balls:

1. Objects detection and classifications.
2. Optimize the power consumption: Optimize the navigation path between the detected balls.
3. Avoid collision with obstacles for a given map during navigation.

Phase III: System Design

The system will be developed to meet the following tasks based on the literature review and the requirements:

1. **Image Processing**

   Detect tennis balls and obstacles by identifying the shape and color of the tennis balls using image feature extraction. The detection process will use artificial intelligence based on a classifier to compare the detected objects with previously saved tennis ball models.

2. **Control system design**

   To ensure that the robot moves at the desired velocity, a speed controller will be designed and implemented using a PI controller or a control module that has already been tuned for the given DC motors.

3. **Optimization and path generation**

   Solve the problem of minimum time navigation between the detected tennis balls and the robot. The optimization problem will be solved by utilizing a well-known optimized search algorithm. These are A-star (A*) search algorithms. The generated path is an optimized, obstacle-free path.

4. **Robot dynamics**

   Derive the equation of motion using Lagrange method, both forward and inverse dynamics for the computation of the required torques by the wheels motors to derive the robot with the required velocity.

5. **Mechanical structure**

   Checking and, if necessary, modifying the structure's strength, precision, and appearance.

1.2 Previous Studies

ROS has been used since its beginnings in 2010, and many mobile robots have been developed using the ROS environment. The deployment of ROS on mobile robots is typically carried out using general purpose computers with sufficient hardware resources. Memory, a powerful processing unit, a graphical processing unit, and so on. This chapter provides an overview of the traditional methods for deploying ROS on a mobile robot computer. Aside from a comparison of host computer specifications and independence. For example, size, weight, cost, and power consumption, as well as collaboration with other computers. By independence, we mean that all robot tasks and algorithms must be executed on the robot's SBC without the use of any external resources, such as another computer or server.

This chapter provides an overview of the most relevant studies to ROS-based mobile robots and their deployment. Furthermore, a discussion and review of their specific advantages and disadvantages in terms of implementation and cost. In addition, a quick rundown of the most recent tennis ball collecting robots.

1.2.1 ROS Based Mobile Robot

ROS-based mobile robots are developed by implementing all robot tasks in the ROS environment and employing various deployment methods. SBC, minicomputers, and even general-purpose laptop computers are examples. Some mobile robots are standalone and autonomous, capable of performing all tasks on the robot computer. Others connected to other external resources (such as servers) to carry out any heavy tasks, such as image processing tasks. In [1], the author builds a Raspberry Pi-based mobile robot for object recognition. The ROS middleware is used to build functions such as mobile robot movement and Kinect sensor image capture. As the control kernel for the mobile robot, the system runs ROS on a Raspberry Pi-based computer. The proposed system, depicted in Figure 1.1, perceives its surroundings using an image sensor. The system is built on ROS and uses the Python programming language to allow the image sensor to capture environmental data and upload it to a structured query language (SQL) database. The SQL server was used to run the object detection and classification algorithms. The extracted data will then be sent back to the computer to be used in the navigation algorithm. Because this technique relies on an external server to perform the highly computational algorithms, the robot is not completely self-contained. The paper [2] describes the design and implementation of an autonomous wheeled mobile robot for exploring air ventilation ducts. The robot moves autonomously through the ducts, gathering data that will be used to create a duct map. The robot is outfitted with various sensors that enable it to properly interact with the unknown ventilation ducts and achieve the overall exploration goal. Such as a laser scanner, two infrared sensors for specific obstacle detection, and a camera to capture video footage of the environment while in motion that can be analyzed offline to detect obstructions and pollution, allowing for optimal maintenance and cleaning operations planning. The mobile robot was created using ROS and deployed on a SBC

(Raspberry Pi 2). The robot does not perform image processing or object detection. It is regarded as a low-cost standalone mobile robot.

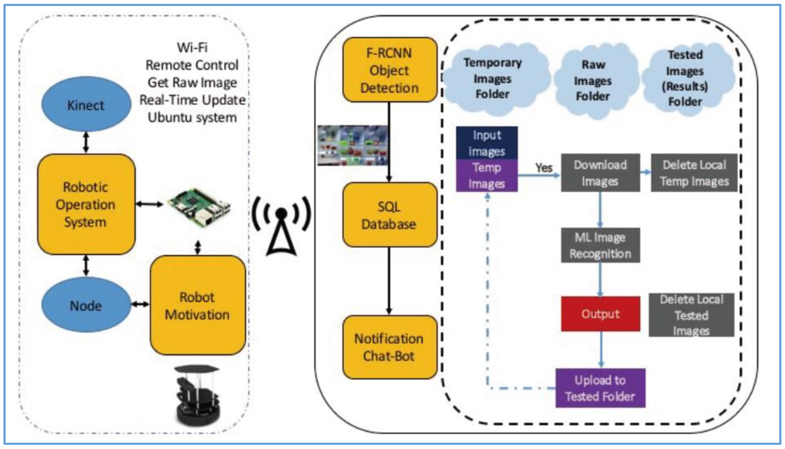


Figure 1.1: The whole scheme of the proposed system. [1]

In [3], a mobile robot is designed to navigate autonomously in a dynamic environment. The robot is outfitted with a 2D LIDAR sensor as well as a depth camera. Two ROS navigation stack system configurations are proposed. The first system is built on a Raspberry Pi 3 and only uses 2D LiDAR. The second system is built on an Intel NUC and employs 2D LiDAR and an RGB-D camera. There are no object detections or classifications. Both sensors' data are used to generate a navigation map. The software is implemented using ROS, which is installed on a Raspberry Pi 3. Due to the lack of object detection with an RGB camera, a Raspberry Pi 3 is sufficient to complete the tasks. [4] describes a ROS-adapted autonomous mobile robot (AMR) capable of navigating in an unpredictable environment. AMRs can sense the parameters of their environment, build a model of it, and then locate themselves within it. The AMR is outfitted with ultrasonic, inertial measurement unit (IMU), and LIDAR sensors. The primary control units are a SBC and a microcontroller. The universal serial bus (USB) is used for communication between the single board computer and the control unit. Figure 1.2 depicts the AMR's schematic design. For object detection, this robot does not use any image sensors.

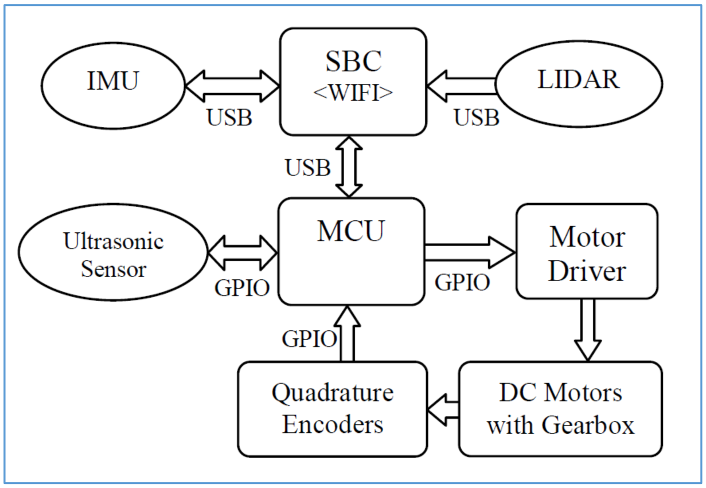


Figure 1.2: AMR schematic diagram.[4]

The author of [5] creates a smart home service robot system based on ROS. The robot is in charge of wireless home appliance control, voice remote control, autonomous positioning and navigation, liquefied gas leakage detection, and human infrared detection alarm. The main control unit of the system is a Raspberry Pi 3B. The robot is outfitted with numerous sensors to collect the necessary data. A ZigBee network that collects home environmental data and controls electrical appliances in the home. The USB camera collects image information about the house. Baidu Speech Recognition API recognizes human speech. When a dangerous situation arises, the GSM module is used to send SMS and phone alarms to users. The bottom controller, an Arduino mega 2560, is used to control the movement of the service robot. The LiDAR and attitude sensor create an indoor environment map of the home. The system schematic diagram is depicted in Figure 1.3. To classify objects in the captured image, the proposed system does not employ any object detection algorithm.

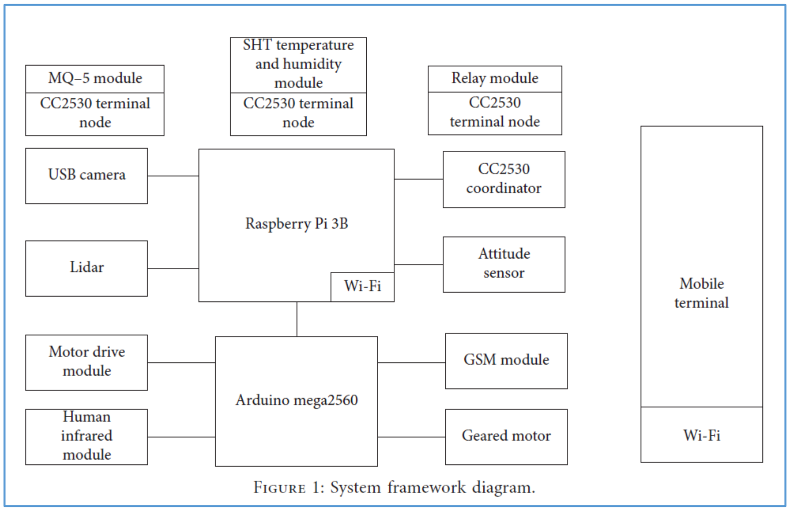


Figure 1.3: System framework diagram.[5]

[6] proposes a human-following mobile robot, with the mobile robot in responsible of detecting and tracking a person. The robot is outfitted with the required sensors, including an RGB-D camera and a LiDAR sensor. A suitable tracking method is introduced and implemented on the person following algorithm. Face detection, leg detection, color detection, and person blob detection are among the tracking methods used. ROS is used for the implementation of all algorithms. As the main processing unit, a laptop is mounted on the robot. This design provides powerful image analysis resources for classifying and detecting objects within the captured image. The proposed system is depicted in Figure 1.4. The disadvantages of using a laptop include increased weight, size, power consumption, and cost.

In [7], a ROS-based autonomous vehicle is implemented as a mock car on a (Field programmable gate array) FPGA board. As shown in Figure 1.5, the car control system is uniquely designed and implemented on a system-on-a-chip (SoCs) that contains both an FPGA and a CPU. A control system for a ROS-based autonomous vehicle can be efficiently implemented by utilizing such

SoCs. This technique creates a high-performance processing unit capable of efficiently performing all of the proposed tasks. However, the presence of the FPGA board makes it costly.

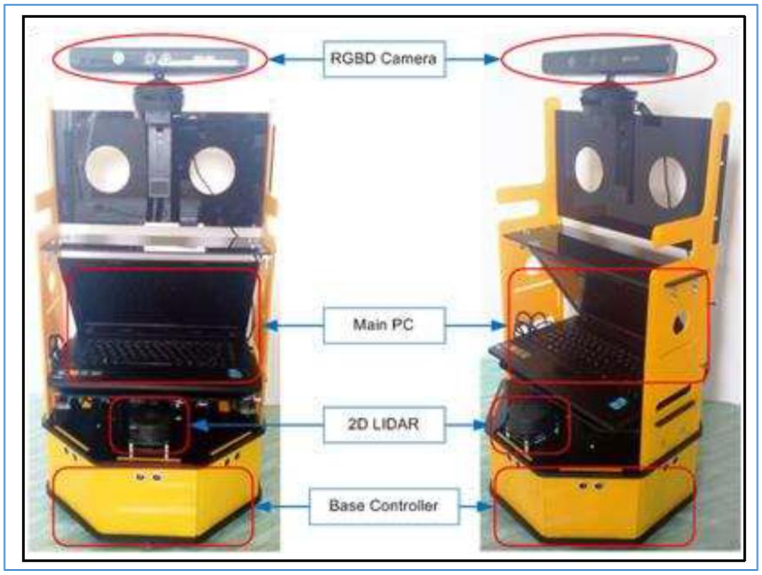


Figure 1.4: Human following Mobile robot.[6]

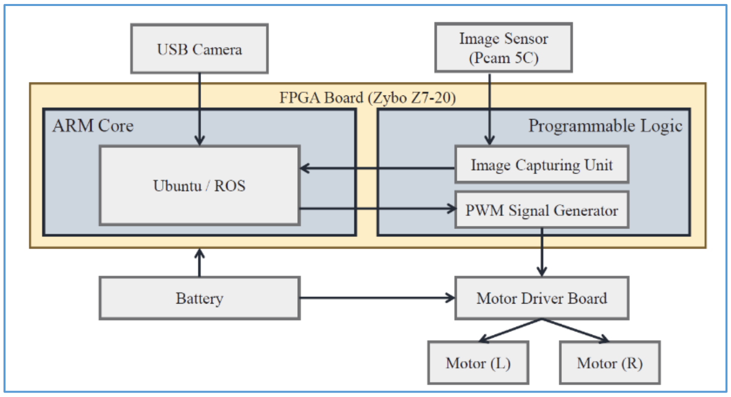


Figure 1.5: System hardware diagram.[7]

The paper [8] describes the development of an autonomous mobile robot for radiation mapping using a commercial mobile robot platform, Turtlebot2, in conjunction with ROS. Using Simultaneous Localization and Mapping (SLAM), a mobile robot can generate a physical map of the targeted environment. It measures the level of radiation intensity at distributed sampling points within a predefined region of interest. This robot is equipped with navigation sensors such as an on-board RGB-D camera, gyro sensors, wheel drop sensors, bump sensors, and cliff sensors. The main control and processing unit of the robot is a laptop. Figure 1.6 depicts the system's physical setup and block diagram. The map is created by the robot using depth image sensor data. This project contains no image processing tasks. Object detection and classification, for example. The laptop installed on the mobile robot provides powerful resources for completing all tasks. This method adds more weight, size, power consumption, and cost.

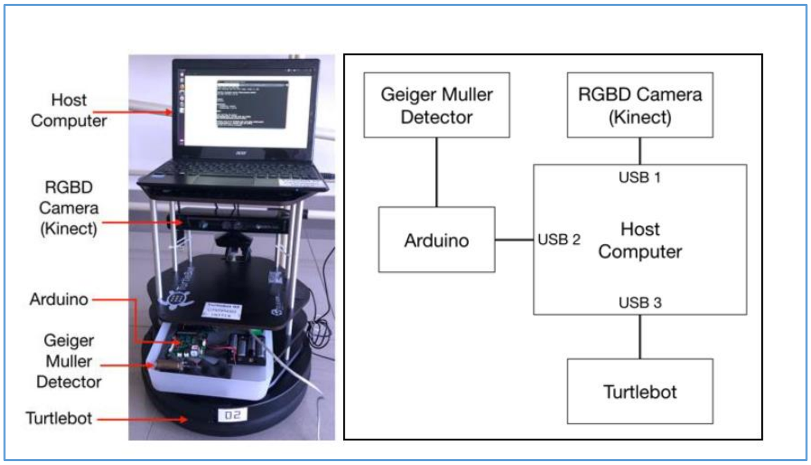


Figure 1.6: Physical setup and block diagram of the system.[8]

This paper [9] describes the creation of an embedded computing system for a mobile robot. As shown in Figure 1.7, the embedded system is made up of an Advanced RISC Machine (ARM) processor computer and an FPGA board. The GPIO’s are used to communicate between the ROS installed and deployed on the ARM processor (Raspberry Pi 3) and the FPGA. The study concentrated on using the FPGA board to accelerate the robot application, sensor data collection and analysis, and algorithm executions. Verilog programming code is used to implement robot-computing functions in FPGA. Because of the combination of two processing units, the FPGA board and the Raspberry Pi, this approach is considered an expensive design.

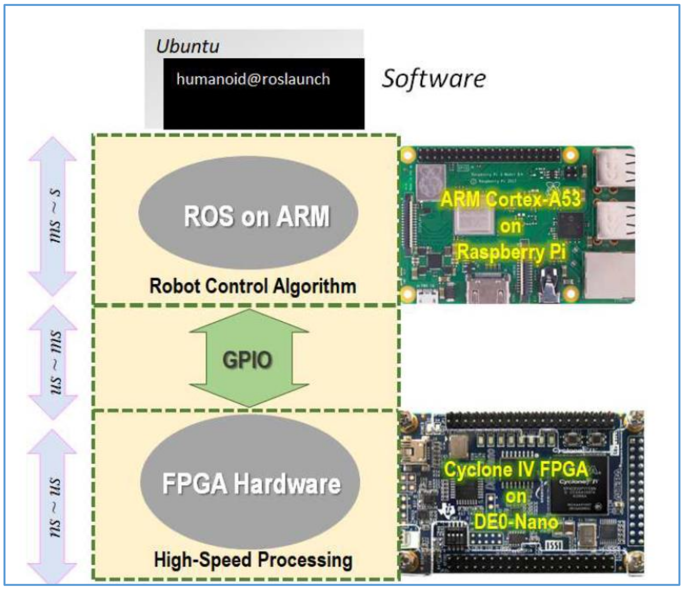


Figure 1.7: System schematic diagram.[9]

The author of [10] propose a low-cost and high-performance Ackerman structure positioning and navigation control system for wheeled mobile robots based on ROS and LiDAR. The LiDAR sensor is used to collect distance information from objects in the vehicle's vicinity. As the core controller, an industrial control computer was used, and the Ubuntu operating system was installed on it. The STM32 microcontroller served as the driving part's control core. The core control unit of this system is a personal computer. As a result, it adds weight, size, cost, and power consumption. The system block diagram is depicted in Figure 1.8.

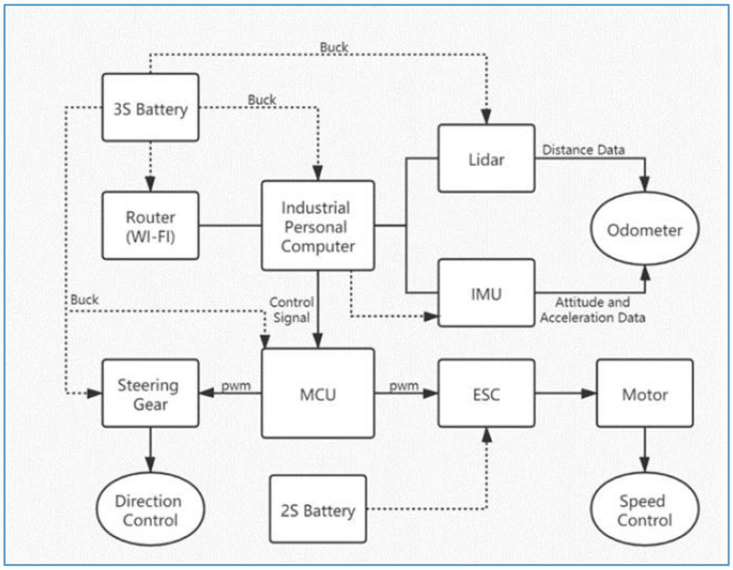


Figure 1.8: System block diagram.[10]

In [11], the 3D sensor is used to create an indoor map using SLAM and 3D models for surrounding objects with a Turtlebot robot. The Raspberry Pi 3 serves as the main processing unit and serves as a replacement for the laptop that was previously used to control the Turtlebot. There are no object detection or classification tasks performed. The proposed low-cost solution is capable of running the ROS-based SLAM algorithm. The proposed system is illustrated in Figure 1.9.

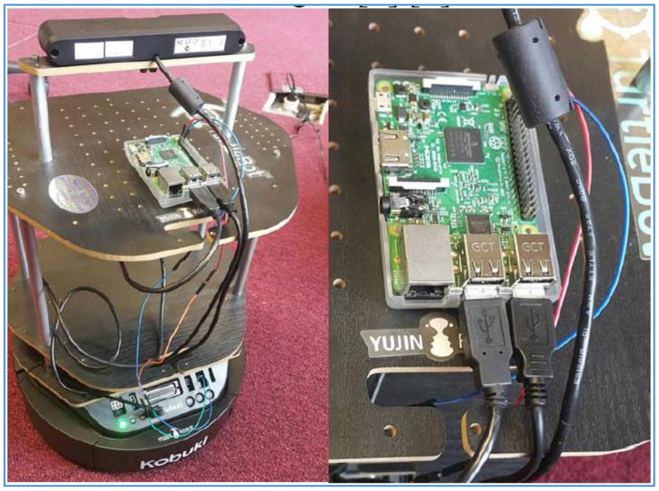


Figure 1.9: Turtlebot mobile robot used raspberry pi as the main processing unit.[11]

[12] proposes a mobile robot for real-time remote control for human detection, tracking, and verification. A system such as this is useful for security monitoring, data collection, and experimentation. An RGB-D camera is used in the design of human detection and tracking as a visual sensing device. ROS was used to implement the robot tasks. The main processing unit for image processing is a laptop. This method establishes a powerful processing unit that can handle all robot tasks. Simultaneously, it adds a cost, size, weight, and power consumption. The proposed system is shown on Figure 1.10.

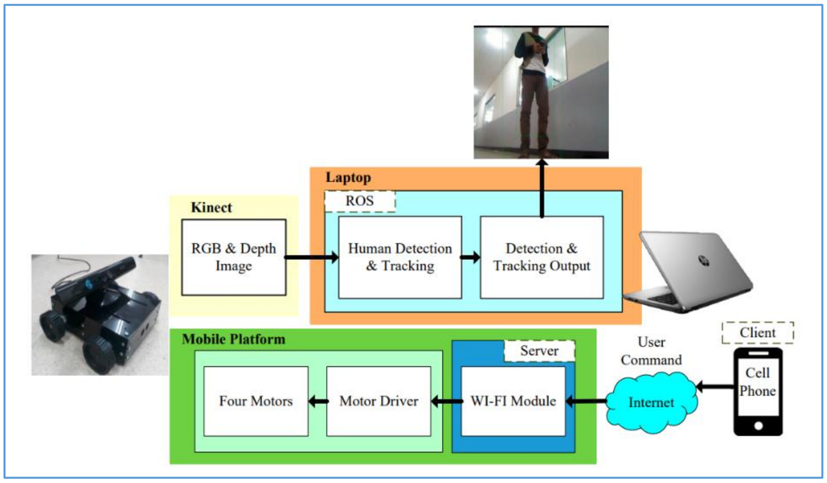


Figure 1.10: System architecture.[12]

An Omni-directional mobile robot equipped with a LiDAR sensor was developed for mapping a room in this study [13]. A SBC (Raspberry Pi 3 B) is used to access the LiDAR sensor, as shown in Figure 1.11. The LiDAR data is then wirelessly transmitted to a laptop for processing into a map. ROS has integrated this laptop and the SBC. This system is completely stand alone and does not rely on image sensors or algorithms to detect objects.

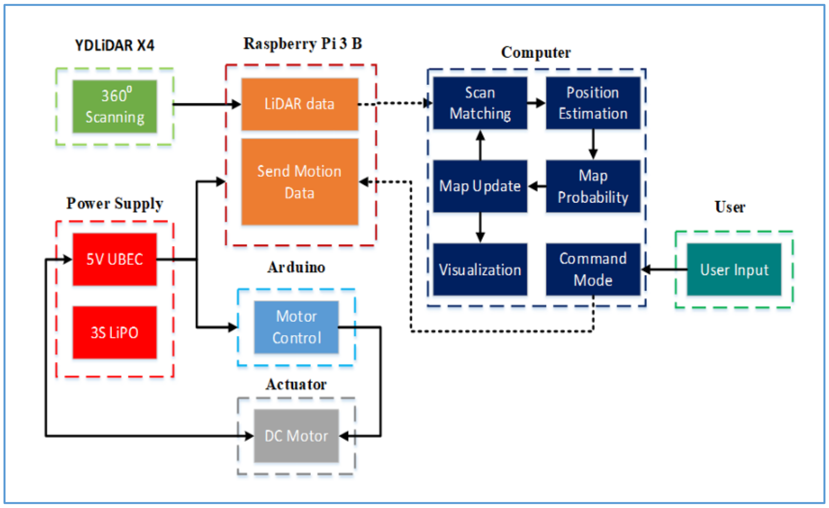


Figure 1.11: System block diagram.[13]

In [14], a control system and navigation are used on a mobile robot, as shown in Figure 1.12. To perform localization and navigation tasks, the system employs a laser scanner to perceive its surroundings. The main processing unit of the robot is a laptop, with real-time control provided by a microcontroller. ROS is used to implement all robot tasks. For object detection, this system does not employ any vision-based algorithms. There was no use of a single-board computer.

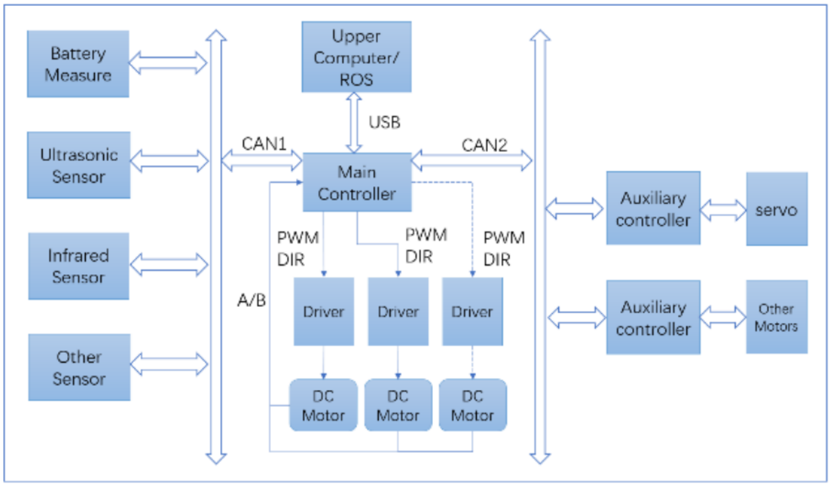


Figure 1.12: System hardware architecture.[14]

1.2.2 Tennis Balls Collecting Mobile Robot

This section provides an overview of the most relevant studies on the tennis ball collecting mobile robot. The study [15] proposed a vehicle for collecting tennis balls. The new collector can be controlled interactively by an Android smartphone via Bluetooth communication, or it can automatically collect tennis balls along a predetermined route. The collecting mechanism is primarily determined by the size of the tennis balls, as illustrated in Figure 1.13. The vehicle is unable to distinguish between tennis balls and other objects of the same size. Because the vehicle does not have a vision system. Furthermore, the vehicle cannot detect the location or absence of balls. It will continue to move and wait until it picks up any random balls. As a result, the system is not optimized.

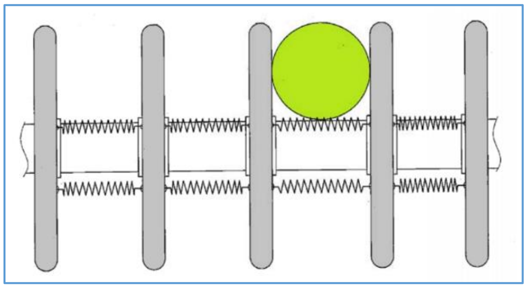


Figure 1.13: Collecting mechanism.[15]

This paper proposes a vision-aided ball retrieving robot that can collect tennis balls in a single run while avoiding obstacles in [16]. As shown in Figure 1.14, the system relies on a camera installed above the field. A live video feed from an overhead camera of the tennis court is processed in order to identify the robot, user, balls, and obstacles for the best path planning to cover the entire court. This method is only applicable to fields equipped with an overhead camera. As a result, the robot is not completely portable.

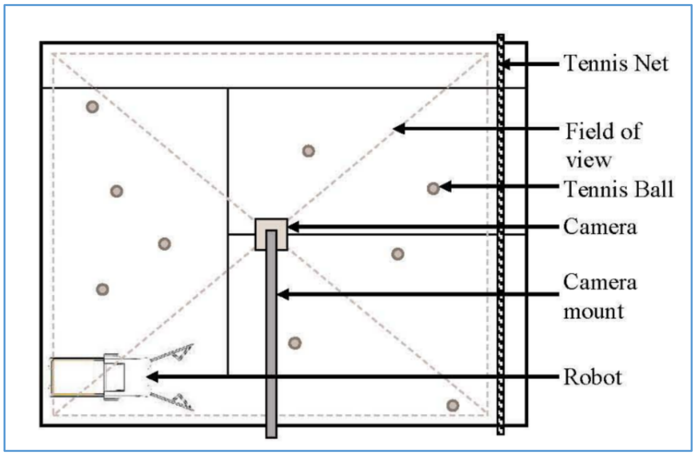


Figure 1.14: Overview of the system implementation.[16]

A novel mechanical structure of a ball collecting device is designed and fabricated in [17]. To actuate and steer the robot, a two-wheel differential driving mechanism with an omnidirectional wheel is used. To provide navigation information to the robot, LiDAR and an RGB-D camera are used to detect environmental information as well as scattered balls in the training field. For efficient ball gathering, a novel impeller structure is designed and installed in front of the mobile robot. To achieve target identification, positioning, and tracking, an RGB-D visual sensor is used. For robot navigation, an A* search algorithm-based path planner has been used.

1.2.3 Summary

This chapter has covered the background topics relevant to the thesis's main topic. ROS-based mobile robots, ROS deployment on SBC (Raspberry Pi), and standalone fully integrated mobile robots using ROS with Raspberry Pi are among the topics covered. According to the literature, there is no complete implementation of a standalone fully integrated mobile robot with ROS that is also deployed on a SBC. In this manner, the mobile robot is capable of performing all required tasks independently without the use of any external computing units. Table 1.1 compares all of the proposed methods presented in this literature. The proposed implementation in this thesis uses the SBC to perform more tasks than those proposed by the literature methods, by avoiding the use of any external computing units such as servers or the integration of single board computers with FPGA boards. Furthermore, most literature methods avoid the implementation of cutting-edge vision-based algorithms capable of assisting the robot in perceiving its environment. Specifically, vision-based tasks are regarded as time-consuming computation tasks. The goal of this thesis is to develop a fully integrated ROS-based mobile robot using a SBC.

Table 1.1: Comparison table of the previous methods.

Method	Software	Hardware	Performed Tasks			Low-Cost	Bulky	Standalone
			Vision(Object Detection)	Mapping	Navigation			
[1]	ROS+SQL	SBC+SQL Server	Yes	Yes	Yes	No	No	No
[2]	ROS	SBC	No	Yes	Yes	Yes	No	Yes
[3]	ROS	SBC+Mini PC	No	Yes	Yes	No	Yes	Yes
[4]	ROS	SBC	No	Yes	Yes	Yes	No	Yes
[5]	ROS	SBC	No	Yes	Yes	Yes	No	No
[6]	ROS	Laptop	Yes	Yes	Yes	No	Yes	Yes
[7]	ROS	ARM Core +FPGA Board	Yes	Yes	Yes	No	No	Yes
[8]	ROS	Laptop	No	Yes	Yes	No	Yes	Yes
[9]	ROS+Verylog HDL	SBC+FPGA Board	Yes	Yes	Yes	No	Yes	Yes
[10]	ROS	Industrial Computer	No	Yes	Yes	No	Yes	Yes
[11]	ROS	SBC	No	Yes	Yes	No	No	Yes
[12]	ROS	Laptop	Yes	No	No	No	Yes	Yes
[13]	ROS	SBC	No	Yes	Yes	Yes	No	Yes
[14]	ROS	Laptop	No	Yes	Yes	No	Yes	Yes

1.3 Background

1.3.1 Robot Operating System

1.3.1.1 Introduction

ROS is a collection of libraries and tools used to program a robot's application, ranging from drivers to cutting-edge algorithms. ROS is regarded as a powerful developer tool because it contains everything required to create a robotics project. It is also classified as open source software. [18]

Robot technology has advanced dramatically in recent years. New robots are capable of performing multiple tasks concurrently and autonomously. Many challenges in robot design have arisen as a result of the complexities in hardware and software design. The mechanical structure of the robot, actuators, and various mounted sensors should all be considered in the hardware design. The mounted sensors provide the robot with self-awareness of its surroundings. Measure the robot's dynamic states and observe how it interacts with its surroundings. The new software design should include multiple algorithms that run in parallel to process sensor data and make decisions about how to perform the desired tasks.

The need for a solution to handle the new robot's technology in terms of complexity and parallelism necessitates the introduction of new platforms. The new platform will have the ability to accelerate the development of robotics applications. Hardware abstraction, low-level device control, sensing, recognition, navigation, manipulations, and package management, debugging, and development tools are examples of such applications.

Hardware abstraction occurs in tandem with the software platform. This enables developers to create robot applications without prior knowledge of robot hardware. As a result, software engineers who are not experts in robotics can contribute to the development of robotics. It's the same as developing mobile apps without knowing the hardware configuration.

ROS is one of these software platforms, it is a Meta-operating system that runs on existing operating systems such as Windows and Ubuntu. ROS uses a virtualization layer between applications and distributed computing resources to perform tasks like scheduling, loading, monitoring, and error handling. ROS, as a meta-operating system, provides a platform for users to develop and manage new applications, as well as application packages for various purposes. ROS has created an ecosystem that distributes packages developed by users.

1.3.1.2 Importance of ROS

ROS is needed because of its ability to reduce robot development time. Furthermore, ROS does not necessitate the development of an existing system and the reprogramming of everything. ROS can easily convert a non-ROSified robot to a ROSified one by inserting a few standardized codes. This improves hardware modularity and code reusability. Furthermore, ROS allows users to manipulate, modify, or create a feature that they are interested in or want to contribute to while leaving other features without any modifications. This can eventually reduce development time. Furthermore, the significance of ROS is derived from its characteristics, which are depicted below. [18]

1. **Reusability of the programs:** A robotics developer can concentrate on one feature while setting up the necessary packages for the remaining functions. Following the development process, developers can share the finished program with one another. This enables developers to divide each developed function and feature into nodes. In ROS, nodes are the smallest unit of computation; this leads to modularization, which increases hardware and software modularity.[19]

2. **ROS is a communication-based program:** Data should be exchanged among nodes after they have been built, and ROS has all of the necessary protocols for doing so. Furthermore, network programming data-which is extremely useful for remote control-allows nodes to be connected even if they are deployed on different hardware.

3. **Support of development tools:** ROS provides software tools required for robot development, easing the development process. Debugging tools, 2D visualization, 3D visualization, and robot model visualization are examples of such tools. Furthermore, ROS records data during experiments and replays it when necessary. For example, graphs and vectors results can be recorded. [19]

4. **Active community:** ROS developers provide a community that allows users to easily emphasize and collaborate. Collaboration, as previously stated, is simple and integrable due to the aforementioned ROS characteristics. As a result, over 5000 packages have been developed and shared voluntarily. And the WikiROS pages that explain their packages have surpassed 18000 pages. Furthermore, the number of posts in the Q&A forums has surpassed 36000, resulting in a collaborative and growing community. The community goes beyond discussing instruction to consider what ROS should entail for the advancement of robotics and collaborates to fill in the gaps in the puzzle. [19]

5. **Formation of the ecosystem:** ROS is shaping its ecosystem by allowing robot hardware developers and robot application software developers to collaborate on projects. Consider smartphones. Their revolutionary development was facilitated by the ecosystem created by software platforms such as Android or iOS. The ROS ecosystem is discussed further in the following sections.

1.3.1.3 Objectives of ROS

ROS main goals is to create a development environment that allows robotic software developers to collaborate on a global scale. As a result, the emphasis is on maximizing code reuse and integrating both the software platform and the hardware platform. The following characteristics help to achieve this goal.

1. **Distributed process:** Processes are coded as nodes, which are the smallest executable unit processes, and each node runs independently while exchanging data systematically. [19]

2. **Package management:** Multiple nodes can be managed as a package, making it simple to use, develop, share, modify, and redistribute. [19]

3. **Public repository:** Each package could be made available on a variety of public repositories, such as GitHub. [19]

4. **Support various programming languages:** ROS programs can be written in any programming language, including C++, Python, and Lisp. Furthermore, it is possible to create a package composed of many nodes, each of which is programmed in a different language. [19]

1.3.1.4 Layers of ROS

The various layers of ROS are explained further below.

1. **Client library:** Code collection that makes the job of the ROS programmer easier. It takes many ROS concepts and makes them code-accessible. Allow you to write ROS nodes, for example. A library of this type can be written in any programming language.

2. **Robotics application framework layer:** Used to create robotics service applications.

3. **Robotics Application:** Is a service application based on the robotics application framework.

4. **Communication Layer:** It serves as the primary means of communication and messaging between ROS nodes. It is in responsible of data transmission and reception.

5. **Hardware interface layer:** Are collections of routines that allow programs to access hardware resources via programmable interfaces.

1.3.1.5 ROS Echo System

The echo system is the framework that connects hardware manufacturers, operating system developers, and app developers with end users. Smartphone manufacturers, for example, create devices that support hardware interfaces with the operating system, and operating system companies develop a generic library to operate devices from various manufacturers. As a result, software developers can create applications for a variety of devices without having to understand hardware. Robotics is also developing an echo system. Various hardware technologies flooded the market at first, but there was no operating system to integrate them. Several software platforms arose, and ROS drew enough attention to support the development of an ecosystem.

1.3.1.6 Concepts of ROS

ROS terminology as given in WikiROS [19]

ROS Master: For node-to-node connections and message communication, the ROS master serves as a name server. When ROS master is launched, a registration process for each node's name is carried out. Without the master, there can be no connection between nodes or message communication, such as topics and services.

Node: It is the smallest processor unit in ROS (one executable program). It is recommended to develop for easy reusability by using a single node for each purpose. In the case of mobile robots, for example, the program to operate the robot is divided into specialized functions. Sensor drive, sensor data conversion, obstacle recognition, motor drive, encoder input, and navigation are all performed by specialized nodes. Based on the registered information, a node can act as a publisher, subscriber, service server, or service client, and nodes can exchange messages using topics and services. According to the application ROS design, ROS is capable of executing all nodes concurrently, sequentially, or a combination of both.

Package: ROS fundamental unit is the package. The ROS application is built in packages, and each package contains either a configuration file or a node to launch other packages or nodes. The package also includes all of the files required to run the package, such as ROS dependency libraries for running various processes, datasets, and a configuration file.

Message: A message is used by a node to send or receive data between nodes. Messages are variables that can be integers, floating points, or Booleans.

Topic: It is a one-way messaging protocol that is used to exchange data between nodes. The publisher node first registers its topic with the master before beginning to publish messages on it. Subscriber nodes that want to receive the topic request the publisher node's information corresponding to the name of the topic registered in the master. Based on this information, the subscriber node establishes a direct connection with the publisher node in order to exchange messages as a topic. Topic communication is a protocol for asynchronous communication. Once connected, the topic continuously transmits and receives a stream of messages; it is frequently used for sensors that must transmit data on a regular basis.

Publish and Publisher: The term "publish" refers to the action of transmitting relevant messages related to the topic. The publisher node broadcasts a message to all connected subscriber nodes interested in the same topic.

Subscribe and Subscriber: The term 'subscribe' refers to the action of receiving related messages related to the topic. The subscriber node receives publisher information from the master, which publishes related topic. The subscriber node directly requests connection to the publisher node and receives messages from the connected publisher node based on the publisher information received.

1.3.2 Path Planning and Trajectory Generation

Flatness Based Path and Trajectory Generation [21]

A trajectory vector $\bar{\boldsymbol{q}}(t)$ for $t \in [t_i, t_f]$ (where t_i & t_f represents the initial and final time, respectively) will be generated that drives a mobile robot from an initial position vector $\bar{\boldsymbol{q}}(t_i) = \bar{\boldsymbol{q}}_i = (x_i, y_i, \theta_i)$ to a final position vector $\bar{\boldsymbol{q}}\left(t_f\right) = \boldsymbol{q}_f = (x_f, y_f, \theta_f)$. The trajectory is divided into a geometric path $\bar{\boldsymbol{q}}(s)$ and a timing law $s = s(t)$. Where s is varied between $s(t_i) = s_i$ and $s\left(t_f\right) = s_f$. One way of choosing s is to take the arc length along the path as the parameter s. As a result, $s_i = 0$ and $s_f = L$, where L is the length of the path. We choose s to be a normalized variable expressed as a polynomial function in time.

s is a normalized variable which has values between 0 and 1. Where:

When $t = 0 \ \rightarrow s = 0$ (1.1)
And when $t = t_f \ \rightarrow s = 1$ (1.2)

And all other values $0 < t < t_f$ are mapped to corresponding values in $0 < s < 1$. Furthermore, there is a one-to-one mapping between the state trajectories in the s-domain and t-domain. Which means at any time instant t:

$$x(t) = x\left(s(t)\right) \tag{1.3}$$

$$y(t) = y\left(s(t)\right) \tag{1.4}$$

$$\theta(t) = \theta\left(s(t)\right) \tag{1.5}$$

The space-time separation implies that:

$$\dot{\bar{\boldsymbol{q}}} = \frac{d\bar{\boldsymbol{q}}}{ds}\frac{ds}{dt} = \grave{\boldsymbol{q}}\,\dot{s} \tag{1.6}$$

The above equation allows us to express the desired output trajectory in terms of the path parameter s, ($x(s), y(s)$). Therefore:

$$\dot{x}(t) = \frac{dx}{ds}\frac{ds}{dt} = \grave{x}\,\dot{s} \tag{1.7}$$

$$\dot{y}(t) = \frac{dy}{ds}\frac{ds}{dt} = \grave{y}\,\dot{s} \tag{1.8}$$

To satisfy the following initial and final conditions of position, velocity and acceleration, a timing law of fifth order polynomial function is considered.

$$s(t) = \alpha_0 + \alpha_1 t + \alpha_2 t^2 + \alpha_3 t^3 + \alpha_4 t^4 + \alpha_5 t^5 \tag{1.9}$$

$$s(0) = 0, \quad s(t_f) = 1$$

$$\dot{s}(0) = v_i, \quad \dot{s}(t_f) = v_f$$

$$\ddot{s}(0) = a_i, \quad \ddot{s}(t_f) = a_f$$

Where:

v_i : Represents the initial linear velocity.

v_f : Represents the final linear velocity.

a_i : Represents the initial linear acceleration.

a_f : Represents the initial linear acceleration.

Finding the coefficients from the boundary conditions, we have:

$$\begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 & t_i & {t_i}^2 & {t_i}^3 & {t_i}^4 & {t_i}^5 \\ 1 & t_f & {t_f}^2 & {t_f}^3 & {t_f}^4 & {t_f}^5 \\ 0 & 1 & 2t_i & 3{t_i}^2 & 4{t_i}^3 & 5{t_i}^4 \\ 0 & 1 & 2t_f & 3{t_f}^2 & 4{t_f}^3 & 5{t_f}^4 \\ 0 & 0 & 2 & 6t_i & 12{t_i}^2 & 20{t_i}^3 \\ 0 & 0 & 2 & 6t_f & 12{t_f}^2 & 20{t_f}^3 \end{bmatrix}^{-1} \begin{bmatrix} s_i \\ s_f \\ v_i \\ v_f \\ a_i \\ a_f \end{bmatrix} \tag{1.10}$$

Planning via Parametrized Cartesian Polynomials [21]

To plan a path in such way which satisfies the boundary conditions, interpolation schemes will be used. Such as using the cubic polynomials that automatically satisfy the boundary conditions on $x \ and \ y$.

$$x(s) = a_0 + a_1 s + a_2 s^2 + a_3 s^3 \tag{1.11}$$

$$y(s) = b_0 + b_1 s + b_2 s^2 + b_3 s^3 \tag{1.12}$$

Such that:

$$x(s_i) = x_i \qquad , y(s_i) = y_i$$
$$x(s_f) = x_f \qquad , y(s_f) = y_f$$
$$\dot{x}(s_i) = \dot{x}_i \qquad , \dot{y}(s_i) = \dot{y}_i$$
$$\dot{x}(s_f) = \dot{x}_f \qquad , \dot{y}(s_f) = \dot{y}_f$$

Finding the coefficients from the boundary conditions, we have:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & s_i & {s_i}^2 & {s_i}^3 \\ 1 & s_f & {s_f}^2 & {s_f}^3 \\ 0 & 1 & 2s_i & 3{s_i}^2 \\ 0 & 0 & 2s_f & 3{s_f}^2 \end{bmatrix}^{-1} \begin{bmatrix} x_i \\ x_f \\ \dot{x}_i \\ \dot{x}_f \end{bmatrix} \tag{1.13}$$

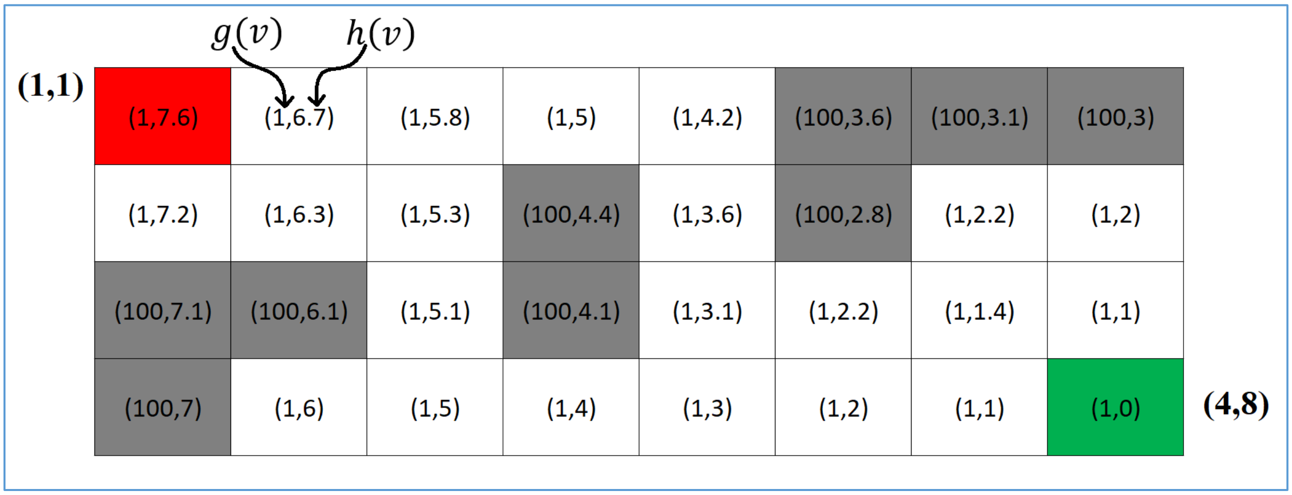
$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 & s_i & {s_i}^2 & {s_i}^3 \\ 1 & s_f & {s_f}^2 & {s_f}^3 \\ 0 & 1 & 2s_i & 3{s_i}^2 \\ 0 & 0 & 2s_f & 3{s_f}^2 \end{bmatrix}^{-1} \begin{bmatrix} y_i \\ y_f \\ \dot{y}_i \\ \dot{y}_f \end{bmatrix} \tag{1.14}$$

1.3.3 A* Search Algorithm and Path Generation

A* search algorithm is one of the best-known motion planning and path finding algorithms. It uses a combination of heuristic searching and searching based on the shortest path. Because each cell in the task space is evaluated by the value $(f(v))$[21, 22]:

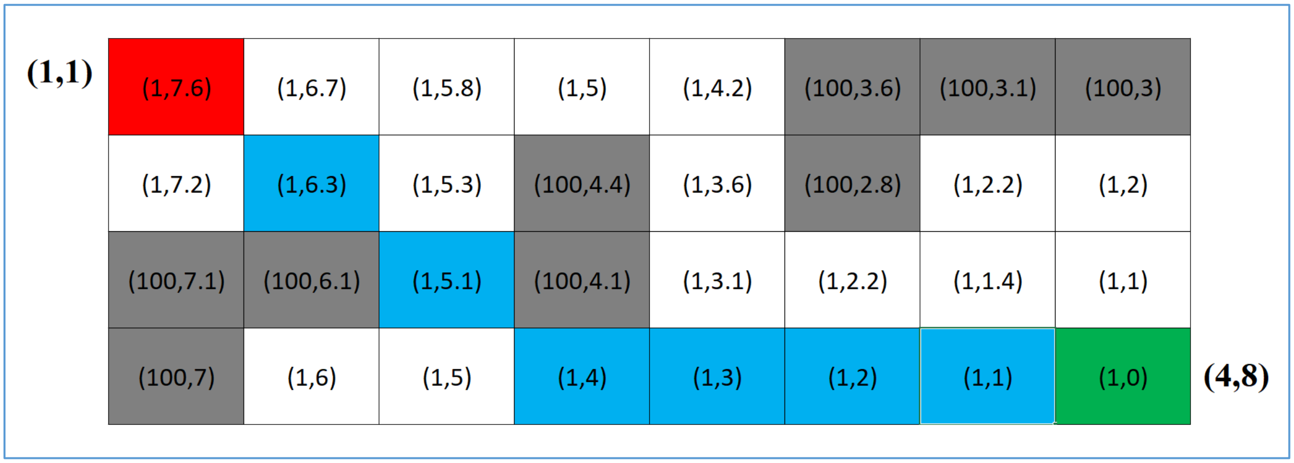
$$f(v) = h(v) + g(v) \tag{1.15}$$

Where $h(v)$ is a heuristic cost of the next picked cell, or simply it's the cost to move from that picked cell to the goal cell according to some cost function. While $g(v)$ is the cost of the path from the initial cell to the goal cell through the selected sequence of cells. For a further explanation, we can consider a 2D Grid map having several blocked cells (colored blacked) as shown in Figure 1.15, and unblocked cells (colored white). We start from a source cell (colored red) to reach towards a goal cell (colored green). Here, $h(v)$ is heuristic cost function (right value inside each cell) of the next picked cell to the goal cell. $g(v)$ (Left value inside each cell) is the length of the path from the initial cell to the goal cell through the selected sequence of cells. Each adjacent cells to the present cell is evaluated by the value $f(v)$. The cell with the lowest value of $f(v)$ is chosen as the next one in the sequence. The A* algorithm will chose the path as explained with the blue cells which represent the shortest path to reach the target cell.



(1,7.6)	(1,6.7)	(1,5.8)	(1,5)	(1,4.2)	(100,3.6)	(100,3.1)	(100,3)
(1,7.2)	(1,6.3)	(1,5.3)	(100,4.4)	(1,3.6)	(100,2.8)	(1,2.2)	(1,2)
(100,7.1)	(100,6.1)	(1,5.1)	(100,4.1)	(1,3.1)	(1,2.2)	(1,1.4)	(1,1)
(100,7)	(1,6)	(1,5)	(1,4)	(1,3)	(1,2)	(1,1)	(1,0)

Figure 1.15: 2D grid map with g cost and h cost allocated for each cell.



(1,7.6)	(1,6.7)	(1,5.8)	(1,5)	(1,4.2)	(100,3.6)	(100,3.1)	(100,3)
(1,7.2)	(1,6.3)	(1,5.3)	(100,4.4)	(1,3.6)	(100,2.8)	(1,2.2)	(1,2)
(100,7.1)	(100,6.1)	(1,5.1)	(100,4.1)	(1,3.1)	(1,2.2)	(1,1.4)	(1,1)
(100,7)	(1,6)	(1,5)	(1,4)	(1,3)	(1,2)	(1,1)	(1,0)

Figure 1.16: Blue cells represent the shortest path found by the A algorithm.*

1.3.4 Vision and Objects Detection

Computer vision is an interdisciplinary field that has gained tremendous traction in recent years. Robotics applications and self-driving cars have risen to prominence. Object detection is regarded as an essential component of computer vision.

Traditional object detection techniques adhere to the three major steps depicted in the diagram below. The first step is to generate a number of region proposals. These region proposals are candidates that may contain objects. Using various image descriptors, a fixed-length feature vector is extracted from each region proposal. Even if the vector varies due to a transformation, such as scale or translation, it should adequately describe an object. The feature vector is then used to assign each region proposal to one of the object classes or the background class. As the number of classes grows, so does the difficulty of creating a model that can distinguish between all of these objects.

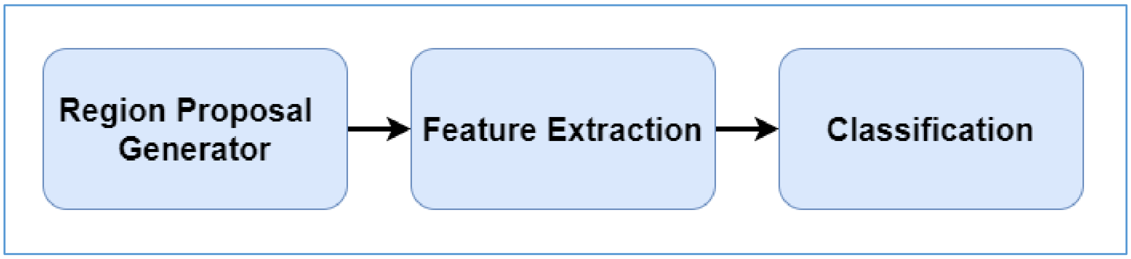


Figure 1.17: Traditional object detection steps.

For the same purpose, several algorithms are available, which are classified according to image classification processing levels: Figure 1.18 depicts two-stage and one-stage object detectors. Two-stage detectors have high accuracy but slow detection speeds, making them unsuitable for real-time detection. One-stage detectors, on the other hand, provide relatively high accuracy and detect objects in real time. This section will provide a brief overview of the most recent object detection algorithms.

R-CNN [23]: Region-based convolutional neural network R-CNN developed in 2014 that can detect 80 different types of objects in images. Compared to the generic pipeline of the object detection techniques shown in Figure 1.17, the main contribution of R-CNN is just extracting the features based on a convolutional neural network (CNN). Other than this, everything is similar to the generic object detection pipeline. Figure 1.19 shows the working of the R-CNN model.

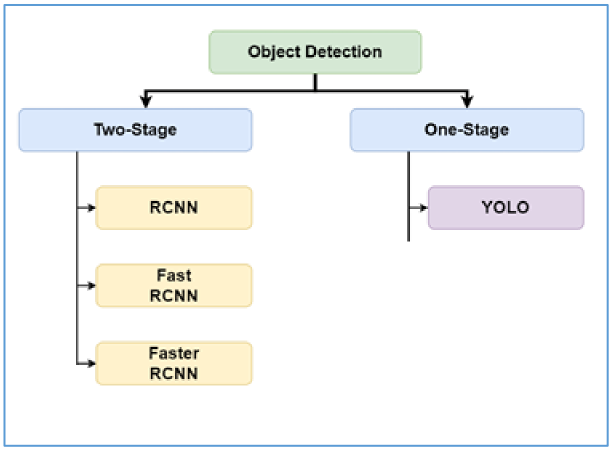


Figure 1.18: Examples of one and two stage detectors.

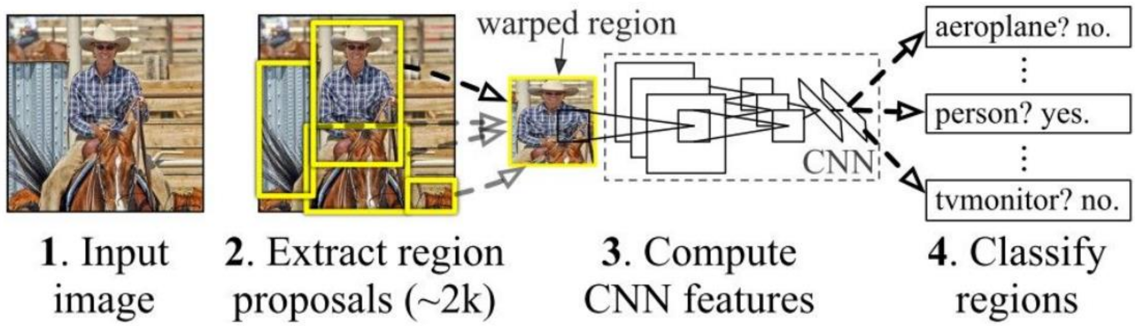


Figure 1.19: R-CNN object detection overview. [23]

The R-CNN consists of three main modules:

1. The first module generates 2,000 region proposals using the **Selective Search** algorithm.
2. After being resized to a fixed pre-defined size, the second module extracts a feature vector of length 4,096 from each region proposal.
3. The third module uses a pre-trained support vector machine (SVM) algorithm to classify the region proposal to either the background or one of the object classes.

The R-CNN model has some drawbacks:

- It is a multi-stage model, where each stage is an independent component. Thus, it cannot be trained end-to-end.
- It caches the extracted features from the pre-trained CNN on the disk to later train the SVMs. This requires hundreds of gigabytes of storage.
- R-CNN depends on the Selective Search algorithm for generating region proposals, which takes a lot of time. Moreover, this algorithm cannot be customized to the detection problem.
- Each region proposal is fed independently to the CNN for feature extraction. This makes it impossible to run R-CNN in real-time.

Fast R-CNN [24]: Fast R-CNN is an object detector that overcomes several issues in R-CNN. As its name suggests, one advantage of the Fast R-CNN over R-CNN is its speed.

The main contributions in fast R-CNN are:

1. Proposed a new layer that extracts equal-length feature vectors from all proposals in the same image.
2. Compared to R-CNN, which has multiple stages (region proposal generation, feature extraction, and classification using SVM), Faster R-CNN builds a network that has only a single stage.
3. Faster R-CNN shares computations (i.e. convolutional layer calculations) across all proposals rather than doing the calculations for each proposal independently, which makes Fast R-CNN faster than R-CNN.
4. Fast R-CNN does not cache the extracted features and thus does not need so much disk storage compared to R-CNN, which needs hundreds of gigabytes.
5. Fast R-CNN is more accurate than R-CNN.

Despite its advantages, the Fast R-CNN model has a significant disadvantage in that it relies on the time-consuming Selective Search algorithm to generate region proposals. The Selective Search method cannot be tailored to a particular object detection task. As a result, it may be insufficiently accurate to detect all target objects in the dataset.

Faster R-CNN [25]: Faster R-CNN is an extension of Fast R-CNN. As its name suggests, Faster R-CNN is faster than Fast R-CNN, this is due to the region proposal network (RPN).

The main contributions in this algorithm are:

1. Proposing **region proposal network (RPN)** which is a fully convolutional network that generates proposals with various scales and aspect ratios. The RPN implements the terminology of **neural network with attention** to tell the object detection (Fast R-CNN) where to look.

2. Rather than using pyramids of images (i.e. multiple instances of the image but at different scales) or pyramids of filters (i.e. multiple filters with different sizes), this algorithm introduced the concept of anchor boxes. An anchor box is a reference box of a specific scale and aspect ratio. With multiple reference anchor boxes, then multiple scales and aspect ratios exist for the single region. This can be thought of as a pyramid of reference anchor boxes. Each region is then mapped to each reference anchor box, and thus detecting objects at different scales and aspect ratios.
3. The convolutional computations are shared across the RPN and the Fast R-CNN. This reduces the computational time.

One disadvantage of Faster R-CNN is that the RPN is trained by extracting all anchors in the mini-batch of size 256 from a single image. Because all samples from a single image may be correlated (i.e. have similar features), the network may take a long time to reach convergence.

To localize the object within the image, all of the previous object detection algorithms used regions. The network does not examine the entire picture. Instead, parts of the image with a strong probability of containing the object are used. This resulted in the development of the YOLO, or "You Only Look Once," algorithm. YOLO is an object detection algorithm that is very different from the region-based algorithms discussed previously. In YOLO, a single convolutional network predicts the bounding boxes as well as their class probabilities.

YOLO Algorithm [26]: YOLO is a real-time object detection algorithm that employs neural networks. The phrase "You Only Look Once" is abbreviated as YOLO. Detecting objects requires only a single forward propagation through a neural network, as the name implies. It detects and recognizes different objects in a picture in real time. YOLO performs object detection as a regression problem and returns the class probabilities of the detected images. This allows YOLO to outperform other object detection algorithms by orders of magnitude. Figure 1.20 depicts an image that has been investigated using the YOLO algorithm.

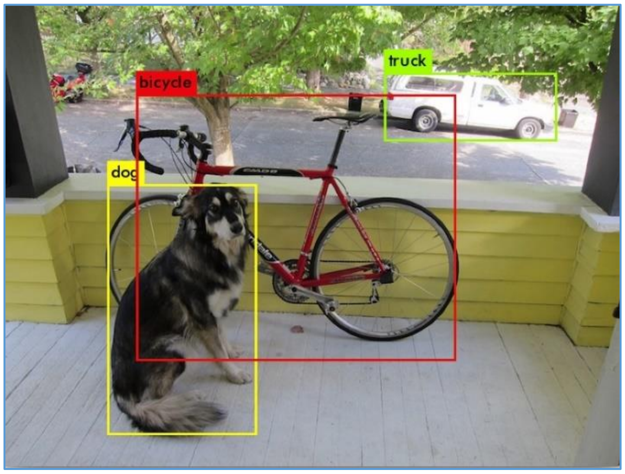


Figure 1.20: Output image from Yolo detector. [25]

Working Principle [26, 27]

YOLO algorithm works using the following three techniques:

- Residual blocks
- Bounding box regression
- Intersection over union

Residual blocks: The image is divided into various grids. Each grid has a dimension of S x S. The following figure shows how an input image is divided into grids. In the image below, there are many grid cells of equal dimension. Every grid cell will detect objects that appear within them.

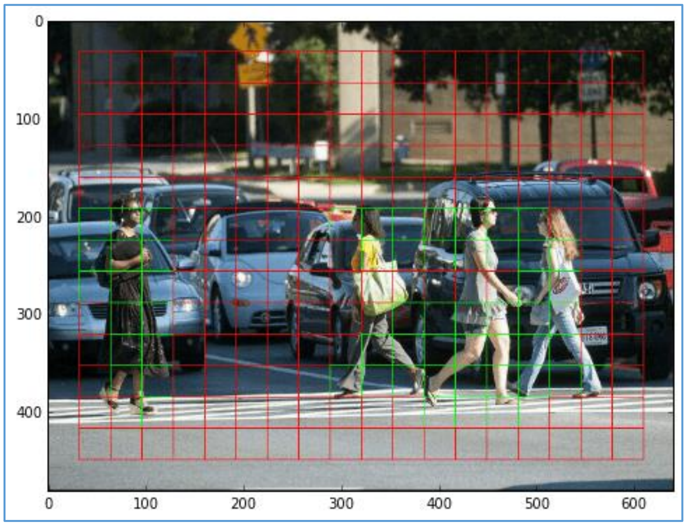


Figure 1.21: input image divided into S x S grid. [26]

Bounding box regression: A bounding box is an outline that highlights an object in an image. Every bounding box in the image consists of the following attributes:

1. Width b_w
2. Height b_h
3. Class (e.g. car, person or tennis ball) represented by letter c.
4. Bounding box center (b_x, b_y)

Figure 1.22 shows an example of a bounding box. The bounding box has been represented by a yellow outline.

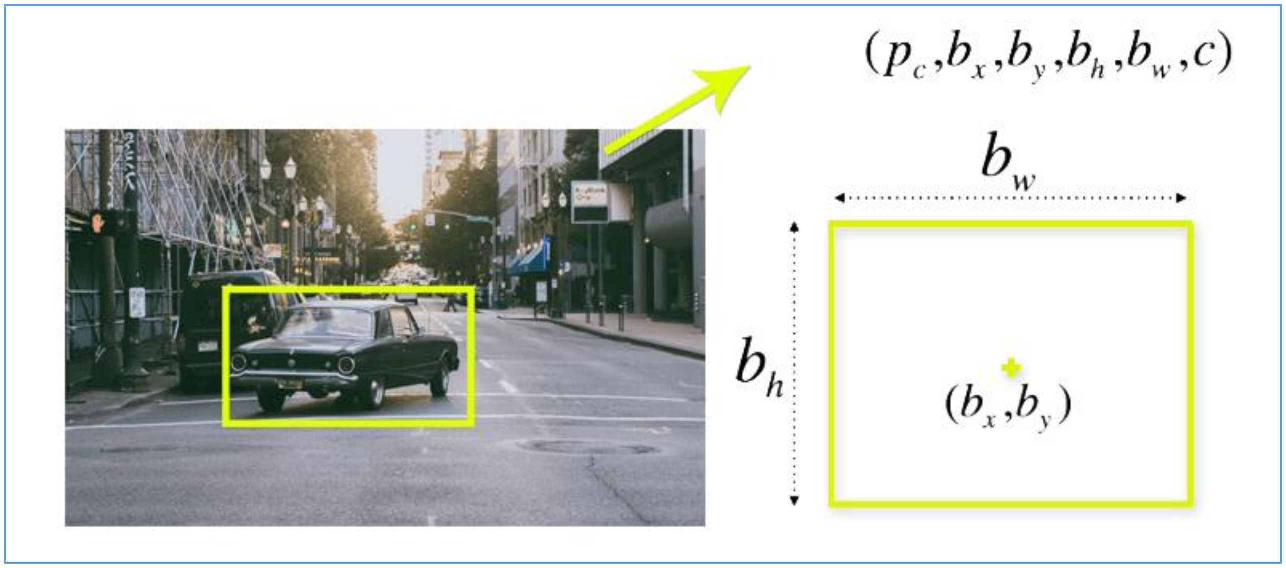


Figure 1.22: Predicted bounding box. [27]

YOLO uses a single bounding box regression to predict the height, width, center, and class of objects. In the image above, P_c represents the probability of an object appearing in the bounding box.

Intersection over union: IOU is a phenomenon in object detection that describes how boxes overlap. YOLO uses IOU to provide an output box that surrounds the objects perfectly. Each grid cell is responsible for predicting the bounding boxes and their confidence scores. The IOU is equal to 1 if the predicted bounding box is the same as the real box. This mechanism eliminates bounding boxes that are not equal to the real box.

The below image (Figure 1.23) illustrate how the three techniques are applied to produce the final detection results. First, the image is divided into grid cells. Each grid cell forecasts B bounding boxes and provides their confidence scores. The cells predict the class probabilities to establish the class of each object.

Intersection over union ensures that the predicted bounding boxes are equal to the real boxes of the objects. This phenomenon eliminates unnecessary bounding boxes that do not meet the characteristics of the objects (like height and width). The final detection will consist of unique bounding boxes that fit the objects perfectly.

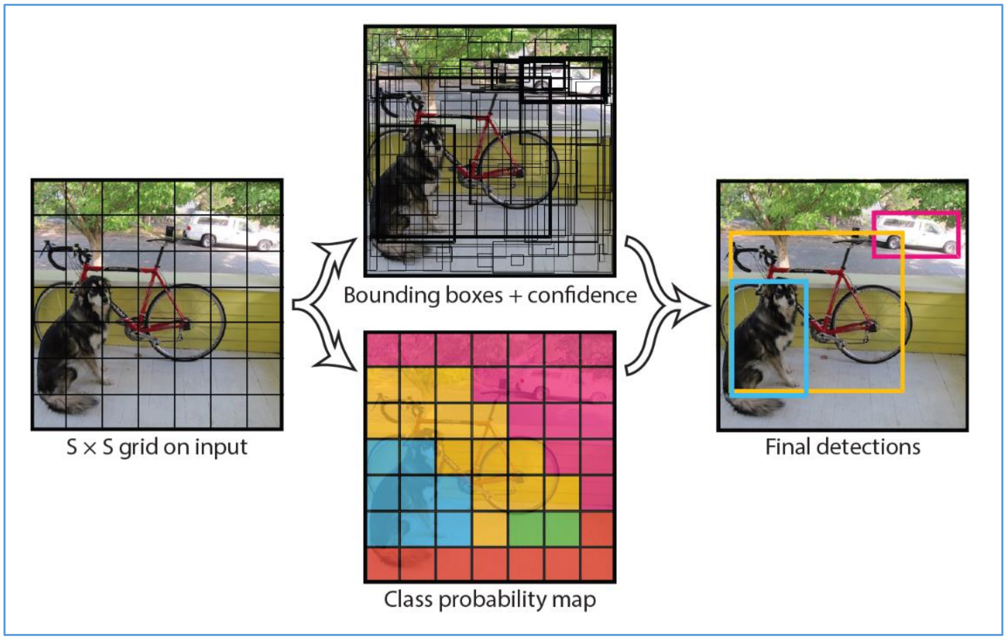


Figure 1.23: YOLO algorithm detection steps. [26]

Depth Image Sensor

Depth Sensor is a type of three-dimensional range finder that collects multi-point distance data across a wide field of view. It's a Time-Of-Flight sensor that projects a structured infrared light onto an object to improve the accuracy of the measured distance and then times the round trip from the sensor to the object and back. Standard digital cameras generate images as a 2D grid of pixels, with weights assigned to the colors Red, Green, and Blue (RGB). A depth camera, on the other hand, has pixels with different weights that represent the distance from the camera (depth). The depth camera has two sensors, spaced a small distance apart. Takes two images from these two sensors and compares them, since the distance between the sensors is known, these comparisons give depth information. Stereo cameras imitate the way how we use our eyes for depth perception. Figure 1.24 represents an image output from a depth camera. Each different color in the depth scale is translated to a distance from the camera. Despite the color scale, the depth data are provided by the sensor as a 2D-matrix which has the same size of the depth image. Each cell in the 2D-matrix contain a digital value which represent the depth of each pixel from the camera. A scaling factor is used to convert the given data to a metric distance.

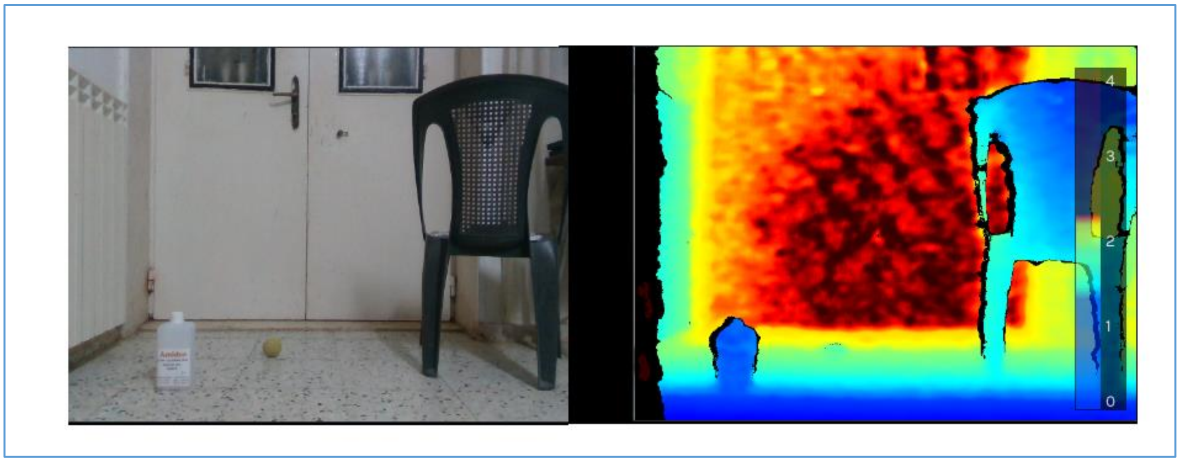


Figure 1.24: Depth Image (Right) in comparison with the original image (Left).

The red pixels represents the farthest distances from the depth sensor, while the blue pixels represents the shortest distances.

1.4 Thesis Organization

This thesis is organized into five logical chapters. The first chapter serves as an introduction and provides background information on ROS-based mobile robot. The tennis ball collecting mobile robot is covered in chapter two. The third chapter demonstrates the proposed system's design methodology, ROS design and implementation. The practical results are presented in Chapter four. The conclusion and future research topics are covered in chapter five.

2

Robot Design and Analysis

The tasks expected to be performed by the mobile robot is detailed in the previous chapters. In order for the robot to run successfully, a set of subtasks must be executed. Such as:

1. Object detection and classification
2. Motion planning with obstacle avoidance
3. Sub-Path planning
4. Motion control

The tennis robot mechanical structure is shown in Figure 2.1[28]. It's differential drive mobile robot that provides a high maneuverability and designed in such a way to meet and satisfy a certain requirements:

1. Reliable
2. Portable
3. Smart
4. User friendly and safe for the end user
5. Capable of collecting and storing up to 30 balls.

The collecting mechanism shown in Figure 2.1 (2) consists from a metal cylindrical rod covered by a cylindrical sponge. Which acts as a highly fuscous frictional surface. When such surface come in contact with the tennis ball, a large force pushes the tennis ball toward the sack (1). The rod is connected with a permanent magnet DC motor that controls its rotational speed. The value of the force is controlled by the speed of rotation. Whenever a tennis ball is detected, the robot steers its head toward it, such that the rotating sponge capture the ball and push it to the ball sack.

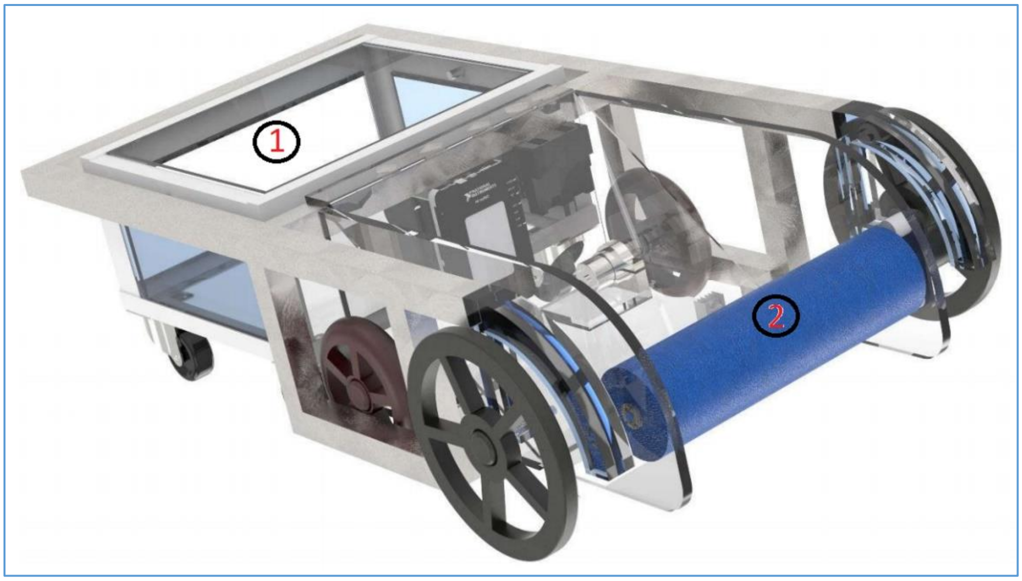


Figure 2.1: Tennis-Balls mobile robot. (1: Tennis ball sack, 2: collecting mechanism).[28]

The main block diagram of the system is illustrated in Figure 2.2. The system is composed of an image sensor (3D camera), a processing unit (SBC), and a low level control & driving unit (Motion sensors and actuators). Each component consists form sub units which are detailed in the bullet points below:

- **Image sensor:** It's a 3D camera (RGB-D) which output images in 2D in addition to a depth image. The 2D images consist from a grid of pixels and each pixel has value associated with it which represent the color value of that pixel. On the other hand, the depth image has pixels which have values that represent the distance from the camera.

- **Processing unit:** A SBC which responsible for executing all tasks algorithms.

- **Motor drivers:** An interface circuit between the processing unit and the motors circuits, used as a power amplifier and a low level controller. It reads the two encoders and use these reading for position or speed control according to the selected mode.

- **Motion sensors:** An encoder which convert motion to an electrical signals that can be read by the processing unit to determine the angular position and speed.

The Image sensor is used to capture 3D images of the robot environment. Then the 3D images is transferred to the minicomputer to be analyzed. Then, a motion planning is executed to drive the robot towards the target point within a specific time law. The main algorithm to be executed by the processing unit will be illustrated in chapter three.

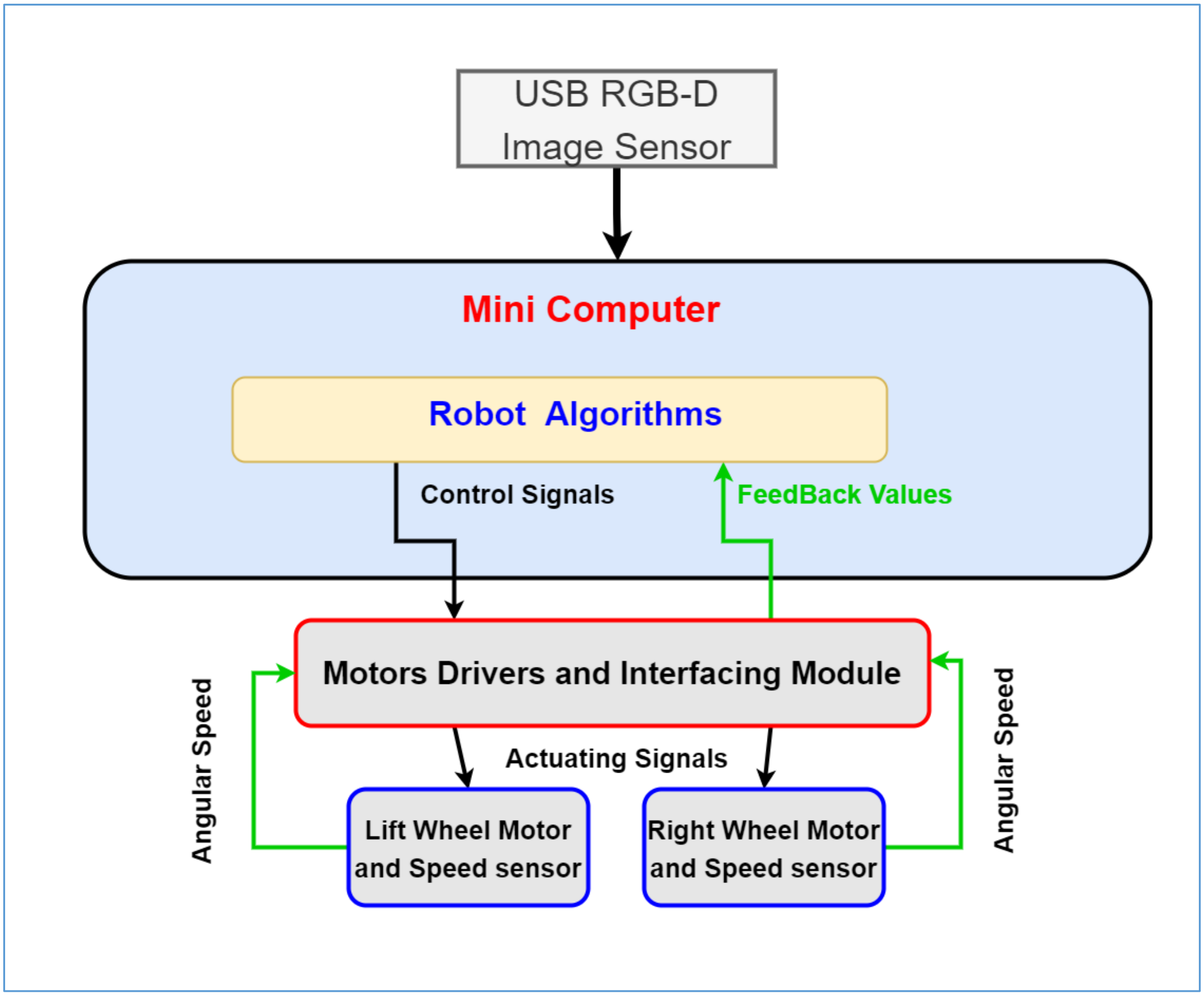


Figure 2.2: System block diagram.

2.1 Kinematics

A kinematic model describes the motion of a robot in mathematical form without considering the forces/ torques that affect the motion. And concerns itself with the geometric relationship between elements. [21]

The Tennis-Collecting mobile robot is a two wheeled differential drive robot (**TWD**), where each wheel is driven independently. A third wheel is a passive coaster wheel needed for the robot balance. To produce a forward motion, both wheels should be driven at the same speed. Evidently, turning right is achieved by driving the left wheel at a higher speed than the right wheel and vice versa for turning left. Moreover, differential drive make it possible for the robot to rotate on the spot , by driving one wheel forward and the second wheel in the opposite direction, both at same speed. Figure 2.3 illustrate the differential drive mobile robot geometry.

Deriving the model for TWD robot is an upward process, where each wheel contributes to the robot motion and at the same time impose constraints on the robot motion. The pose vector (position and orientation) and its speed are respectively represented by:

$$P = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \dot{P} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \qquad (2.1)$$

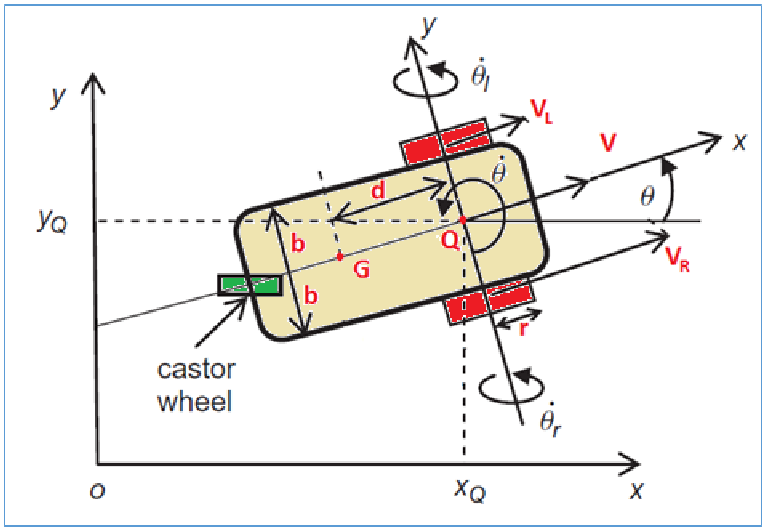


***Figure 2.3*: Geometry of differential drive robot.**

Differential drive mobile robot has the base kinematic model of a unicycle. Unicycle mobile robots are non-holonomic due to the constraints imposed by its mechanical structure [21]. i.e. the robot cannot move in a drift motion meaning to roll without any slipping. This leads that the robot can't drive strait to a goal that is not on the same line with its orientation. Thus, it must either rotate to the desired orientation before moving forward or rotate as it moves.

The no-slipping condition does not allow the generalized coordinates $\dot{x},\ \dot{y}$ & $\dot{\theta}$ to take arbitrary values. Accordingly, the generalized coordinates are constrained by the following equations:

$$\dot{x} = V\ \cos\theta\,,\ \dot{x} = V \sin\theta \qquad (2.2)$$

Where V is the linear velocity of the wheels mid-point Q. The angular position and speed of the right and left wheels are$\left(\theta_R, \dot{\theta}_R, \right)$ and $(\theta_L, \dot{\theta}_L)$.

Before deriving the Kinematics model, the following assumption are made:

- Wheels are rolling without slippage.

The left and right wheels linear velocities are V_L and V_R respectively, thus:

$$V_R = V + b\dot{\theta} \qquad (2.3)$$

$$V_L = V - b\dot{\theta} \qquad (2.4)$$

Adding and subtracting 2.3 and 2.4, the direct kinematic model for the TWD robot:

$$V = \frac{V_R + V_L}{2} = \frac{r(\dot{\theta}_R + \dot{\theta}_L)}{2} \qquad (2.5)$$

$$\dot{\theta} = \frac{V_R - V_L}{b} = \frac{r(\dot{\theta}_R - \dot{\theta}_L)}{b} \qquad (2.6)$$

And the component of linear velocities $\dot{x}$ and $\dot{y}$

$$\dot{x} = V \cos\theta\ \ and\ \ \dot{y} = V \sin\theta \qquad (2.7)$$

The forward kinematic model in matrix form:

$$\dot{P} = \begin{bmatrix} (r/2)\cos\theta & (r/2)\cos\theta \\ (r/2)\sin\theta & (r/2)\sin\theta \\ (r/b) & -(r/b) \end{bmatrix} \begin{bmatrix} \dot{\theta}_R \\ \dot{\theta}_L \end{bmatrix} \qquad (2.8)$$

Inverse kinematic model:

$$\dot{\theta}_R = \frac{2V + b\dot{\theta}}{2r} \qquad (2.9)$$

$$\dot{\theta}_L = \frac{2V - b\dot{\theta}}{2r} \qquad (2.10)$$

In matrix form:

$$\begin{bmatrix} \dot{\theta}_R \\ \dot{\theta}_L \end{bmatrix} = \begin{bmatrix} (1/r) & (b/2r) \\ (1/r) & -(b/2r) \end{bmatrix} \begin{bmatrix} V \\ \dot{\theta} \end{bmatrix} \qquad (2.11)$$

Where the V & $\dot{\theta}$ represents the control inputs that will steer the robot along a certain trajectory.

2.2 Dynamics

Dynamics is a branch of physical mechanics, defined as the process of deriving the equation of motion of material bodies with respect to their masses/moments of inertia and the forces/ torques that produce the motion [21]. This can be done using the following two methodologies:

- Newton-Euler method
- Lagrange method

Through the study, Lagrange method is utilized to derive the dynamic motion equations. The system model can be derived using Lagrange formula as follows: [21]

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \left(\frac{\partial L}{\partial q_i}\right) = Q_i - \left(\frac{\partial R}{\partial \dot{q}_i}\right) \tag{2.12}$$

$$R = \frac{1}{2}\,\overrightarrow{\dot{q}_\iota}^{T}\,[c]\,\overrightarrow{\dot{q}_\iota} \tag{2.13}$$

Where:

L: Stands for difference of kinetic energy T, and potential energy U.
q_i: Stands for the generalized coordinates.
Q_i: For the generalized force that acts on the mechanical system.
R: Represents Rayleigh dissipation function.
$[c]$: Is the damping matrix (Positive definite).

Because the mobile robot moves only on a plane surface, potential energy is zero ($U = 0$). And we have to find only the kinetic energy. The kinetic energy of the whole structure is given by the following equation:

$$T_{Tot} = T_{Tra} + T_{Rot} + T_{Whl} \tag{2.14}$$

Where:

T_{Tra}: The robot translation kinetic energy.
T_{Rot}: Kinetic energy of mobile robot rotation.
T_{Whl}: Kinetic energy of rotation of wheels and rotors of DC motors.

The aforementioned energy terms can be expressed by the following equations:

$$T_{Tra} = \frac{1}{2}mV_C^2 = \frac{1}{2}m(\dot{x}_C^2 + \dot{y}_C^2) \tag{2.15}$$

$$T_{Rot} = \frac{1}{2}I_Q\dot{\theta}^2 \tag{2.16}$$

$$T_{Whl} = \frac{1}{2} I_0 \dot{\theta}_R^2 + \frac{1}{2} I_0 \dot{\theta}_L^2 \quad (2.17)$$

Where:

m: The mass of the entire mobile robot(kg)
V_C: Linear velocity of the robot center of mass C (mm/s)
$\dot{X}_C, \dot{Y}_C$: Are velocities of point C in direction of x-axis and y-axis respectively (mm/s)
I_Q: Moment of inertia of entire mobile robot with respect to point Q $(kgmm^2)$
θ: Is the mobile robot orientation (rad)
$\dot{\theta}$: Mobile robot rotational speed (rad/s),
I_0: Moment of inertia of combined drive motor (rotor) and wheel $(kgmm^2)$

Components of the velocity at point Q can be expressed in terms of $\dot{\theta}_R$ and $\dot{\theta}_L$:

$$\dot{X}_A = \frac{r}{2}\left(\dot{\theta}_R + \dot{\theta}_L\right) cos\,\theta \quad (2.18)$$

$$\dot{Y}_A = \frac{r}{2}\left(\dot{\theta}_R + \dot{\theta}_L\right) sin\,\theta \quad (2.19)$$

$$\dot{\theta} = \frac{r(\dot{\theta}_R - \dot{\theta}_L)}{b} \quad (2.20)$$

Where:
$\dot{X}_A$: Presents velocity at point Q in the direction of X-axis (mm/s),
$\dot{Y}_A$: Presents the velocity at point Q in the direction of the Y-axis (mm/s).

Now the components of the velocity at point C referring to point Q are now:

$$\dot{X}_C = \dot{X}_A + d\dot{\theta}\, sin\,\theta \quad (2.21)$$

$$\dot{Y}_C = \dot{Y}_A - d\dot{\theta}\, cos\,\theta \quad (2.22)$$

Total kinetic energy of the mobile robot can be calculated in terms of $\theta_R\ and\ \theta_L$, likewise:

$$T_{Tot} = \dot{\theta}_R^2\left(\frac{mr^2}{8} + \frac{(d^2r^2m)}{2b^2} + \frac{r^2 I_Q}{2b^2} + \frac{1}{2} I_0\right) + \quad \dot{\theta}_L^2\left(\frac{mr^2}{8} + \frac{(d^2r^2m)}{2b^2} + \frac{r^2 I_Q}{2b^2} + \frac{1}{2} I_0\right) + \dot{\theta}_R\dot{\theta}_L\left(\frac{r^2}{4} - \frac{md^2r^2}{b^2} - \frac{r^2 I_Q}{b^2}\right) \quad (2.23)$$

$$Let\ A = \left(\frac{mr^2}{8} + \frac{(d^2r^2m)}{2b^2} + \frac{r^2I_Q}{2b^2} + \frac{1}{2}I_0\right) \tag{2.24}$$

$$And \qquad B = \left(\frac{r^2}{4} - \frac{md^2r^2}{b^2} - \frac{r^2I_Q}{b^2}\right) \tag{2.25}$$

The system has viscous damping, its motion will be resisted by a force whose magnitude is proportional to that of the velocity but in the opposite direction. Now, applying Lagrange equations:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_R}\right) - \left(\frac{\partial L}{\partial \theta_R}\right) = M_R - K\dot{\theta}_R \tag{2.26}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_L}\right) - \left(\frac{\partial L}{\partial \theta_L}\right) = M_L - K\dot{\theta}_L \tag{2.27}$$

Where:

M_R, M_L: are right and left actuating torques, respectively in $(kgmm^2/s^2)$.
$K\dot{\theta}_R$, $K\dot{\theta}_L$: viscous friction values of right and left wheel motor systems, in $(kgmm/s^2)$

Finally, the obtained dynamic motion equation can be expressed as:

$$2A\,\ddot{\theta}_R + B\ddot{\theta}_L = M_R - K\dot{\theta}_R \tag{2.28}$$

$$B\,\ddot{\theta}_R + 2A\ddot{\theta}_L = M_L - K\dot{\theta}_L \tag{2.29}$$

$$\begin{bmatrix} 2A & B \\ B & 2A \end{bmatrix}\begin{bmatrix} \ddot{\theta}_R \\ \ddot{\theta}_L \end{bmatrix} = \begin{bmatrix} M_R - K\dot{\theta}_R \\ M_L - K\dot{\theta}_L \end{bmatrix} \tag{2.30}$$

Due to the presence of a low level control loops for each wheel that are integrated in the hardware or software architecture, it is not possible to command directly the wheel torques [21]. Thus the dynamic model obtained in (2.30) will be used in the case of calculating the maximum required torque to move the robot under full load condition. And presented here for the sake of completeness.

2.3 Robot Sensors and Actuators

Two basic components of the tennis robot are discussed in this section. Actuators which represent the main actuating system, responsible of actuating the wheels of the tennis robot. Successively, the robot sensors which classified as proprioceptive sensors and exteroceptive sensors, the former used to measure quantities that characterize the internal states of the robot. The later used to acquire information from the robot environment.

Actuators

Based on the mechanical structure of the tennis mobile robot, the wheels requires two main actuators, one for each wheel in order to perform any imposed admissible motion. Each actuator consists of:

- Power Amplifier
- Servomotor
- Transmission:

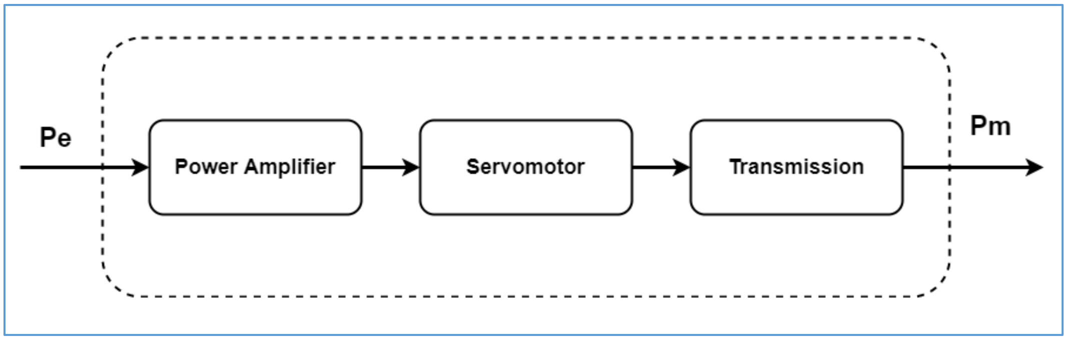


Figure 2.4: Components of joint actuating system.

Power amplifier takes a fraction of the power available at the main source (Power Supply) which is proportional to the control signal (P_e) and transmit it to the servomotor in terms of voltage or current. Servomotor which is a permanent magnet DC motor, responsible of converting the electrical power to mechanical power. Due to the fact that the tennis robot requires a low speed with high torque motion, and the servomotor provides high speed with low torque. It is necessary to impose a transmission gear to optimize the transfer of mechanical power from the motor to the wheels (P_m). The servomotor simplified transfer function can be described by the following equation [29]:

$$\frac{\dot{\theta}_m(s)}{E_a(s)} = \frac{K_t/(R_aJ_m)}{s\left[s+\frac{1}{J_m}\left(D_m+\frac{K_tK_b}{(R_a)}\right)\right]} \tag{2.31}$$

$$J_m = J_a + J_L(\frac{N_1}{N_2})^2 \tag{2.32}$$

$$D_m = D_a + D_L(\frac{N_1}{N_2})^2 \tag{2.33}$$

Where:

$\dot{\theta}_m$: Servomotor output angular velocity.

E_a: Applied armature voltage on the servomotors terminals.

K_t: Servomotor torque constant.

K_b: Back electromotive force constant.

J_m: Equivalent inertia at the armature (includes both the armature inertia J_a and the load inertia J_L reflected to the armature).

D_m: Equivalent viscous damping at the armature (includes both the armature viscous damping D_a and the load viscous damping D_L reflected to the armature).

R_a: Servomotor armature resistance.

N_1: Number of teeth's of input drive gear.

N_2: Number of teeth's of output drive gear.

Sensors

The sensor network elements are chosen based on the tasks performed by the robot, such as:

- Object detection and classification (Tennis balls and obstacles).
- Motion planning and navigations.

These tasks require specialized sensors in order to acquire the necessary data. In this work two types of sensors are used to gather data, which are:

- Proprioceptive sensors: An incremental encoder used to measure the internal states of the robot, such as angular position and angular speed. Two encoders are used, whereas each encoder is mounted on the robot wheel.

- Exteroceptive sensor: A 3D image sensor containing two camera sets is used to provide the robot with the information needed to execute intelligent actions in an autonomous way. A 2D RGB camera, which generates a 2D colored images that could be used for objects detections and classifications. And a depth camera which provides a structural data given by the distance of the detected object and the corresponding measurement direction. Eventually, the data provided by the camera are used by the robot to avoid obstacles, build maps of the work space environment and recognize objects.

3

Robot Vision and Motion

The robot requirements that the tennis robot should provide in order to perform the desired tasks are detailed in the previous chapters. These requirements include both hardware and software. In this chapter, a detailed design methodology for accomplishing each task will be presented.

In order for the system to collect tennis balls, a miniature series of tasks should be executed. Such tasks are, command image sensor used to capture 3D images of the robot environment. Then the 3D images is transferred to the SBC to be analyzed. The algorithms within the processing unit provides an objects detection and classification. A motion planning and servoing is executed to generate the optimal paths toward the target point (Tennis Ball). And accordingly, a trajectory is generated to drive the robot towards the target point within a specific time law. After that, an inverse kinematic model will be used to obtain velocity vector in joint space. Finally, a control signals for the robot motors will be generated and applied to the robot motors with a negative feedback system which ensure the output requirements are met. Figure 3.1 demonstrate the main algorithm to be executed by the processing unit.

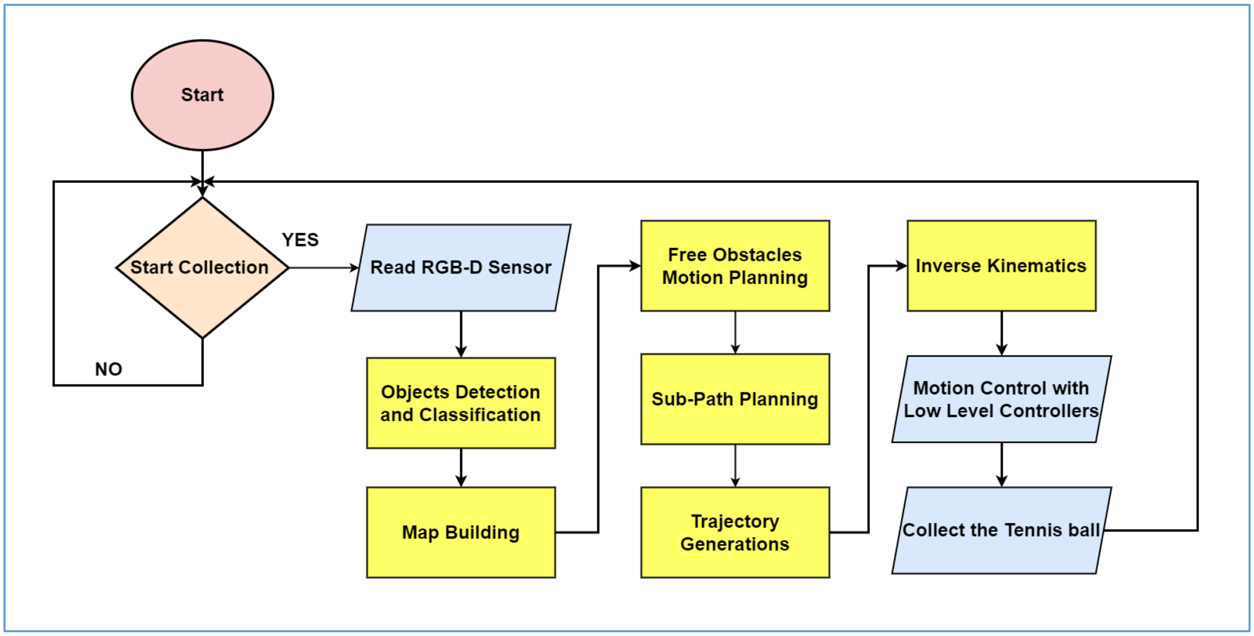


Figure 3.1: Tennis robot main algorithm.

The detailed block diagram of the system is illustrated in Figure 3.2. As shown in the figure, the system is composed of a sensor (3D camera), a processing unit (Single Board Computer), and a driving unit (Motion sensors and actuators). The main algorithm will be implemented within the processing unit under the ROS environment as will be illustrated in the following sections.

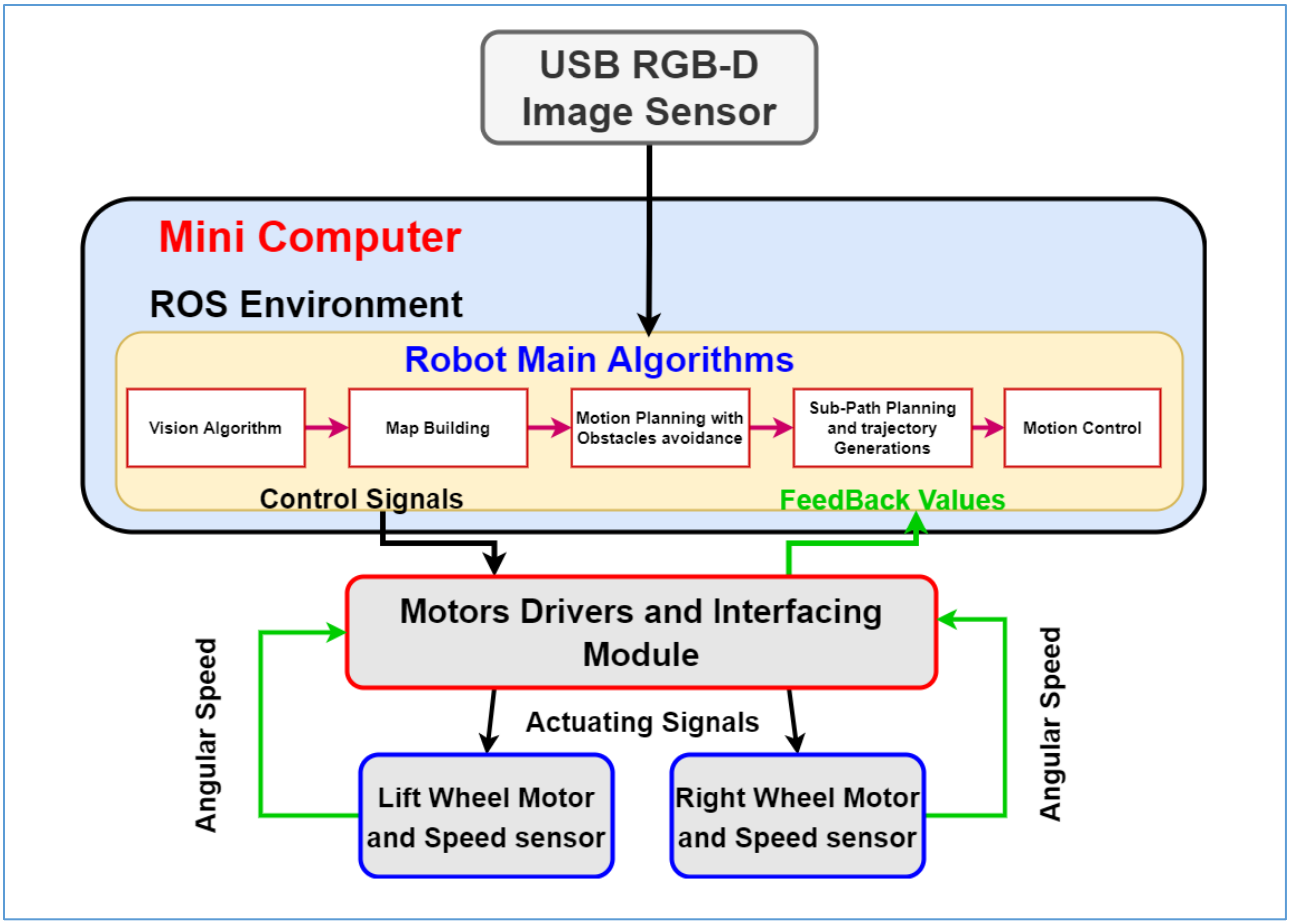

USB RGB-D Image Sensor
Mini Computer
ROS Environment
Robot Main Algorithms
Vision Algorithm
Map Building
Motion Planning with Obstacles avoidance
Sub-Path Planning and trajectory Generations
Motion Control
Control Signals
FeedBack Values
Motors Drivers and Interfacing Module
Angular Speed
Actuating Signals
Angular Speed
Lift Wheel Motor and Speed sensor
Right Wheel Motor and Speed sensor

Figure 3.2: Detailed block diagram.

3.1 Tennis Mobile Robot Vision

After the exteroceptive sensor (image sensor) provides the processing unit of the tennis robot with the needed data. A vision algorithm will be executed in order to perform the tasks related to object detection and classifications. The hardware capabilities of the processing unit impose some limitations on the selected algorithm in terms of accuracy and speed. It's known that the output accuracy is proportional to the execution time meaning using the same hardware with higher accuracy introduce a large latency. Thus, a more hardware resources is required for a higher accuracy. Therefore, for the tennis robot, there is a tradeoff between accuracy and speed that must by optimized. Such that the algorithm must be capable of detecting the smallest object in the workspace (Tennis ball).

Among the reviewed detectors which are detailed in chapter one, YOLO algorithm will be used for the following criteria [26]:

- Speed: Improves the speed of detection.
- Accuracy: Provides relatively high accuracy with minimal errors.
- Learning capabilities: It has an excellent learning capability that enable it to learn representations of an object and apply them in object detection.

The following algorithm illustrate the steps for detecting objects within a captured image.

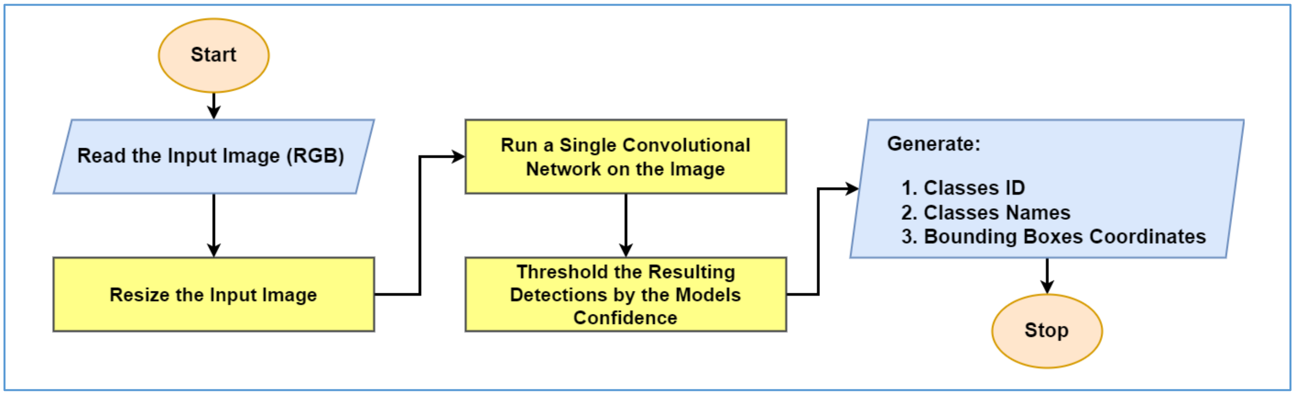

Start
Read the Input Image (RGB)
Resize the Input Image
Run a Single Convolutional Network on the Image
Threshold the Resulting Detections by the Models Confidence
Generate:
1. Classes ID
2. Classes Names
3. Bounding Boxes Coordinates
Stop

Figure 3.3: YOLO algorithm.

After the image is analyzed, each object in the output image is bounded by a bounding box as illustrated previously in Figure 1.22. Each bounding box is given a detailed information which listed in the pullet points below:

- Class Name
- Class ID
- Matching Probability
- Bounding Box Dimensions (in pixels):

$$X_{min} \text{ \& } X_{max}$$
$$Y_{min} \text{ \& } Y_{max}$$

From Bounding Box Dimensions, we can calculate the coordinated of the center of each object, such that:

$$X = X_{max} - X_{min} \tag{3.1}$$

$$Y = Y_{max} - Y_{min} \tag{3.2}$$

Depth Image Sensor

A stereo depth image sensor is mounted on the robot in order to measure the distance for each detected object. It capable of measuring the objects distance both indoor and outdoor in a wide variety of lighting conditions. The depth data are provided by the sensor as a 2D-matrix (size of 640 x 480) which has the same size of the RGB image. Each cell in the 2D-matrix contain a digital value which represent the depth of each pixel from the camera. The depth data can be utilized after extracting the bounding box coordinates from the YOLO algorithm. Thus, the detected object coordinates can be calculated using equations (3.1 & 3.2), and these coordinates will be utilized to access the depth data in the 2D-matrix in order to construct the workspace map. The depth information is scaled to a metric distance using a scaling factor which given in the depth sensor datasheet.

3.2 Motion Planning

Knowing the location of the robot and coordinates of its surrounding is an essential process for autonomous navigation system. Which is required to transfer the robot from an initial posture to a final posture, in order to execute the assigned task without colliding with the obstacles. Very often, besides obstacles avoidance, the robot must also satisfy some other requirements or optimize certain performance criteria such as searching for the shortest admissible path. The navigation problem can be divided into two sub-problems.

1. Workspace map building
2. Path-Finding using search algorithms

Workspace Mapping

Creating a live map will help the tennis mobile robot to perceive its environment which contains all the objects locations that are detected by the vision system. A two-dimensional map will be defined with x represents the rows and y represents the columns. For every coordinate in the map, there is a cell with a specific dimension. The cell dimension is chosen according to the length of the robot collecting mechanism, which is 30 cm. Thus, the cell dimension is $(30cm\ X\ 30cm)$, which can represents a tennis ball (red cells), a player, a static obstacle (black cells) or an empty cell. All the cells have the same dimensions that capable of containing a workspace object. The empty cells can be grouped together to form an optimized feasible obstacles-free path. Each detected object will occupy a cell or certain cells based on its dimensions. The depth image sensor can measure distances up to 9 meters long, and 2.1 meter width; thus, the workspace environment is partially known. Furthermore, due to the fact that the depth sensor cannot know what objects are laying beyond obstacles, the map building algorithm will consider all the cells behind the obstacle cells an obstacles cells (blue cells). Figure 3.4 shows an example of work space map with obstacles, start position (green cell) and end position. Furthermore, the robot route must be far by one cell from the obstacles as a guard distance (orange cells). Thus, all the neighboring cells around the obstacles will be given a similar high local cost value. The real scene is represented by Figure 3.5. It should be noted that for the simplicity, the figure represents only a section of the workspace. The area inside the blue line represents the observed sighted region of the robot vision sensors, it represents the angle of view of the camera sensors. The uncharted cells around the robot (Yellow cells) are given a cost value similar to the cells that contains obstacles. This makes the algorithm avoid uncharted cells.

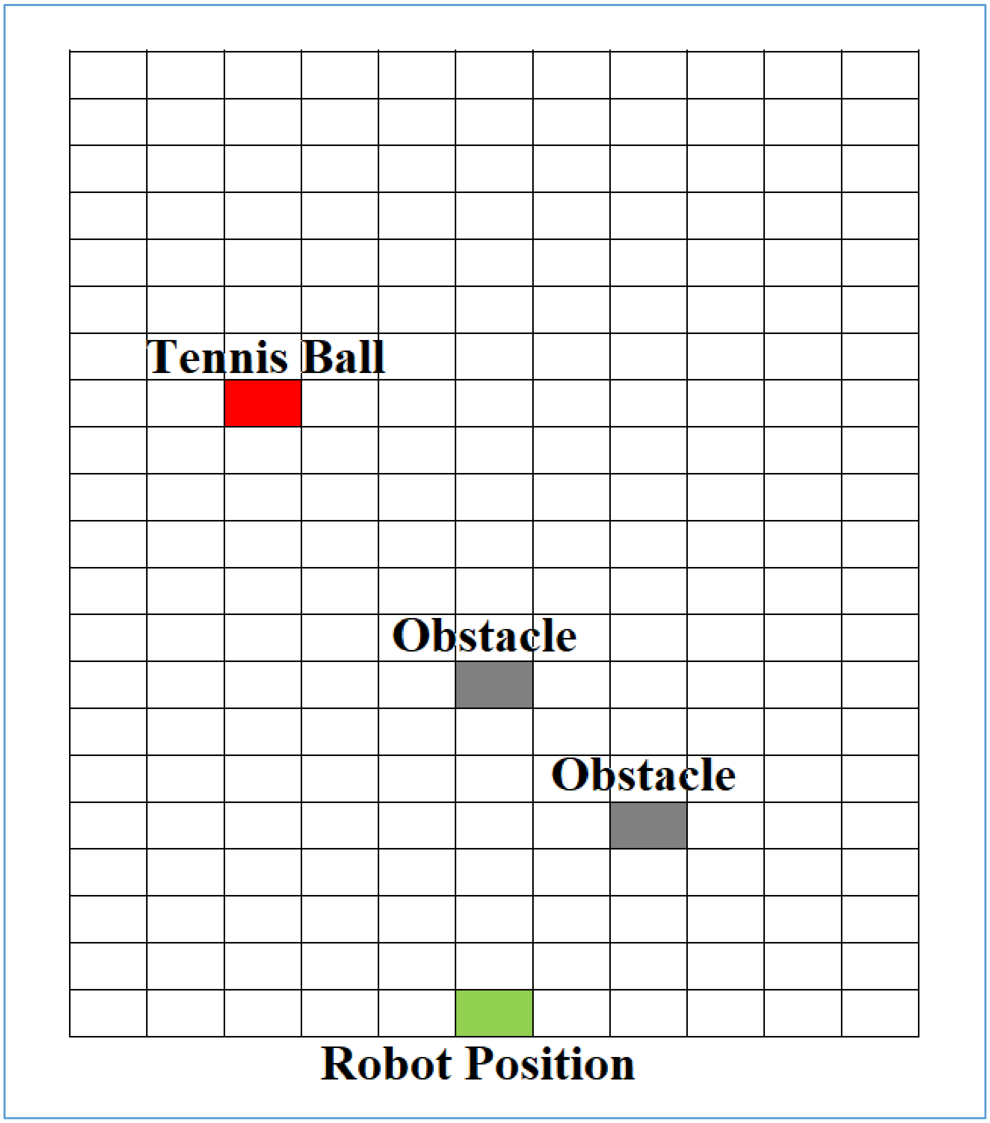


Figure 3.4: Robot Workspace.

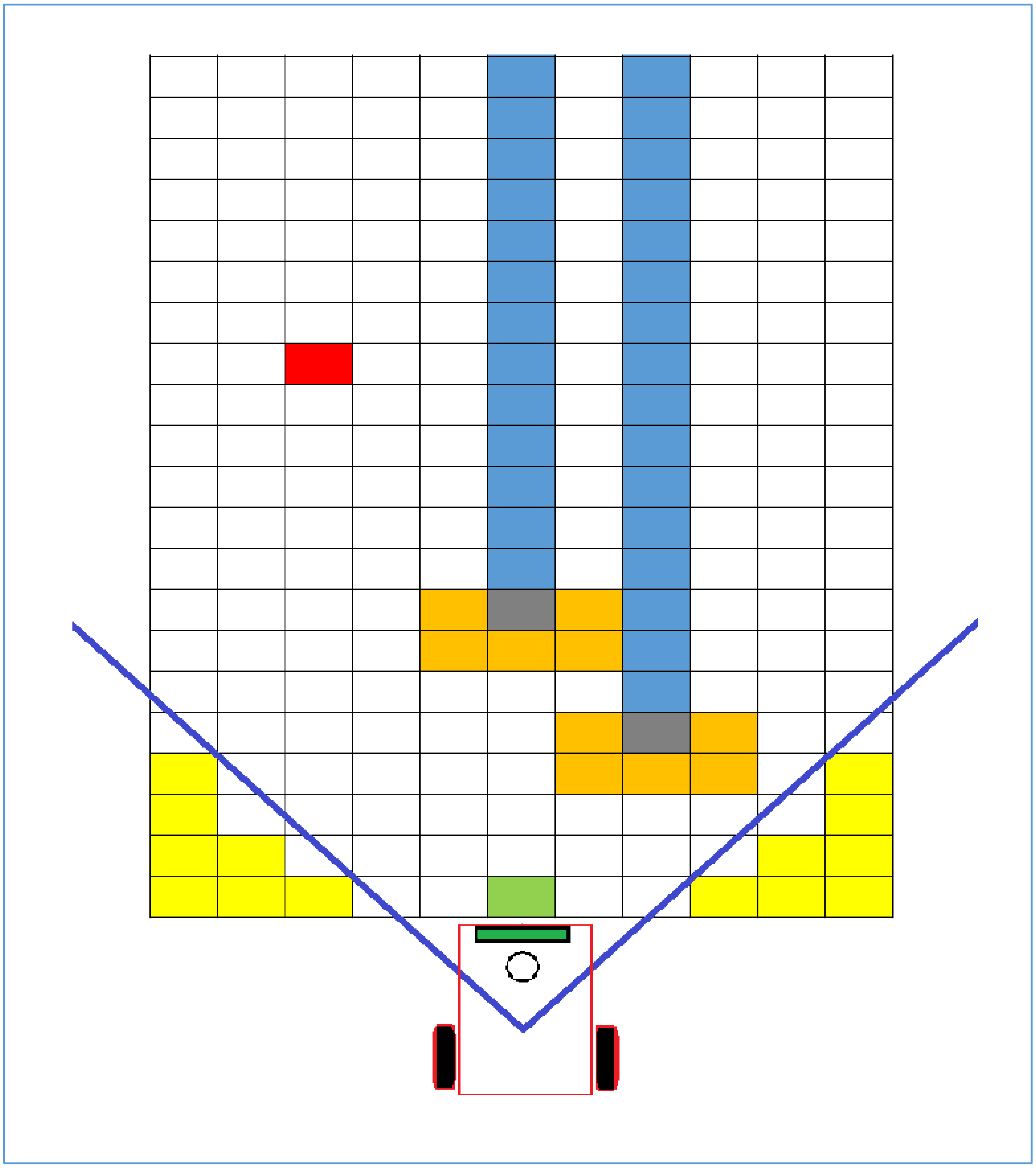


Figure 3.5: Robot field of view.

Path-Finding using A-Star (A*) Search Algorithm

A* algorithm is one of the best-known path planning algorithms, which can be applied on the tennis robot task space. As detailed in chapter 1, it uses a combination of heuristic searching and searching based on the shortest path. Because each cell in the task space is evaluated by the value of $f(v)$ in equation (1.15). The cell with the lowest value of $f(v)$ is chosen as the next one in the sequence.

For the A* algorithm to work properly, a cost map should be built based on the gathered data. For each position in the map there will be a local cost and a distance cost. The local cost is calculated based on the difficulty of commuting through a cell. One way to make the robot avoid obstacles is to give cells occupied by an obstacle a high local cost. i.e. in a scale from 1 to 1000 give an occupied cell (1000). And the empty cells will be given the lowest cost value (1). Due to the robot mechanical structure (Differential drive), it capable of moving in any angle in the map. Thus, the distance cost can be simply obtained using Euclidian distance formula (d) from each cell to the goal cell. Figure 3.6 illustrate the algorithm that generate the local cost matrix.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{3.3}$$

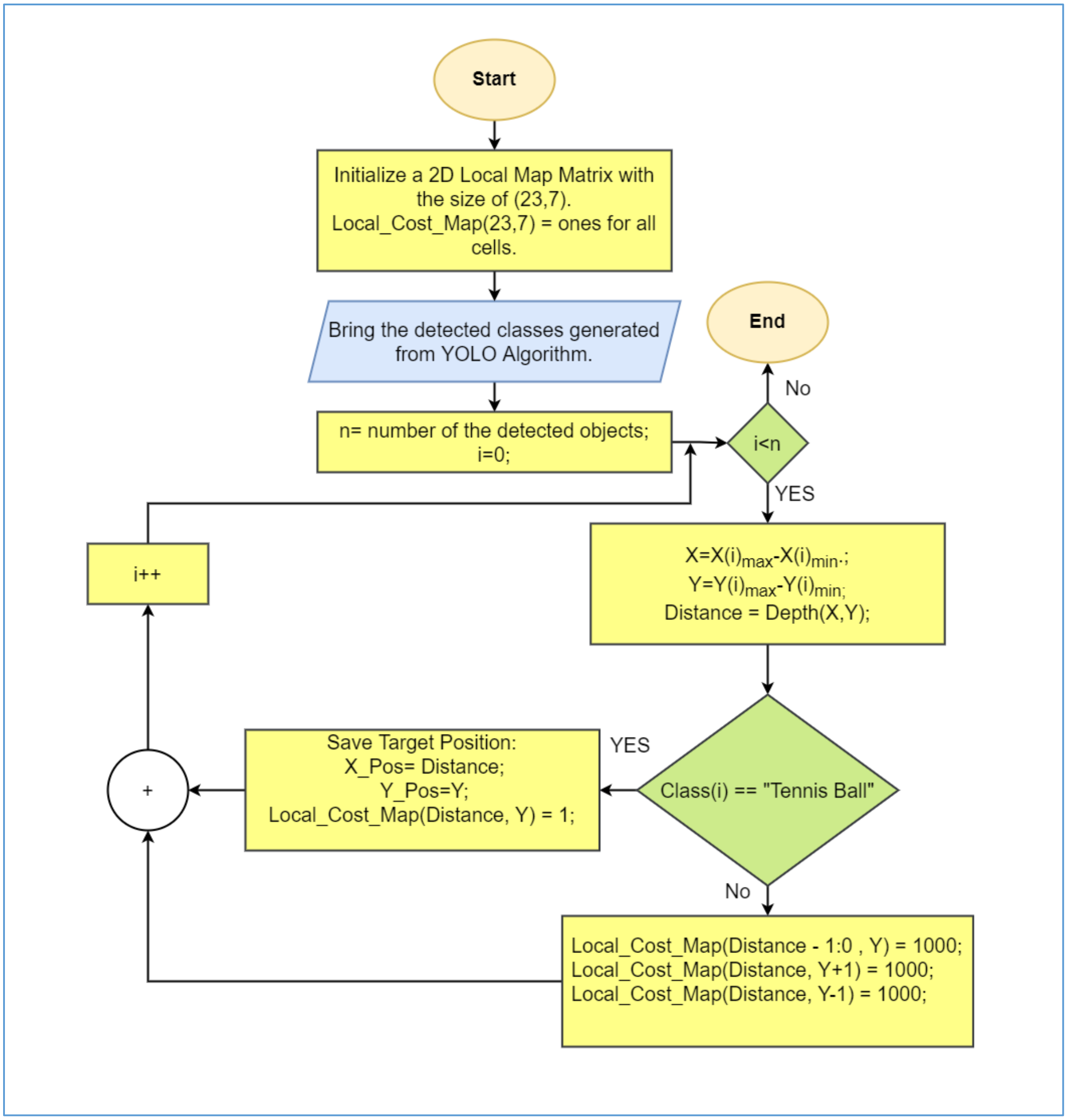
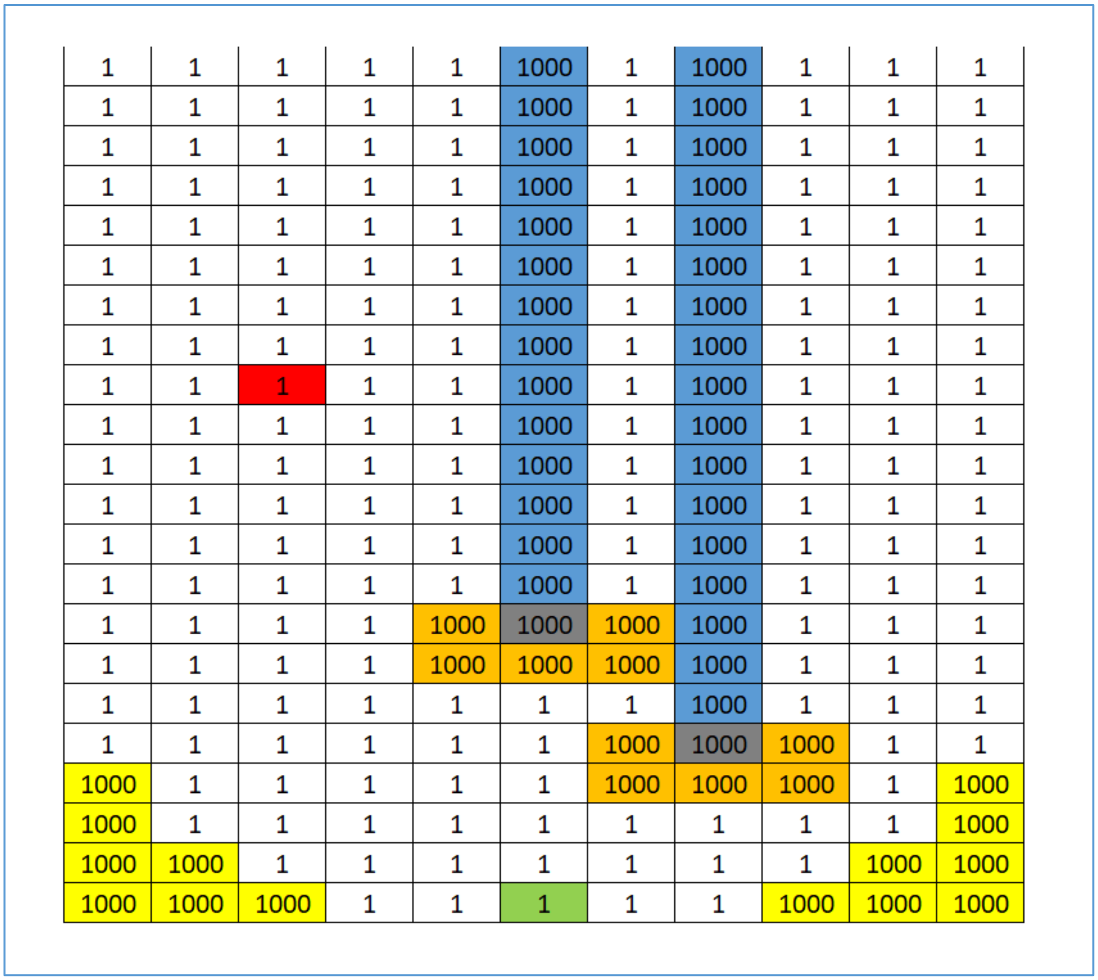

Start
Initialize a 2D Local Map Matrix with the size of (23,7).
Local_Cost_Map(23,7) = ones for all cells.
Bring the detected classes generated from YOLO Algorithm.
n= number of the detected objects;
i=0;
End
No
i<n
YES
i++
$X=X(i)_{max}-X(i)_{min\cdot};$
$Y=Y(i)_{max}-Y(i)_{min;}$
Distance = Depth(X,Y);
Save Target Position:
X_Pos= Distance;
Y_Pos=Y;
Local_Cost_Map(Distance, Y) = 1;
YES
+
Class(i) == "Tennis Ball"
No
Local_Cost_Map(Distance - 1:0 , Y) = 1000;
Local_Cost_Map(Distance, Y+1) = 1000;
Local_Cost_Map(Distance, Y-1) = 1000;

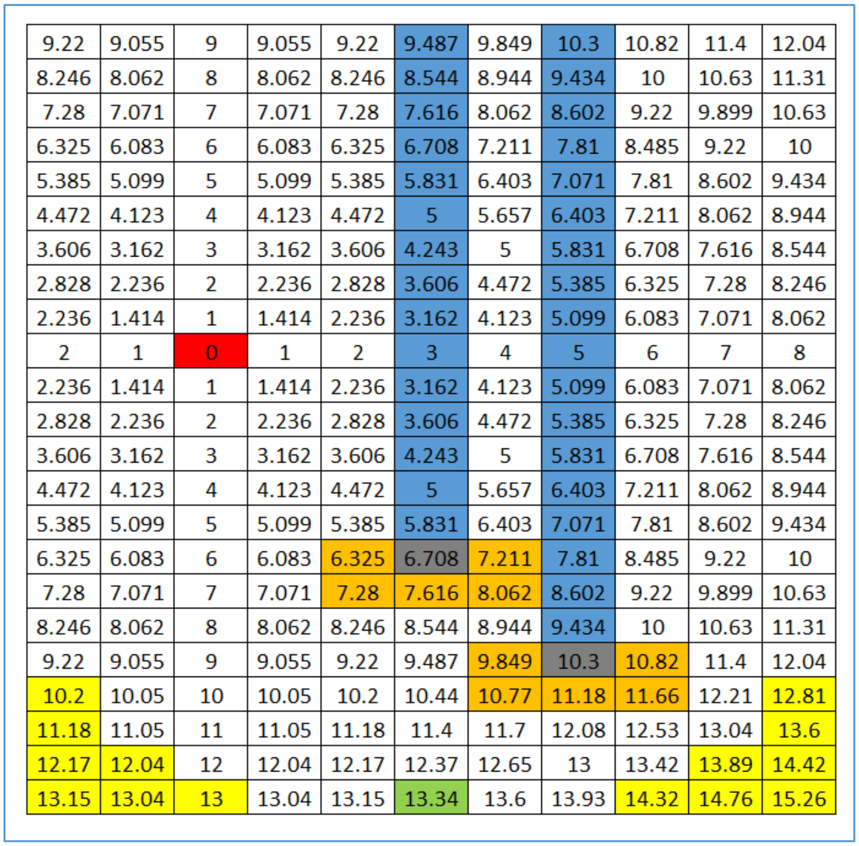
Figure 3.6: Local cost matrix generation Algorithm.

For the scene given in Figure 3.5, the local cost matrix is illustrated in Figure 3.7. And the distance cost matrix is illustrated in Figure 3.8. The local cost matrix represents the value of $g(v)$, while the distance cost matrix represents the value of $h(v)$. Since all the value are obtained, the value of $f(v)$ for each cell will be obtained as shown in Figure 3.9. And accordingly, the next cell in the path will be selected as illustrated in Figure 3.12. The algorithm is illustrated in Figure 3.11.



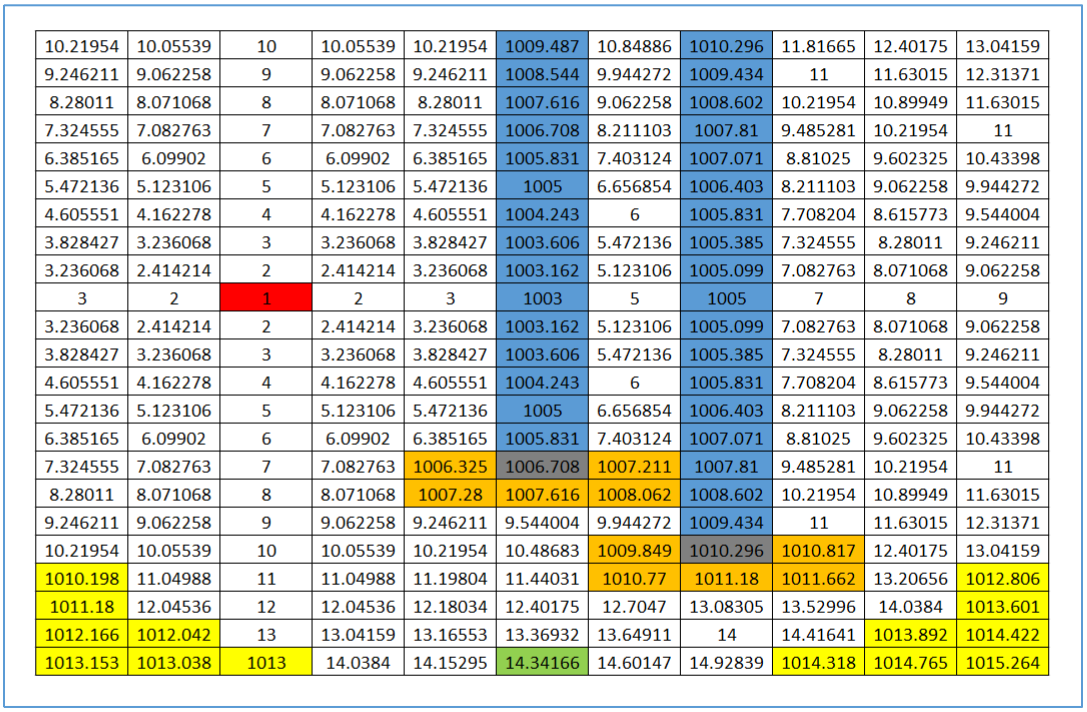
1	1	1	1	1	1000	1	1000	1	1	1
1	1	1	1	1	1000	1	1000	1	1	1
1	1	1	1	1	1000	1	1000	1	1	1
1	1	1	1	1	1000	1	1000	1	1	1
1	1	1	1	1	1000	1	1000	1	1	1
1	1	1	1	1	1000	1	1000	1	1	1
1	1	1	1	1	1000	1	1000	1	1	1
1	1	1	1	1	1000	1	1000	1	1	1
1	1	1	1	1	1000	1	1000	1	1	1
1	1	1	1	1	1000	1	1000	1	1	1
1	1	1	1	1	1000	1	1000	1	1	1
1	1	1	1	1	1000	1	1000	1	1	1
1	1	1	1	1	1000	1	1000	1	1	1
1	1	1	1	1	1000	1	1000	1	1	1
1	1	1	1	1000	1000	1000	1000	1	1	1
1	1	1	1	1000	1000	1000	1000	1	1	1
1	1	1	1	1	1	1	1000	1	1	1
1	1	1	1	1	1	1000	1000	1000	1	1
1000	1	1	1	1	1	1000	1000	1000	1	1000
1000	1	1	1	1	1	1	1	1	1	1000
1000	1000	1	1	1	1	1	1	1	1000	1000
1000	1000	1000	1	1	1	1	1	1000	1000	1000

Figure 3.7: Local cost matrix of the workspace map.



9.22	9.055	9	9.055	9.22	9.487	9.849	10.3	10.82	11.4	12.04
8.246	8.062	8	8.062	8.246	8.544	8.944	9.434	10	10.63	11.31
7.28	7.071	7	7.071	7.28	7.616	8.062	8.602	9.22	9.899	10.63
6.325	6.083	6	6.083	6.325	6.708	7.211	7.81	8.485	9.22	10
5.385	5.099	5	5.099	5.385	5.831	6.403	7.071	7.81	8.602	9.434
4.472	4.123	4	4.123	4.472	5	5.657	6.403	7.211	8.062	8.944
3.606	3.162	3	3.162	3.606	4.243	5	5.831	6.708	7.616	8.544
2.828	2.236	2	2.236	2.828	3.606	4.472	5.385	6.325	7.28	8.246
2.236	1.414	1	1.414	2.236	3.162	4.123	5.099	6.083	7.071	8.062
2	1	0	1	2	3	4	5	6	7	8
2.236	1.414	1	1.414	2.236	3.162	4.123	5.099	6.083	7.071	8.062
2.828	2.236	2	2.236	2.828	3.606	4.472	5.385	6.325	7.28	8.246
3.606	3.162	3	3.162	3.606	4.243	5	5.831	6.708	7.616	8.544
4.472	4.123	4	4.123	4.472	5	5.657	6.403	7.211	8.062	8.944
5.385	5.099	5	5.099	5.385	5.831	6.403	7.071	7.81	8.602	9.434
6.325	6.083	6	6.083	6.325	6.708	7.211	7.81	8.485	9.22	10
7.28	7.071	7	7.071	7.28	7.616	8.062	8.602	9.22	9.899	10.63
8.246	8.062	8	8.062	8.246	8.544	8.944	9.434	10	10.63	11.31
9.22	9.055	9	9.055	9.22	9.487	9.849	10.3	10.82	11.4	12.04
10.2	10.05	10	10.05	10.2	10.44	10.77	11.18	11.66	12.21	12.81
11.18	11.05	11	11.05	11.18	11.4	11.7	12.08	12.53	13.04	13.6
12.17	12.04	12	12.04	12.17	12.37	12.65	13	13.42	13.89	14.42
13.15	13.04	13	13.04	13.15	13.34	13.6	13.93	14.32	14.76	15.26

Figure 3.8: Distance cost matrix of the workspace map.



10.21954	10.05539	10	10.05539	10.21954	1009.487	10.84886	1010.296	11.81665	12.40175	13.04159
9.246211	9.062258	9	9.062258	9.246211	1008.544	9.944272	1009.434	11	11.63015	12.31371
8.28011	8.071068	8	8.071068	8.28011	1007.616	9.062258	1008.602	10.21954	10.89949	11.63015
7.324555	7.082763	7	7.082763	7.324555	1006.708	8.211103	1007.81	9.485281	10.21954	11
6.385165	6.09902	6	6.09902	6.385165	1005.831	7.403124	1007.071	8.81025	9.602325	10.43398
5.472136	5.123106	5	5.123106	5.472136	1005	6.656854	1006.403	8.211103	9.062258	9.944272
4.605551	4.162278	4	4.162278	4.605551	1004.243	6	1005.831	7.708204	8.615773	9.544004
3.828427	3.236068	3	3.236068	3.828427	1003.606	5.472136	1005.385	7.324555	8.28011	9.246211
3.236068	2.414214	2	2.414214	3.236068	1003.162	5.123106	1005.099	7.082763	8.071068	9.062258
3	2	1	2	3	1003	5	1005	7	8	9
3.236068	2.414214	2	2.414214	3.236068	1003.162	5.123106	1005.099	7.082763	8.071068	9.062258
3.828427	3.236068	3	3.236068	3.828427	1003.606	5.472136	1005.385	7.324555	8.28011	9.246211
4.605551	4.162278	4	4.162278	4.605551	1004.243	6	1005.831	7.708204	8.615773	9.544004
5.472136	5.123106	5	5.123106	5.472136	1005	6.656854	1006.403	8.211103	9.062258	9.944272
6.385165	6.09902	6	6.09902	6.385165	1005.831	7.403124	1007.071	8.81025	9.602325	10.43398
7.324555	7.082763	7	7.082763	1006.325	1006.708	1007.211	1007.81	9.485281	10.21954	11
8.28011	8.071068	8	8.071068	1007.28	1007.616	1008.062	1008.602	10.21954	10.89949	11.63015
9.246211	9.062258	9	9.062258	9.246211	9.544004	9.944272	1009.434	11	11.63015	12.31371
10.21954	10.05539	10	10.05539	10.21954	10.48683	1009.849	1010.296	1010.817	12.40175	13.04159
1010.198	11.04988	11	11.04988	11.19804	11.44031	1010.77	1011.18	1011.662	13.20656	1012.806
1011.18	12.04536	12	12.04536	12.18034	12.40175	12.7047	13.08305	13.52996	14.0384	1013.601
1012.166	1012.042	13	13.04159	13.16553	13.36932	13.64911	14	14.41641	1013.892	1014.422
1013.153	1013.038	1013	14.0384	14.15295	14.34166	14.60147	14.92839	1014.318	1014.765	1015.264

Figure 3.9: Total cost matrix which represent the value of f(v) for each cell.

There are two ways to design the A* algorithm to search for the next cell with the lowest $f(v)$ value. The first method is to search only in four directions: up, down, left and right (Figure 3.10-a). The second method is to search in eight adjacent cells as the next search direction (Figure 3.10-b). Figure 3.10 demonstrate the two types of search direction. The second method generate a shorter path than the first one, but it’s more complex in implementation. In this thesis, the first method will be used.

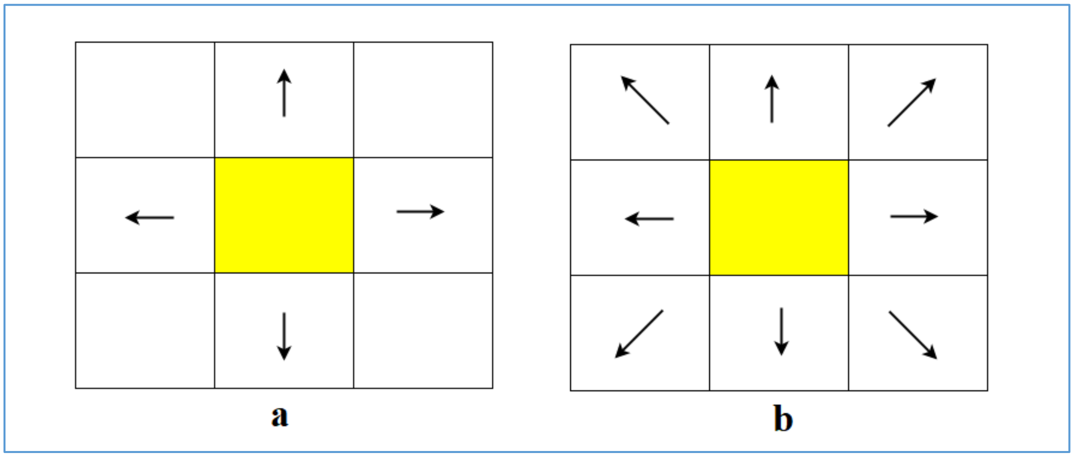
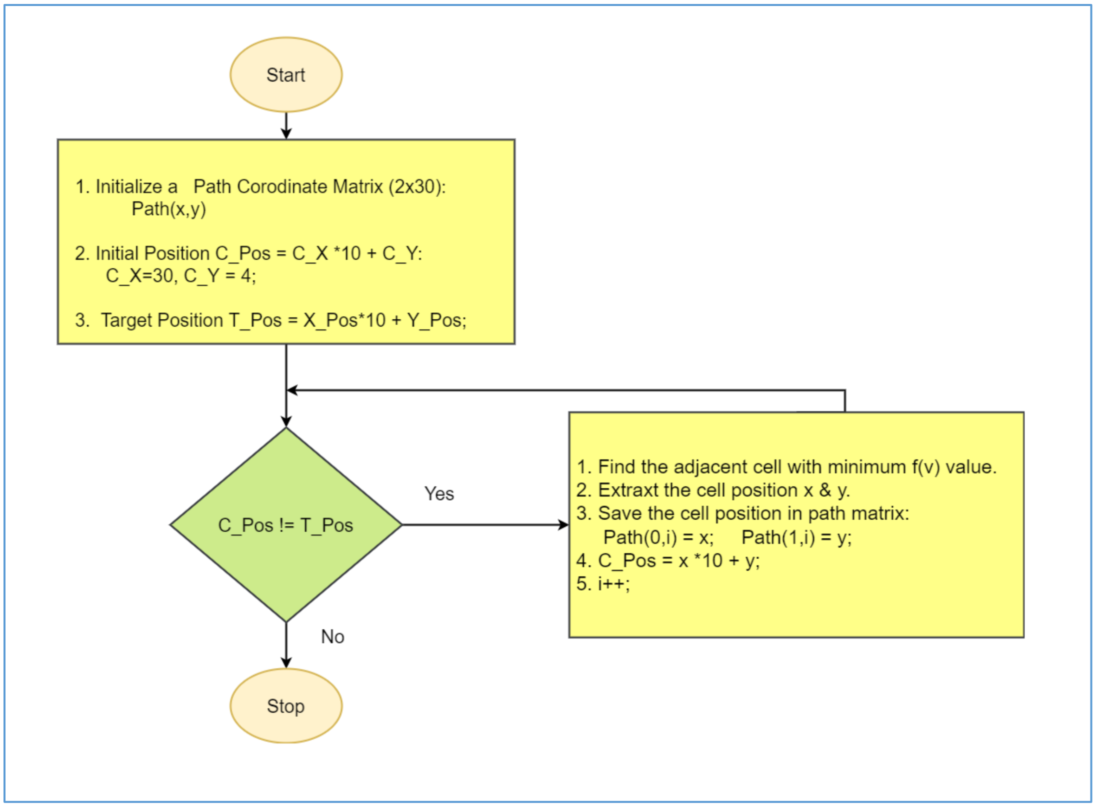


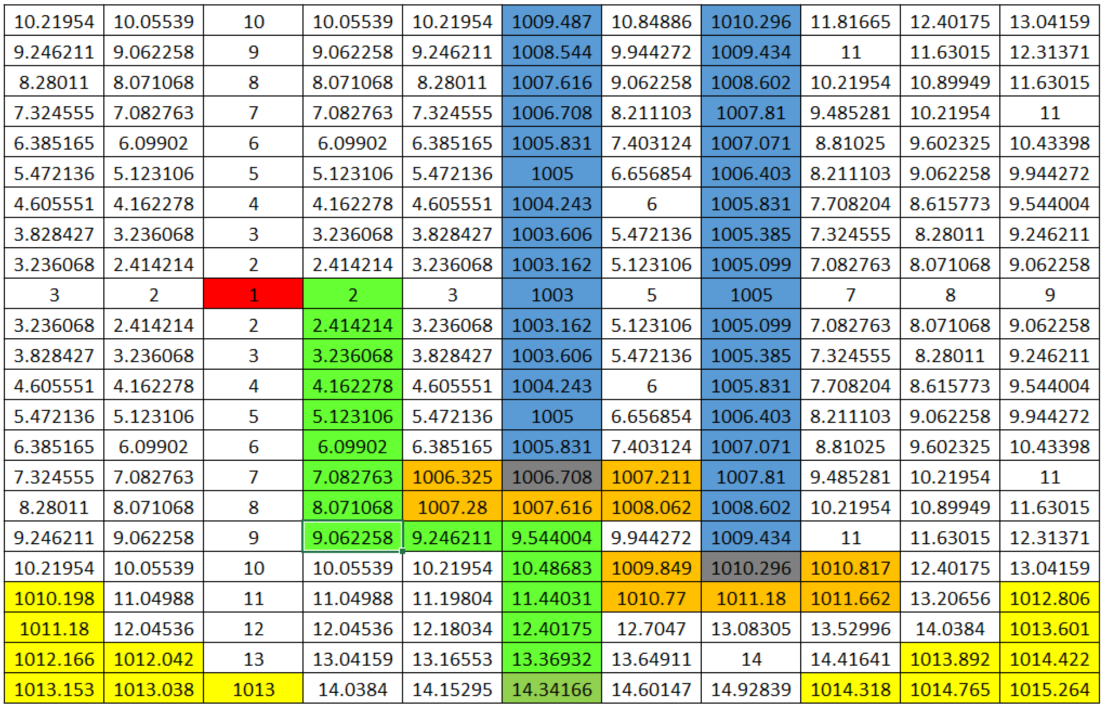
Figure 3.10: A search directions.*



1. Initialize a Path Corodinate Matrix (2x30):
   Path(x,y)
2. Initial Position C_Pos = C_X *10 + C_Y:
   C_X=30, C_Y = 4;
3. Target Position T_Pos = X_Pos*10 + Y_Pos;

1. Find the adjacent cell with minimum f(v) value.
2. Extraxt the cell position x & y.
3. Save the cell position in path matrix:
   Path(0,i) = x; Path(1,i) = y;
4. C_Pos = x *10 + y;
5. i++;

Figure 3.11: A algorithm flowchart.*



10.21954	10.05539	10	10.05539	10.21954	1009.487	10.84886	1010.296	11.81665	12.40175	13.04159
9.246211	9.062258	9	9.062258	9.246211	1008.544	9.944272	1009.434	11	11.63015	12.31371
8.28011	8.071068	8	8.071068	8.28011	1007.616	9.062258	1008.602	10.21954	10.89949	11.63015
7.324555	7.082763	7	7.082763	7.324555	1006.708	8.211103	1007.81	9.485281	10.21954	11
6.385165	6.09902	6	6.09902	6.385165	1005.831	7.403124	1007.071	8.81025	9.602325	10.43398
5.472136	5.123106	5	5.123106	5.472136	1005	6.656854	1006.403	8.211103	9.062258	9.944272
4.605551	4.162278	4	4.162278	4.605551	1004.243	6	1005.831	7.708204	8.615773	9.544004
3.828427	3.236068	3	3.236068	3.828427	1003.606	5.472136	1005.385	7.324555	8.28011	9.246211
3.236068	2.414214	2	2.414214	3.236068	1003.162	5.123106	1005.099	7.082763	8.071068	9.062258
3	2	1	2	3	1003	5	1005	7	8	9
3.236068	2.414214	2	2.414214	3.236068	1003.162	5.123106	1005.099	7.082763	8.071068	9.062258
3.828427	3.236068	3	3.236068	3.828427	1003.606	5.472136	1005.385	7.324555	8.28011	9.246211
4.605551	4.162278	4	4.162278	4.605551	1004.243	6	1005.831	7.708204	8.615773	9.544004
5.472136	5.123106	5	5.123106	5.472136	1005	6.656854	1006.403	8.211103	9.062258	9.944272
6.385165	6.09902	6	6.09902	6.385165	1005.831	7.403124	1007.071	8.81025	9.602325	10.43398
7.324555	7.082763	7	7.082763	1006.325	1006.708	1007.211	1007.81	9.485281	10.21954	11
8.28011	8.071068	8	8.071068	1007.28	1007.616	1008.062	1008.602	10.21954	10.89949	11.63015
9.246211	9.062258	9	9.062258	9.246211	9.544004	9.944272	1009.434	11	11.63015	12.31371
10.21954	10.05539	10	10.05539	10.21954	10.48683	1009.849	1010.296	1010.817	12.40175	13.04159
1010.198	11.04988	11	11.04988	11.19804	11.44031	1010.77	1011.18	1011.662	13.20656	1012.806
1011.18	12.04536	12	12.04536	12.18034	12.40175	12.7047	13.08305	13.52996	14.0384	1013.601
1012.166	1012.042	13	13.04159	13.16553	13.36932	13.64911	14	14.41641	1013.892	1014.422
1013.153	1013.038	1013	14.0384	14.15295	14.34166	14.60147	14.92839	1014.318	1014.765	1015.264

Figure 3.12: Free obstacles path (sequence of green cells) as generated by the A* algorithm.

3.3 Path Planning and Trajectory Generation

The optimized shortest path is generated by means of the A* algorithm, which ensures an obstacles free route for the robot from the robot initial posture to the tennis ball posture. This path will be broken down to sub-pieces in order to plan a trajectory. Thus, a path and trajectory planning algorithm will be used in order to lead the robot along the generated A* algorithm path. This process can be accomplished by finding a geometrical path in the task space and defining a timing law. Figure 3.13 shows the way that the A* path is divided to sub-pieces to generate a continuous arc path. This will allow the robot to navigate in an arc length paths around the corners, instead of stopping and turning around the corner. The main path started from the green cell and ends at the red cell. Firstly, the wye points is generated at the start and end of a path piece. We mean by a wye point is the points where a new piece of path is initiated from cell to cell. Each wye point will be given a specific configuration (x, y, θ) according to its location. Then a planning via Cartesian polynomials will be used to plan a path between each consecutive wye points.

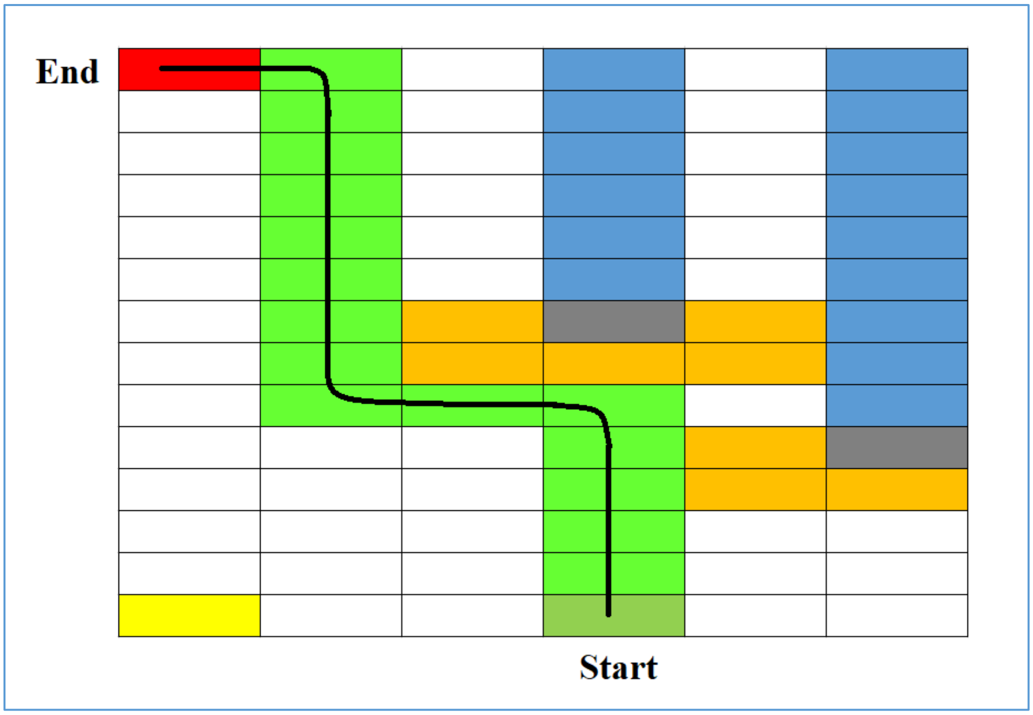


Figure 3.13: A Path converted to an arc path.*

Using equations (1.9) and (1.10) as illustrated in chapter 1, a path can be planned using a parametrized Cartesian polynomial. Due to the fact that the robot orientation is of main concerns during the navigation process. The initial and final orientation of the robot must be imposed within the planning equations. And by referring to chapter 1.3.3, it is obvious that the initial and final configuration related to the orientation is not known. And at the same time, the orientation at each point is related to $\dot{x}\ and\ \dot{y}$. Thus, it is necessary to impose an additional boundary condition.

Differential Flatness

The kinematic model of the tennis mobile robot is considered to be a flat system. Where, the outputs are the Cartesian coordinates of the robot mid-point ($Q\ point\ see\ Figure$ 2.3). Whenever the flat outputs (x, y) are determined, the reference trajectory as well as the corresponding control inputs can be expressed in terms of (x, y) and their derivatives. From the inverse kinematic model, we can determine the control inputs $\left(v(t),\ \omega(t)\right)$ that will be exploited to steer the robot along a Cartesian path. This path will be generated from the given initial and final configurations. From the kinematic model which was detailed in section 2.2, the non-holonomic constraints impose the following condition for geometric admissibility of the path. [21]

$$[\sin\theta \quad -\cos\theta \quad 0]\,\dot{q} = \dot{x}\sin\theta - \dot{y}\cos\theta = 0 \tag{3.4}$$

The above equation shows that the tangent to the Cartesian path must be aligned with the robot sagittal axis. As a result, any path with a broken lines (discontinuity) is not admissible. The admissible paths of the tennis robot are the solutions of the system: [21]

$$\dot{x} = \tilde{v}\cos\theta \tag{3.5}$$

$$\dot{y} = \tilde{v}\sin\theta \tag{3.6}$$

$$\dot{\theta} = \widetilde{\omega} \tag{3.7}$$

Where $\tilde{v}$ & $\widetilde{\omega}$ are related to the control inputs v & ω by:

$$v(t) = \tilde{v}(s)\,\dot{s}(t) \tag{3.8}$$

$$\omega(t) = \widetilde{\omega}(s)\,\dot{s}(t) \tag{3.9}$$

Where the control inputs in time domain can be obtained from the kinematic model as:

$$v(t) = \pm\sqrt{\dot{x}(t)^2 + \dot{y}(t)^2} \tag{3.10}$$

$$\theta(t) = Atan2(\,\dot{y}(t), \dot{x}(t)) \tag{3.11}$$

And with referring to the inverse kinematic model, the wheels angular speed can be obtained according to (2.9) & (2.10).

Now, given a Cartesian path ($x(s), y(s)$), the associated state trajectory is:

$\overline{\boldsymbol{q}}(s) = [x(s) \quad y(s) \quad \theta(s)]$

Where:

$$\theta(s) = Atan2(\, \grave{y}(s),\ \grave{x}(s)\,) \qquad (3.12)$$

$$\tilde{v}(s) = \sqrt{\grave{x}(s)^2 + \grave{y}(s)^2} \quad (for\ forward\ motion) \qquad (3.13)$$

$$\widetilde{\omega}(s) = \frac{\grave{\grave{y}}(s)\grave{x}(s) - \grave{\grave{x}}(s)\grave{y}(s)}{\left(\grave{x}(s)\right)^2 + (\grave{y}(s))^2} \qquad (3.14)$$

From equation (3.12), it is known that:

$$\tan\theta = \frac{\grave{y}}{\grave{x}} \qquad (3.15)$$

By considering an additional two arbitrary constants $k_i\ and\ k_f$, where $k_i \neq 0\ and\ k_f \neq 0$. The equation (3.15) becomes:

$$\frac{\grave{y}}{\grave{x}} = \frac{k_i \sin\theta_i}{k_i \cos\theta_i} \ , \quad at\ s = 0 \qquad (3.16)$$

$$\frac{\grave{y}}{\grave{x}} = \frac{k_f \sin\theta_f}{k_f \cos\theta_f} \ , \quad at\ s = 1 \qquad (3.17)$$

The equations (3.16) and (3.17) guarantees that the initial and final orientation are satisfied. Where k_i & k_f represents the initial and final velocities respectively in the geometric model. Now subjecting equations (1.9) and (1.10) to the initial conditions (x_i, y_i, θ_i) and final conditions$\left(x_f, y_f, \theta_f\right)$, then solving for the equation's coefficients.

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & s_i & {s_i}^2 & {s_i}^3 \\ 1 & s_f & {s_f}^2 & {s_f}^3 \\ 0 & 1 & 2s_i & 3{s_i}^2 \\ 0 & 0 & 2s_f & 3{s_f}^2 \end{bmatrix}^{-1} \begin{bmatrix} x_i \\ x_f \\ k_i \cos\theta_i \\ k_f \cos\theta_f \end{bmatrix} \tag{3.18}$$

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 & s_i & {s_i}^2 & {s_i}^3 \\ 1 & s_f & {s_f}^2 & {s_f}^3 \\ 0 & 1 & 2s_i & 3{s_i}^2 \\ 0 & 0 & 2s_f & 3{s_f}^2 \end{bmatrix}^{-1} \begin{bmatrix} y_i \\ y_f \\ k_i \sin\theta_i \\ k_f \sin\theta_f \end{bmatrix} \tag{3.19}$$

Now, an interpolation scheme will be applied between each two adjacent cells within the A* path. Thus, each cell in the path will be given a specific coordinated (x_n, y_n, θ_n) according to its location in the path.

Wey Points Generation

Each cell in the workspace map represents a Wey point. Thus, each cell has a specific configuration(x_n, y_n, θ_n). The x_n & y_n are specified according to the cell position in the map, while θ_n will be calculated based on the navigation direction between the current cell and the next cell. Using the following formula, the orientation at each Wey point can be determined. The first cell in the path will be given a 90^oi.e. $(24, 4, 90^o)$.

$$\theta(s) = Atan2(\ x_n - x_{n+1}, y_{n+1} - y_n\ ) \tag{3.20}$$

A* Path Smoothening

After generating a configuration for each Wey point, an accumulative interpolation between each adjacent cells is applied. The configuration of each Wey point must be converted to a metric unit by multiplying the $(x_n \, \& \, y_n)$ by $(0.3\ m)$. Figure 3.14 represent the algorithm that will generate an arc path. Accordingly, the robot will make an arc path maneuvering in order to turn around sharp corner. The corners curvature are related to the values of the constants k_i & k_f.

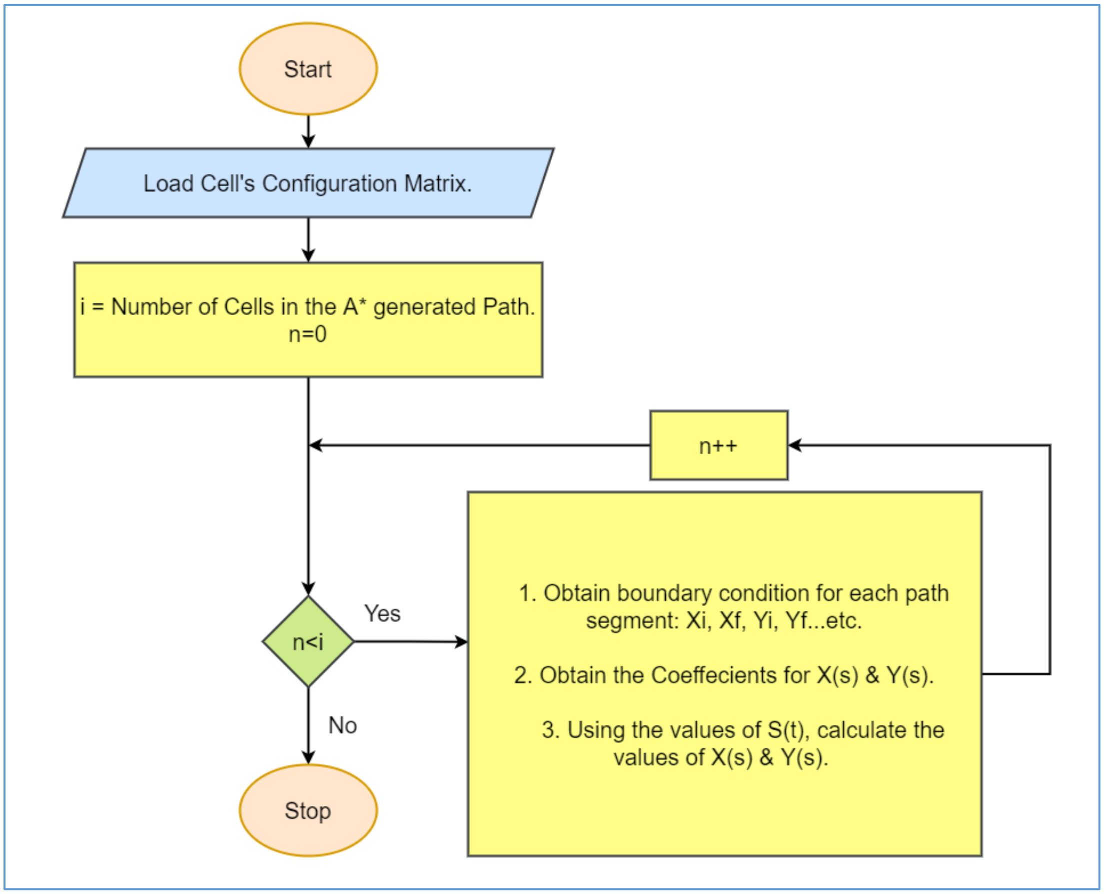

Start
Load Cell's Configuration Matrix.
i = Number of Cells in the A* generated Path.
n=0
n++
n<i
Yes
No
Stop
1. Obtain boundary condition for each path segment: Xi, Xf, Yi, Yf...etc.
2. Obtain the Coeffecients for X(s) & Y(s).
3. Using the values of S(t), calculate the values of X(s) & Y(s).

Figure 3.14: Algorithm for Path planning between each adjacent cells in the A path generated.*

3.4 Motion Control

In this section, a low level control loop will be designed to implement a trajectory tracking, such that it accept as inputs a reference trajectory value (wheels angular speeds $\dot{\theta}_R$ & $\dot{\theta}_L$) that is obtained in the previous section. Then, the reference values will be reproduced as accurately as possible by a standard regulation action (PI controller). The PI controller is integrated in the hardware architecture (Driver circuit). Figure below illustrate the scheme for the wheel space control.

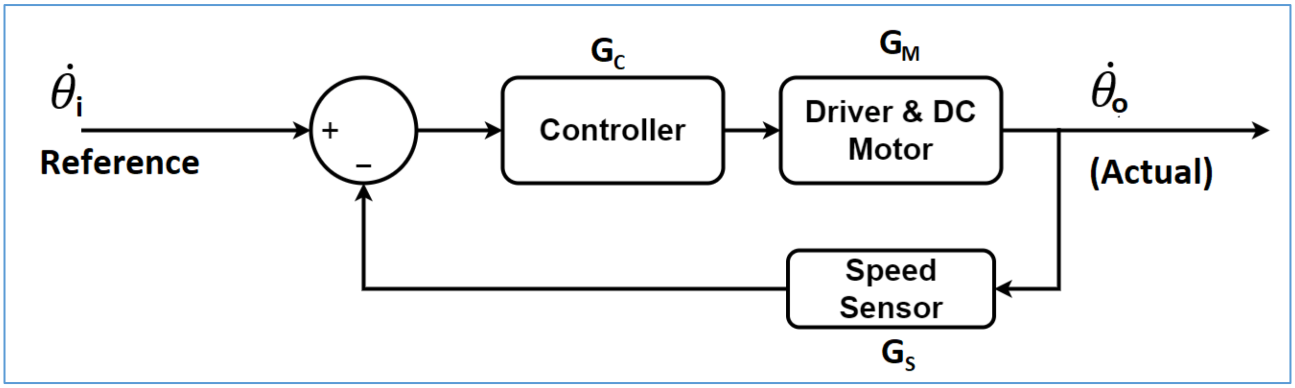


Figure 3.15: General scheme for the wheel space control.

The transfer function of the robot DC motor with the gear for each wheel can be approximated as a first order system [29]:

$$\frac{\dot{\theta}_m(s)}{E_a(s)} = \frac{K_t/(R_a J_m)}{\left[s+\frac{1}{J_m}\left(D_m+\frac{K_t K_b}{R_a}\right)\right]} \tag{3.21}$$

Or can be simplified to:

$$\frac{\dot{\theta}_m(s)}{E_a(s)} = \frac{K}{s+a} \tag{3.22}$$

Where:

$$K = \frac{K_t}{(R_a J_m)} \tag{3.23}$$

$$a = \frac{1}{J_m}\left(D_m + \frac{K_t K_b}{R_a}\right) \tag{3.24}$$

Substituting DC motor parameters [28], (3.21) become:

$$\frac{\dot{\theta}_m(s)}{E_a(s)} = \frac{74.12}{s+8.89} \tag{3.25}$$

The closed loop transfer function:

$$\frac{\dot{\theta}_o(s)}{\dot{\theta}_i(s)} = \frac{K(K_P s+K_I\ )}{s^2+s(a+KK_P)+KK_I} \qquad (3.26)$$

Where:

$\dot{\theta}_i$: Represents the reference input wheel speed (RPM).

$\dot{\theta}_o$:Represents the actual wheel output speed (RPM).

K_P:The proportional controller gain.

K_I: Is the integrator controller time gain (1/sec.)

The general second order transfer function look like this [29]:

$$G_{CL} = \frac{\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2} \qquad (3.27)$$

Where:

ω_n: Represents the natural frequency.

ζ:Represents the damping ratio.

The percent overshoot (%OS) and the settling time for the second order system can be obtained using (3.28) & (3.29) respectively [29].

$$OS\% = \ e^{-(\zeta\pi/\sqrt{1-\zeta^2})} * 100\% \qquad (3.28)$$

$$T_S = \frac{4}{\zeta\omega_n} \qquad (3.29)$$

By choosing the settling time and overshot to be 0.1 sec. and 1% respectively, the desired closed loop transfer function denominator:

$$s^2 + 80\ s + 2322.28 \qquad (3.30)$$

By equating 3.30 with the denominator of 3.26:

$K_P = 2.5$

$K_I = 31.33s^{-1}$

The controller parameters will be adjusted in the DC motor driver circuit, Moreover, the DC motor driver circuit provides an auto tuning feature. The auto tuning mode identify the dc motor model parameters. And accordingly, obtaining the best values for the controller. Thus, the controller parameters design are presented her for the sake of completeness.

4

ROS Design and Implementation

This chapter will show how the ROS environment will be used to implement the algorithms that will be responsible for performing the previously mentioned robot tasks. Such as, Object detection, localization, motion planning, and motion control. ROS nodes will be used to execute all algorithms related to the desired tasks. Furthermore, in the following sections, an explanation of the robot ROS topics that are in charge of exchanging appropriate data between these nodes will be discussed. Furthermore, a complete demonstration of the hardware implementation with constraints and solutions is presented.

4.1 Robot System ROS Design

Figure 4.1 shows the ROS architecture in the tennis mobile robot. The red circles represent the nodes, while the blue rectangles represent the topics. It's noticeable that some nodes are subscribers, others are publishers, while the remaining nodes are both. One can distinguished a node type by the direction of the arrow. If the arrow head direction is from the topic toward the node, then the corresponding node is a subscriber node. On the other hand, if the arrow head direction from the node towards the topic, then the node is a publisher.

Each subscriber node is an interrupt driven node. This means a node will not be executed unless a new data is published to its subscribed topic rather than freely executed at any time. e.g. the image processing node (YOLO node) will not be executed until a new data is published from the camera node to the RGB image topic. This will improve the overall performance of the robot by reducing the processing load on the main processing unit. Interrupt driven nodes allows the processing unit to accomplish the other nodes as fast as possible. Moreover, the camera node will not be executed till the last node (Motion Control) finished and publish a Done-status value. A ROS package is created which contains all the nodes and configuration files

In order to increase the robot performance and reduce the execution time for each node. Many techniques are used during the implementation:

1. All the nodes are written with C++ language due to its high performance and closure to hardware. Since all the C++ source code finally becomes a machine codes, which is faster.
2. All nodes are written from scratch except YOLO. To ensure an optimal algorithm implementation. Hence, some general-purpose applicable ROS libraries require an additional execution time.
3. Configure the nodes to be run on interrupt-driven mode. Thus, to prevent the CPU from executing unnecessary nodes. This means the nodes will be executed only when needed.

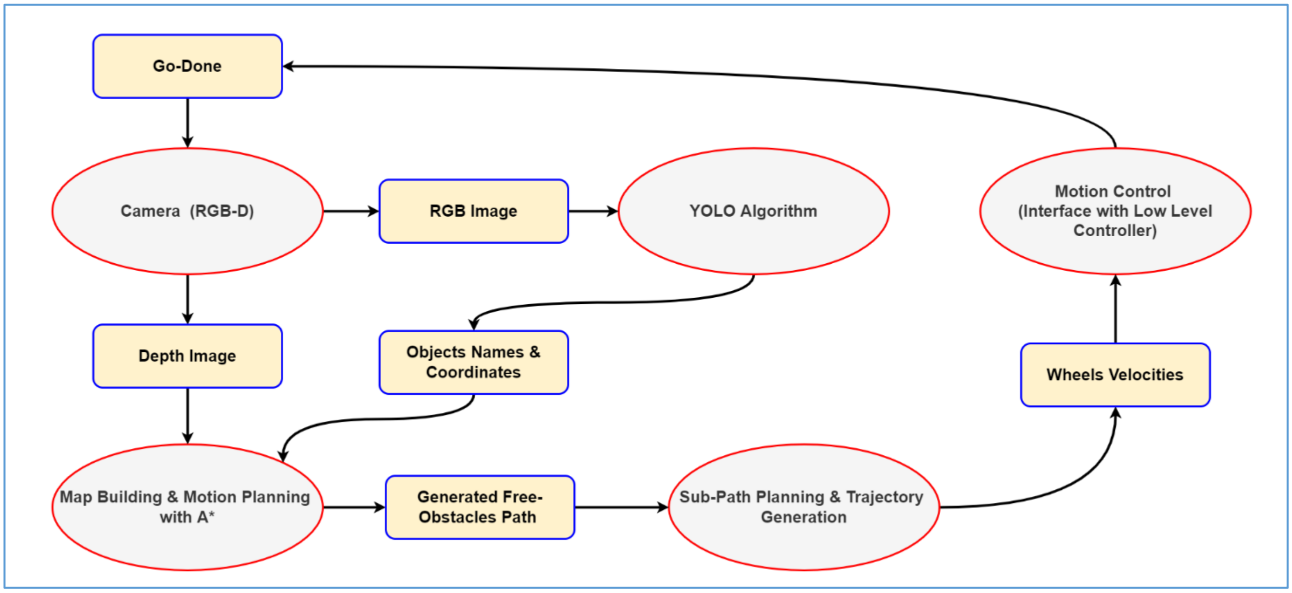

Go-Done
Camera (RGB-D)
RGB Image
YOLO Algorithm
Motion Control (Interface with Low Level Controller)
Depth Image
Objects Names & Coordinates
Wheels Velocities
Map Building & Motion Planning with A*
Generated Free-Obstacles Path
Sub-Path Planning & Trajectory Generation

Figure 4.1: ROS system design. Ovals represent the system nodes while rectangles represent the system topics.

4.1.1 System Nodes

1. *Camera Node:* camera node is classified as a publisher node which publishes the raw RGB image data, in addition to the depth data under the *RGB (/camera/rgb /image_raw) Image* topic. This node is responsible of executing the camera driver framework SDK for both, RGB camera and Depth camera sensors. All configuration parameters related to both cameras can be modified to fit the appropriate requirements, such as frame rate and resolution. The published information are RGB image and depth image. Figure 4.2 represents the execution sequence within the node. The package, source code and configurations files are illustrated in ROS wrapper for Intel devices on GitHub [36].

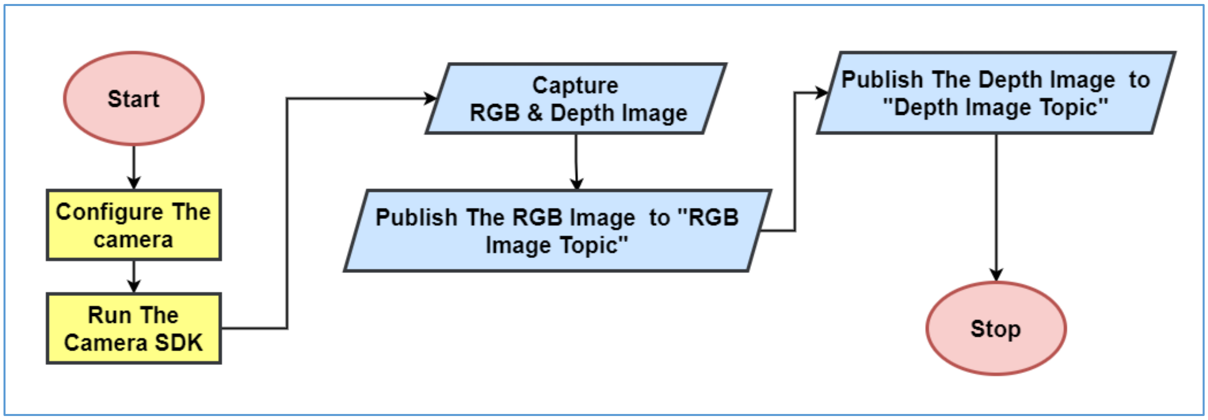
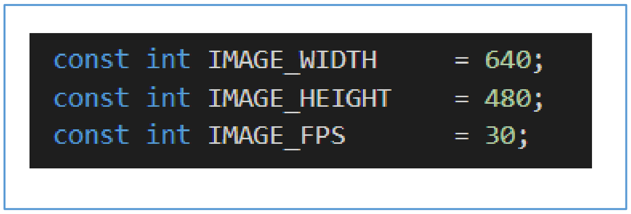

Start
Configure The camera
Run The Camera SDK
Capture RGB & Depth Image
Publish The RGB Image to "RGB Image Topic"
Publish The Depth Image to "Depth Image Topic"
Stop

Figure 4.2: Camera Node Algorithm.

The header file (*constants.h*) which is a part of the camera node package [36], was used to configure basic camera setting such as image resolution and camera frame rate. A snippet of code for basic camera settings is shown in Figure 4.3. The following ROS command was used to test this node:

Roslaunch realsense2_camera rs_camera.launch

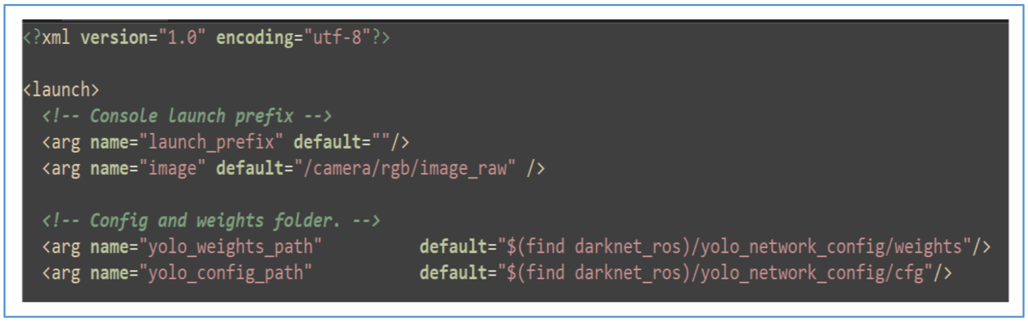


```
const int IMAGE_WIDTH      = 640;
const int IMAGE_HEIGHT     = 480;
const int IMAGE_FPS        = 30;
```

Figure 4.3: Image setting configuration parameters.

2. *YOLO Node:* This node is in charge of executing the YOLO algorithm, which was previously described in Chapter 1. The YOLO node serves as both a publisher and a subscriber node. Simultaneously, this node subscribes to the RGB image topic (/camera/rgb/image raw) and publishes the detected object classes and coordinates within the image, as described in Chapter three. The YOLO package (*darknet_ros*) -which available on GitHub [37] - contains an *xml* launch file (*darknet_ros.launch*), this file was used to specify the topic name and the weight directory which is essential for the YOLO's algorithm for listening and classification as shown in Figure 4.4. This node was tested using the following ROS command:

   Roslaunch darknet_ros darknet_ros.launch



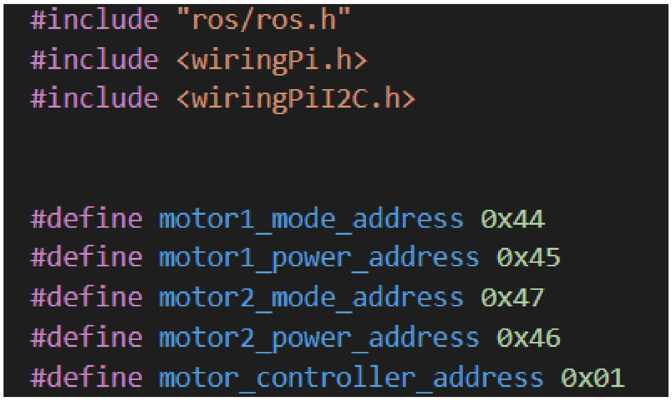
```xml
<?xml version="1.0" encoding="utf-8"?>

<launch>
  <!-- Console launch prefix -->
  <arg name="launch_prefix" default=""/>
  <arg name="image" default="/camera/rgb/image_raw" />

  <!-- Config and weights folder. -->
  <arg name="yolo_weights_path"          default="$(find darknet_ros)/yolo_network_config/weights"/>
  <arg name="yolo_config_path"           default="$(find darknet_ros)/yolo_network_config/cfg"/>
```

Figure 4.4: Snippet code from YOLO xml launch file.

3. *Map Building and Motion Planning with A*:* This node is a subscriber and a publisher node. Two roles will be performed by this node. The first role, is to build the robot environment map -as explained in chapter three section two- using the information provided from the YOLO algorithm node. Second role is, to find the shortest path to the target (Tennis ball) without colliding with obstacles. The output of this nodes is the optimized path coordinates values(x, y). Which will be published to the topic related to the next node.

4. *Sub-Path planning and Trajectory Generation*: Inside this node, four algorithms were executed after a new data is received from the previous node. The first algorithm is the ***wye*** point creation. The second one is the path generation using Cartesian polynomial interpolation as explained previously. The third algorithm is the trajectory calculations. Lastly, calculating the trajectory vectors that will be used to evaluate the robot control inputs $\big(v(t),\ \omega(t)\big)$, then calculating the corresponding wheels velocities$(\dot{\theta}_R(t),\ \dot{\theta}_L(t))$. Finally, the velocity vector for each wheel will be published for the last motion control node under the topic *Wheels Velocities*.

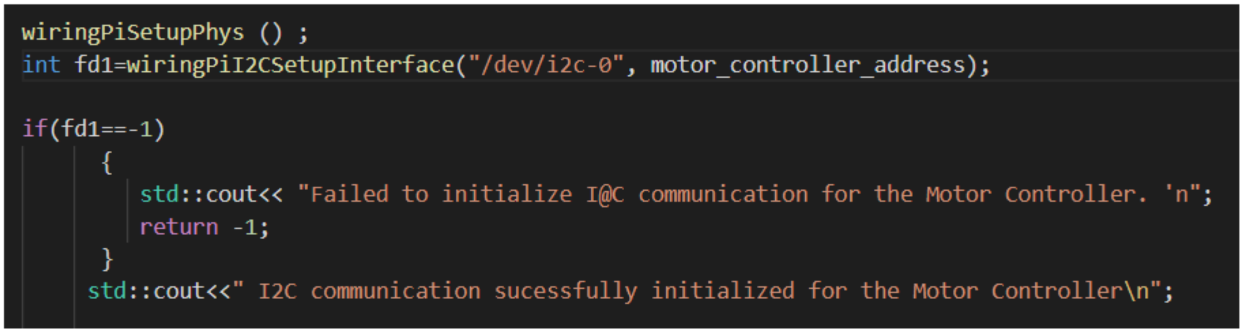
5. *Motion Control:* This node is a subscriber node which listens to the *Wheels Velocities* topic. It's responsible for initializing the (I^2C) protocol, configuring the DC motor controller, and transmitting physical quantities of the wheels velocity $(\dot{\theta}_R(t),\ \dot{\theta}_L(t))$ to the DC motor controller based on (I^2C) protocol. The script shown below in Figure 4.5 represents a snippet code used to define the addresses for the DC motor controller, Right motor address channel within the controller, and the left motor address channel within the controller. Based on the DC motor controller datasheet, the configuration registers addresses were assigned [33].



```
#include "ros/ros.h"
#include <wiringPi.h>
#include <wiringPiI2C.h>


#define motor1_mode_address 0x44
#define motor1_power_address 0x45
#define motor2_mode_address 0x47
#define motor2_power_address 0x46
#define motor_controller_address 0x01
```

Figure 4.5: Snippet code for defining the addresses.

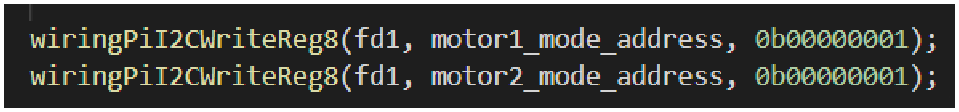
After defining the addresses, an initialization command was used to initialize the (I^2C). Snippet in Figure 4.6 shows the initialization command with a simple message system in case of communication frailer.



```
wiringPiSetupPhys () ;
int fd1=wiringPiI2CSetupInterface("/dev/i2c-0", motor_controller_address);

if(fd1==-1)
    {
      std::cout<< "Failed to initialize I@C communication for the Motor Controller. 'n";
      return -1;
    }
   std::cout<<" I2C communication sucessfully initialized for the Motor Controller\n";
```

Figure 4.6: $\left(I^2C\right)$ initialization command.

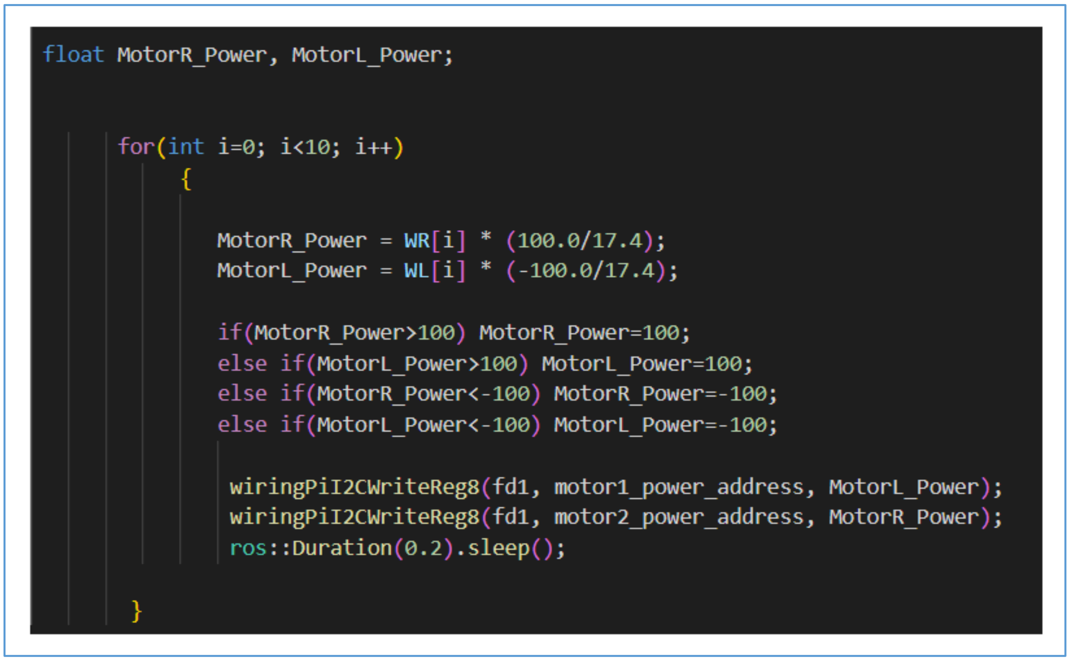
The DC motor controller is configured to make the motors run at constant speed mode. This mode will cause the firmware to adjust the motor power to compensate for the changing loads in order to maintain a constant motor speed. Figure 4.7 shows the configuration registers of the two motors.



```
wiringPiI2CWriteReg8(fd1, motor1_mode_address, 0b00000001);
wiringPiI2CWriteReg8(fd1, motor2_mode_address, 0b00000001);
```

Figure 4.7: Configuring the DC motor controller to run in constant speed mode.

Following the completion of the aforementioned configuration, the physical velocities quantities which represents the reference velocity values are transformed to electrical signals. Such that a control loops inside the DC motor controller are dedicated for each wheel. The controller measures the actual speeds from the wheels encoders and calculate the errors signals. Then an actuating voltage signal (PWM) is generated according to the error's signals, and then the PWM signal is delivered to the wheel's motors. The control loop inside the controller is out of the ROS management circle, i.e. the ROS does not read the wheels encoders. Each wheel velocity pair will be transmitted according to a timing schedule (every 0.02 second) to ensure a real time trajectory execution as illustrated by the snippet code shown in Figure 4.8. The *MotorR_Power* and *MotorL_Power* are variables represent the mapped velocity value for each wheel. The variables are a PWM values ranges from -100 to 100 [33]. The 100 represent maximum speed in clockwise direction, while the -100 represents the maximum speed in counter clockwise direction.



```
float MotorR_Power, MotorL_Power;


     for(int i=0; i<10; i++)
         {

            MotorR_Power = WR[i] * (100.0/17.4);
            MotorL_Power = WL[i] * (-100.0/17.4);

            if(MotorR_Power>100) MotorR_Power=100;
            else if(MotorL_Power>100) MotorL_Power=100;
            else if(MotorR_Power<-100) MotorR_Power=-100;
            else if(MotorL_Power<-100) MotorL_Power=-100;

             wiringPiI2CWriteReg8(fd1, motor1_power_address, MotorL_Power);
             wiringPiI2CWriteReg8(fd1, motor2_power_address, MotorR_Power);
             ros::Duration(0.2).sleep();

      }
```

Figure 4.8: Transmitting velocity values to the controller.

4.1.2 System Topics

1. *RGB Image*: This topic is an image data type, it saves the RGB images which published from the camera node, to be read by the YOLO node.

2. *Depth Image*: This topic is the same as the previous one, the topic saves the depth images which published by the camera node to be read by the mapping node.

3. *Object Names & Coordinates*: This topic saves multiple data related to object detection such as Object classes (type string) and objects position with the image (type int).

4. *Generated Free Obstacles Path:* The topic is an array data type (float array). Is used to save the path data after concatenating them in a one vector.

5. *Wheels Velocities:* This topic is of the same data time as in the previous one. It's used to save a one dimensional float array. $(\dot{\theta}_R vector,\ \dot{\theta}_L vector\ and\ timing\ vector\ cocatinated\ in\ a\ one\ vector)$.

6. *Go-Done*: This topic is used as an indicator for the accomplishment of the last node. It holds a string value ("Go" or "Done") according to the status of the nodes execution. The default value is the "Done", which indicate that the system is ready to start a new iteration of nodes execution. During the beginning of execution of the YOLO node, the node will publish a "GO" status. Eventually, the "Done" value will published only when the last node is finished.

4.2 Implementations

In this section, all previously explained methods and algorithms will be implemented in the ROS environment. The integration of these algorithms and theories results in a dependable machine for collecting tennis balls. A task (node) performed by the robot follows a sequentially executed process. In other words, the robot cannot plan a free obstacle path unless it knows the classes of the objects and their coordinated data. All other nodes follow the same principles. As a result, the execution begins at the time specified by the initialization of the first node. And the end time is determined by the execution of the last instruction in the last node. As a result, the system's overall execution time is not constant. This is due to the tennis balls' different physical locations, which will affect the path length. Furthermore, the presence of obstacles necessitates maneuvering movements, which lengthen the path. As a result, the execution time will vary. Moreover, as the amount of data in the environment grows, so will the execution time. Using faster actuators with shorter execution times is one way to speed up the collection process. Alternatively, faster processing units can be used to reduce the execution time for each node. Or by a combination of the two.

4.2.1 Hardware Implementation

In this section, a detailed overview of the robot hardware will be presented. Figure 4.9 illustrate the hardware block diagram of the system which detailed previously in Figure 3.2. An Intel RGB-D camera (D415) is mounted on the robot for the purpose of perceiving the robot environment. The depth camera can operate indoor and outdoor environment with a range up to 10 meters. The camera support interfacing with the host computer through a USB communication protocol [29]. A Raspberry Pi 4 Model B (RPI4B) is used as the robot mini-computer. And a two channels DC motor controller (HiTechnic) is used as the low level controller [28]. It has a connection of two DC motors and two incremental encoders. It’s a PID controller with an auto-tuning capability, it measures the motor speed for each channel and produce the appropriate PWM signal as the actuating signal to the DC motors. The auto-tuning property allow the controller to choose the optimal controller parameters (K_P, K_I & K_D) that produce the optimal transient and steady state requirements. Thus, it capable of deriving the wheels motors with the desired speed. The interfacing between the RPI4B and the controller is done through the I^2C serial communication protocol. Each motor is attached with an incremental encoder.

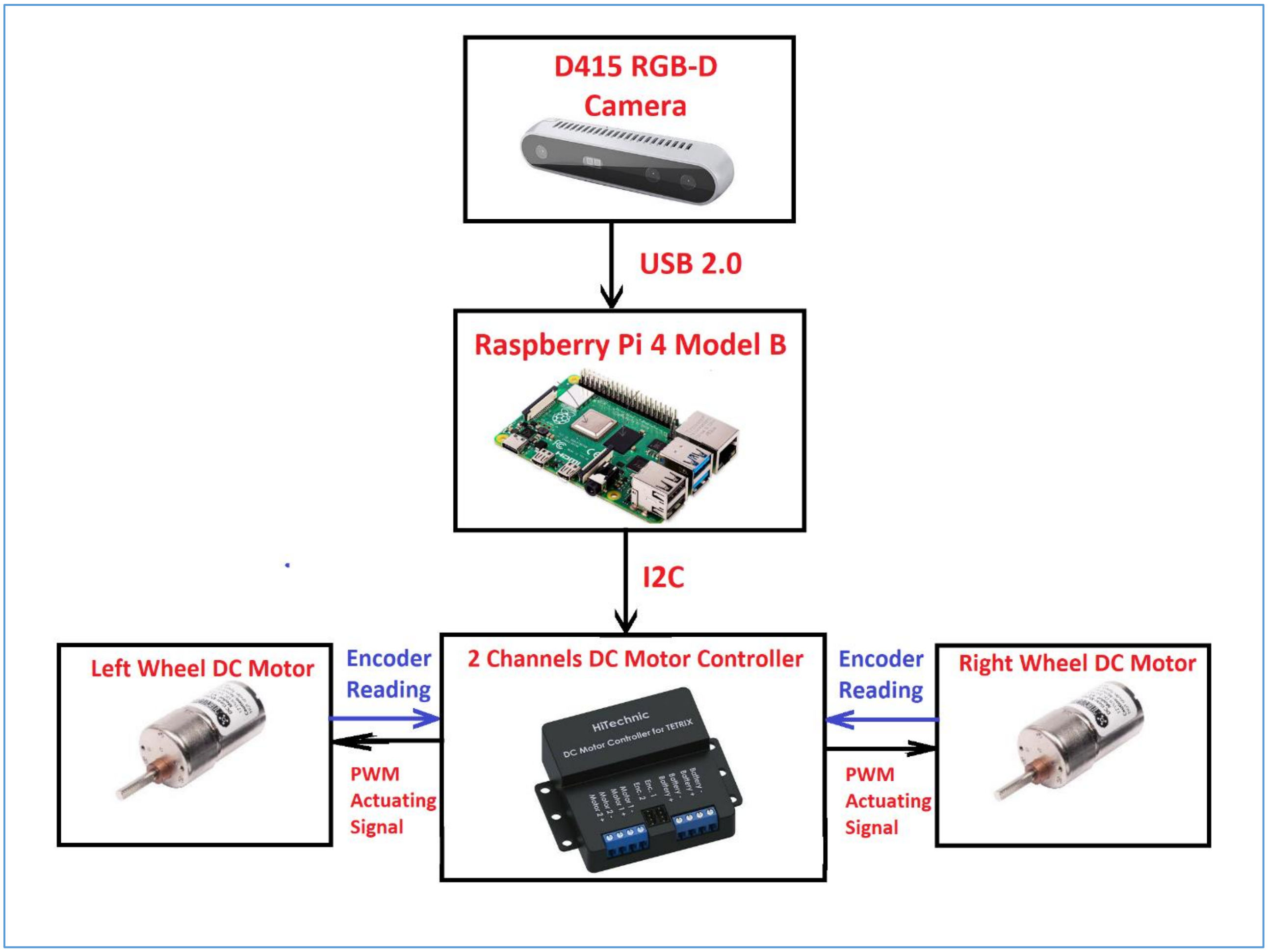


Figure 4.9: Robot hardware block diagram.

4.2.1.1 Raspberry Pi 4 Structure and Architecture [31]

The RPI4B is the first of a new generation of Raspberry Pi computers supporting more RAM and with significantly enhanced CPU, GPU, USB ports and I/O performance. The RPI4B is available with either 1, 2, 4 or 8 Gigabytes of LPDDR4 SDRAM.

The RPI4B is 40 $ single board computer with a Quad core 64-bit ARM-Cortex A72 processor. The RPI4B consumes power of maximum 15*W*. It has an overall dimension of 85mm×56mm×17mm. as mentioned previously, one of the objectives of this study is the development of a light weight mobile robot which requires small dimension to fit it with the robot mechanical structure. Thus, the size of RPI4B is suitable for this application. The RPI4B is also compatible with Ubuntu operating system which was chosen as the host operating system which provides a flexible framework to run ROS. Ubuntu MATE 20.04 is chosen to install the upgraded version of ROS Noetic [32].

RPI4B provides several communication ports for interfacing with other external modules such as USB port and serial communication modules (e.g. I^2C). The USB ports used to interface the RGB-D camera (Color and depth) with the RPI4B. Furthermore, the I^2C module is used to interface the actuators driver with the RPI4B.

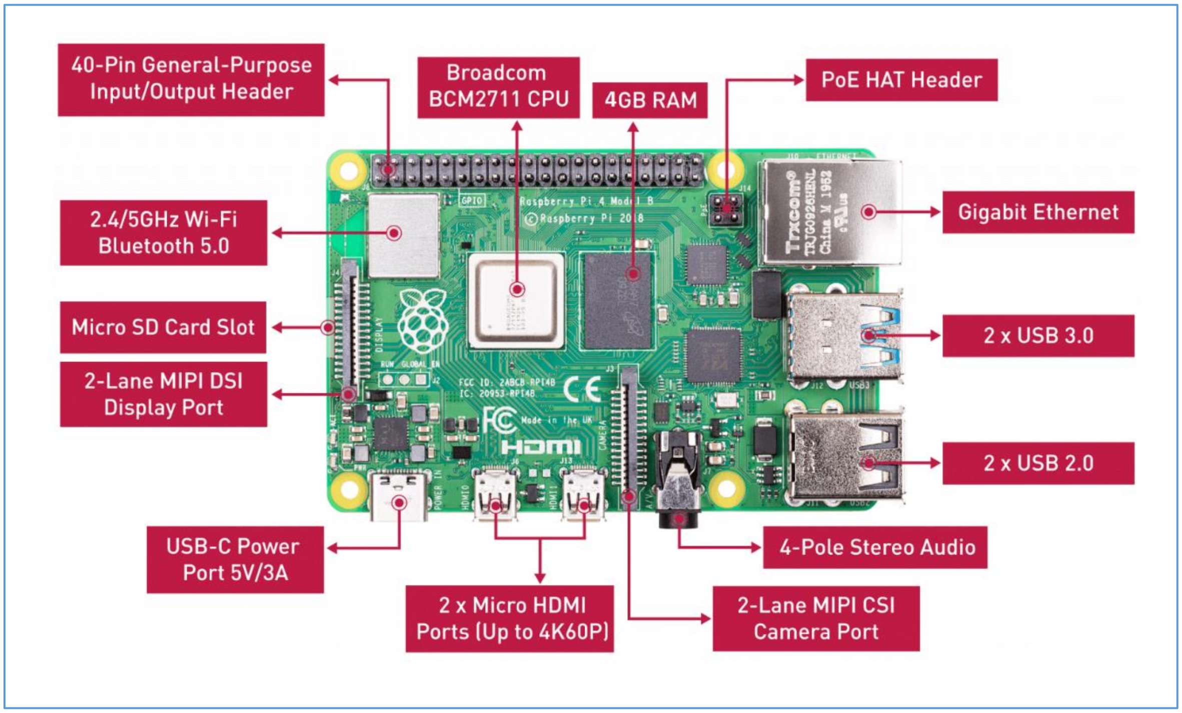

40-Pin General-Purpose Input/Output Header
Broadcom BCM2711 CPU
4GB RAM
PoE HAT Header
2.4/5GHz Wi-Fi Bluetooth 5.0
Gigabit Ethernet
Micro SD Card Slot
2 x USB 3.0
2-Lane MIPI DSI Display Port
2 x USB 2.0
USB-C Power Port 5V/3A
4-Pole Stereo Audio
2 x Micro HDMI Ports (Up to 4K60P)
2-Lane MIPI CSI Camera Port

Figure 4.10: RPI4B Hardware overview. [31]

4.2.1.2 ROS Installation on Single Board Computer (RPI4B)

ROS deployment on any computer requires a host computer to be installed with a Linux operating system. Thus, the ROS is a Meta operating system as described in chapter 1. The first step to deploy ROS, is to install one of the Linux distributions (e.g. Ubuntu Mate 20.04) which provides a hardware accessibility to all the RPI4B hardware modules. Particularly the GPIOs. In addition, to be compatible with the ROS version which intended to be used and its associated packages and libraries. Moreover, a very important point to be considered, is that the Linux distribution must be able to support all SDK's that will be used to create application for specific platforms. In this thesis the RPI4B must be installed with the RGB-D camera SDK in order to establish an interface communication with the camera through the USB port. To install ROS on Ubuntu Mate, steps should be followed as mentioned in wiki ROS page [35].

4.2.2 Real Time System Considerations

The cyclic scheduling approach is used to schedule tennis robot periodic tasks based on a precomputed off-line schedule table. The table specifies when each job of a periodic task is executed and whether it is repeatable. Real-time measurements of execution time for each node are performed during software implementations. These measurements are required for task scheduling purposes. A scheduling approach should be used when performing a schedulability test [34]. The table below shows the execution time for each node. The execution time measurements are performed programmably at the start of a node execution by triggering an output pin among the RPI4B GPIOs. Also, at the end of the node execution, the same pin is reset. Thus, a pulse signal is generated with a time width related to the execution time. This signal was displayed on an Oscilloscope which is connected to the GPIO pin. These numbers represent the worst-case scenario. As a result, the execution time varies depending on the CPU load imposed by the Ubuntu operating system.

Table 4.1: Nodes execution time.

Node Name	Maximum Theoretical Execution Time(Sec.)	Practical Execution Time (Sec.)
Camera Node (RGB-D)	$10 * 10^{-3}$	$5.97 * 10^{-3}$
YOLO	10	8.29
Motion Planning with A*	$15 * 10^{-3}$	$12.61 * 10^{-3}$
Sub-Path Planning & Trajectory Generation	$20 * 10^{-3}$	$17.42 * 10^{-3}$
Motion Control (Between two adjacent cells)	2	2

It should be noted that each trajectory section between two adjacent cells in the motion control node takes 2 seconds. The results of the above tests show that the system can meet the deadlines for each task.

4.3 Constraints and Solutions

Many challenges and constraints arose during the implementation process, as detailed below.

1. *ROS Incompatibility with Raspian OS:* ROS was initially installed on RPI4B in 2019. The Raspian Buster OS was used as the primary operating system on the RPI4B. It should be noted that this is the official operating system for the RPI4B. This operating system is lightweight and fully supported, and it supports all RPI4B GPIOs [31]. According to the WikiROS page, the Raspian OS is not compatible with ROS and its packages at the time of implementation. Initially, a few trial modifications are made to the Raspian workspace in order to install the ROS. These changes are made to the ROS installation directories within the Raspian workspace.

2. *The Raspian OS does not support the camera SDK (Intel RealSense D415):* When the camera SDK was installed on the Raspian OS, this issue arose. A pop-up error message explained that an incompatible architecture problem between the Raspian OS and the camera SDK had been detected. To address this issue, the Ubuntu Mate 16.0 OS was used instead of the Raspian Buster, but it was also incompatible with the camera SDK. Furthermore, the Ubuntu Mate 16.0 operating system does not support RPI4 GPIO access. As a result, a new version of Ubuntu Mate (version 20.04) was used. This version is compatible with the camera SDK and supports ROS installation. Ubuntu MATE is available for the Raspberry Pi in armhf (ARMv7 32-bit) and arm64 (ARMv8 64-bit) images [38]. However, when compared to the Raspian Buster, the Ubuntu Mate OS is considered to be a heavier OS on the RPI4B CPU.

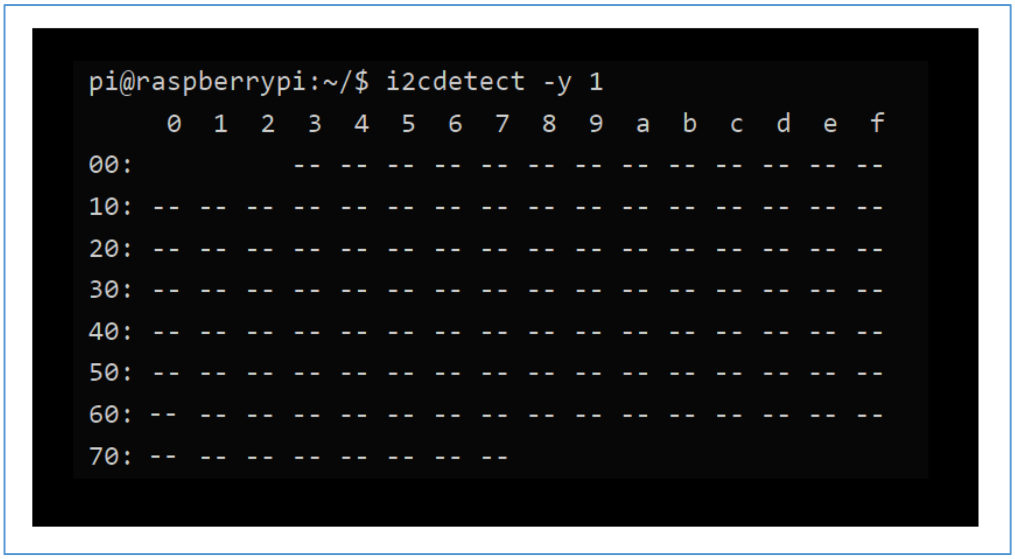
3. *Large execution time for the YOLO algorithm:* To detect and classify objects, the YOLO algorithm employs a convolutional network. This type of algorithm performs better when run on a graphics processing unit (GPU). This is known as hardware acceleration. The YOLO algorithm's overall performance is significantly improved by hardware acceleration. To use the GPU, special libraries and tools (NVidia's CUDA) must be installed. Unfortunately, the RPI4B GPU does not support these tools. Because the RPI4B GPU unit is not manufactured and supplied by NVidia. As a result, the CPU cores perform all YOLO algorithm computations. This reduces the robot's overall performance. As a result, a lite tiny version of the YOLO algorithm known as YOLO-Tiny was used. This version is faster on small embedded systems because it employs smaller detection models. Although the YOLO-Tiny is faster and requires less computational time than the traditional YOLO, the detection accuracy will be reduced.

4. *Depth Sensor Interface with ROS:* The depth data are not scaled to a metric unit during software implementation. This issue was caused by of the ROS messaging system architecture, which requires depth data from the camera node to be published as a one-dimensional vector. The length of the vector is 307200. As a result, a new library is required to convert the one-dimensional vector to a two-dimensional image. The 16-bit data must then be scaled in order to be converted to metric data. As a result, no object localization was performed in this thesis.

5. I^2C Protocol Configuration: This protocol allows multiple peripherals (referred to as slaves) to communicate with one or more controllers (referred to as masters). Each device works with this protocol has its own unique address. In this thesis, the I^2C serial protocol was used to interface the DC motor controller with the RPI4B through I^2C port 1. It should be noted that RPI4B contains two I^2C ports. In this work, the RPI4B is the master and the DC motor controller is the slave. During the implementation, a problem was risen from the fact that the address for the DC motor controller is $0x01$ and cannot be changed (static address) according to the datasheet [33]. At the same time, this address is reserved by the RPI4B and are not available for the user. Thus, it's not permissible to connect the DC motor controller to this address. Thus, a forcing technique must be performed in order to make this address available for the slave. A scan test for the I^2C port 1 is performed to show all the available addresses as follow.

   After connecting the DC motor controller to the I^2C port 1, a detection program (*i2cdetect*) was used to check if the controller is detected by the RPI4B or not on a specific port. The following command is used to scan the I^2C port 1: *i2cdetect –y 1*

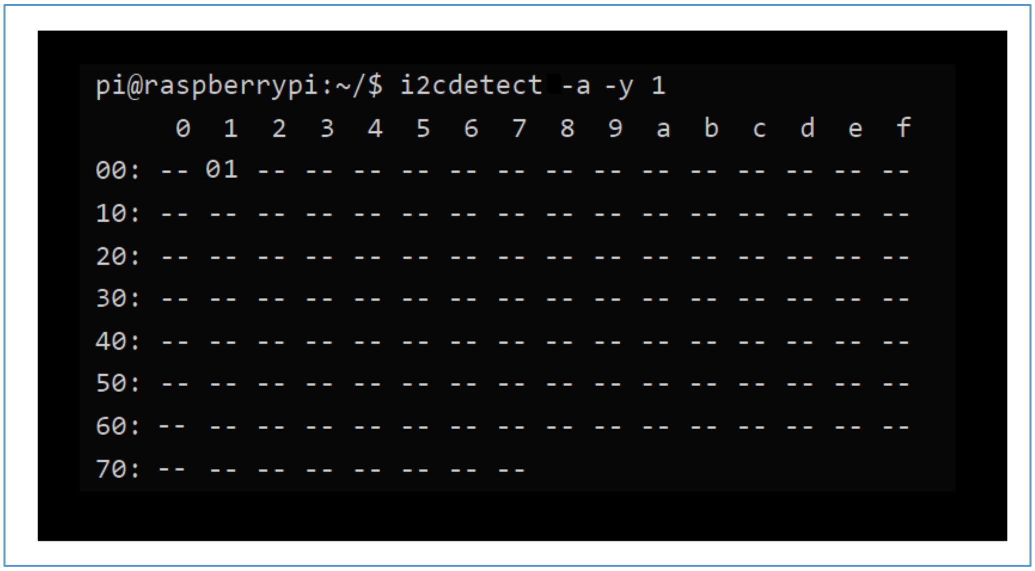
   The first parameter $(-y)$ is for disabling interactive mode. The second parameter (1) indicates that we are scanning for I^2C devices connected on I^2C port 1. This command will probe all the addresses on a I^2C port 1, and report whether any devices are presented or not. Figure 4.11 represents a RPI4B I^2C detection map for port 1. This map indicates that there are no peripherals detected on port 1.



```
pi@raspberrypi:~/$ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figure 4.11: I^2C Raspberry Pi4 detection map.

   However, the addresses $(0x00, 0x01$ & $0x02)$ are still unavailable for use, as shown in Figure 4.11. As a result, reconfiguring I^2C port 1 is required to make the address $0x01$ available to the user. As a result, the command *i2cdetect -a –y 1* shown in Figure 4.12 will force open all addresses and make them available.



```
pi@raspberrypi:~/$ i2cdetect -a -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -- 01 -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figure 4.12: ***I^2C detection map after opening $\mathbf{0x01}$ address.***

Finally, an RPI4B was chosen as the main computer to build and implement a mobile robot with ROS Noetic and a single board computer. The RPI4B's main operating system was Ubuntu Mate 20.04, which supported the ROS Noetic as the host OS. Furthermore, it supports the Intel camera SDK. If the slave peripheral addresses are among the reserved ones, the first three addresses in the I^2C port must be opened.

5

Results and Discussion

Throughout the thesis, a model for a mobile robot with a single computer board was discussed to details. Finally, it was proven that it's possible with mild modification to algorithms and hardware integration to use such SBC at a core of a multi-disciplinary robot. In the following sections the detection, classification and path planning results are presented.

5.1 Object Detection and Classifications using YOLO Algorithm

Figure 5.1 below show an example of an input image (left image), which represent an indoor scenery. The image is analyzed using the algorithm and generate the required data. Each object in the output image (right image) is bounded by a bounding box. A class identity (ID) and a Class name is given for each detected object according to the model matching probability. Furthermore, Figure 5.2 shows a detailed information about one of the detected objects (bottle). Such information's are:

- Class Name: "bottle"
- Class ID: 39
- Probability: 34.7%
- Bounding Box Dimension (in pixels):

$$X_{min} = 148, \;\; X_{max} = 196$$
$$Y_{min} = 317, \;\; Y_{max} = 451$$

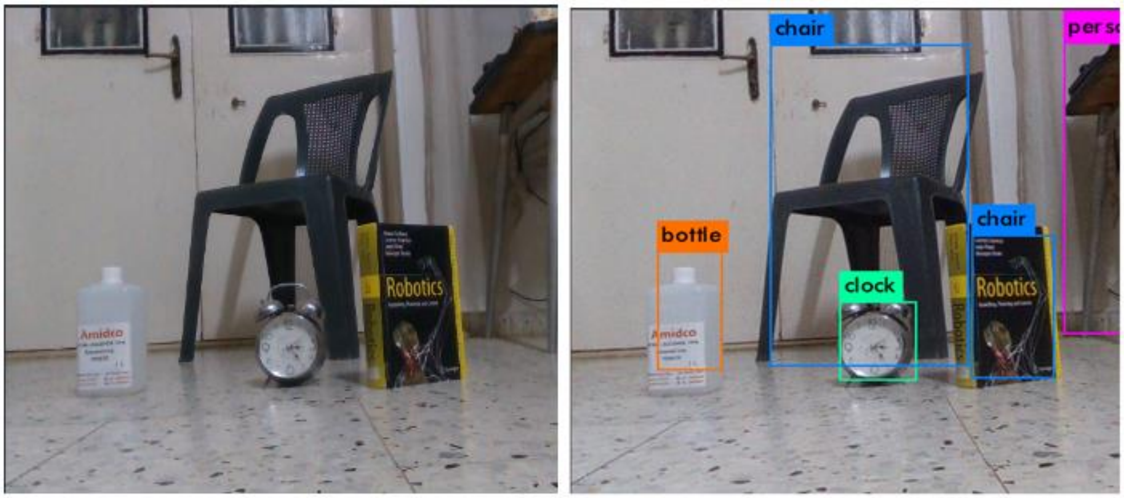
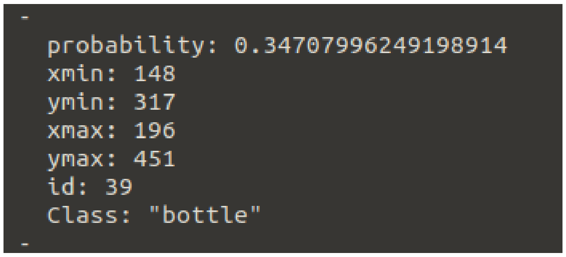


Figure 5.1: An input image (left) is applied to the YOLO algorithm and an output image (right) with each detected object.



```
-
  probability: 0.34707996249198914
  xmin: 148
  ymin: 317
  xmax: 196
  ymax: 451
  id: 39
  Class: "bottle"
-
```

Figure 5.2: Example of a generated data from YOLO algorithm for a detected object (Bottle).

In Figure 5.3 an outdoor scenery is applied to the algorithm which contains a tennis ball (sport ball). The algorithm is capable of detecting the tennis balls even within another objects. This scenery will be used as a validation example to test if the robot can avoid obstacles within the workspace. Figure 5.4 show a sample of the published data from the YOLO node to the topic *"Objects Names and Coordinates"*. For each detected object, several data are published such as bounding boxes coordinates ($x_{min}, y_{min}, x_{max}$ & y_{max}), class names, class ID and the class probability.

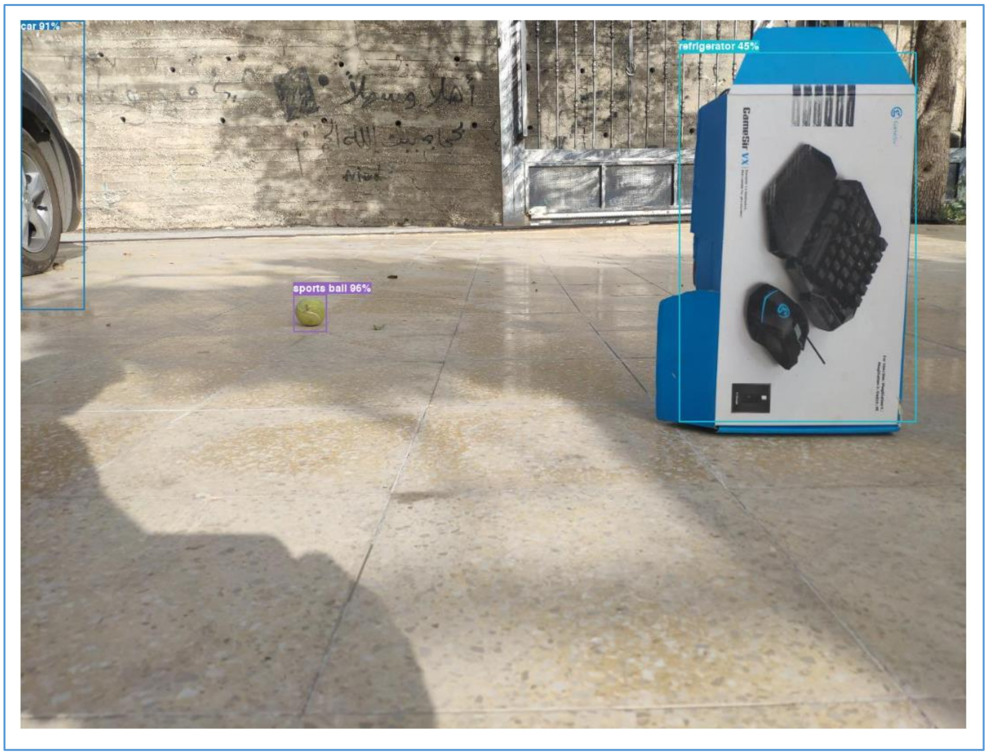
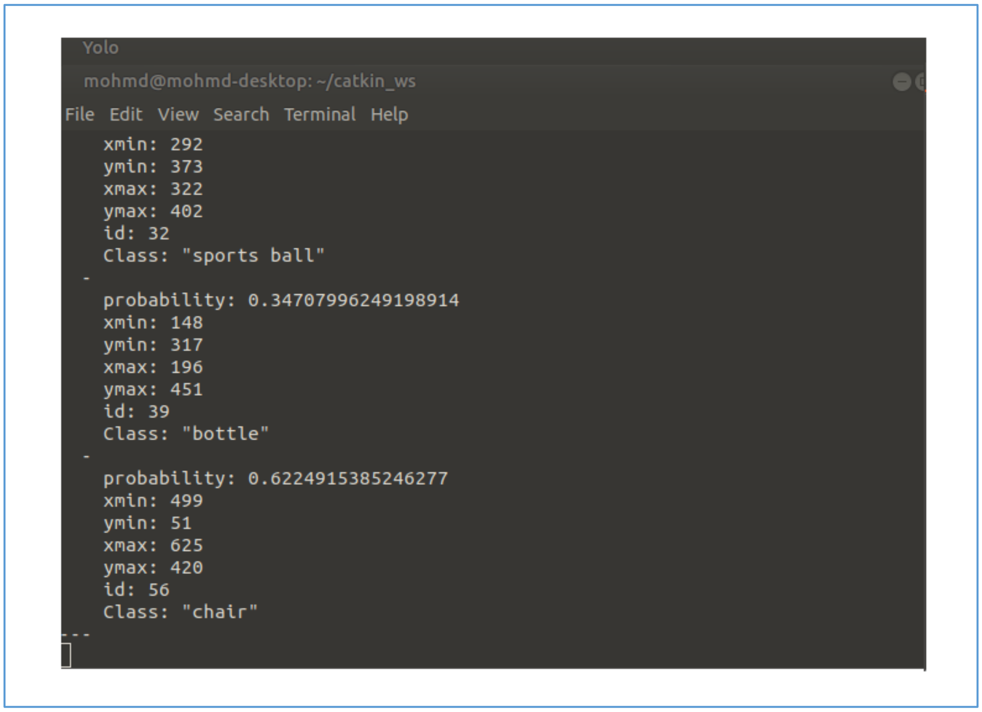


Figure 5.3: YOLO output image with a tennis ball detection.


Yolo
mohmd@mohmd-desktop: ~/catkin_ws
File Edit View Search Terminal Help

```
    xmin: 292
    ymin: 373
    xmax: 322
    ymax: 402
    id: 32
    Class: "sports ball"
  -
    probability: 0.34707996249198914
    xmin: 148
    ymin: 317
    xmax: 196
    ymax: 451
    id: 39
    Class: "bottle"
  -
    probability: 0.6224915385246277
    xmin: 499
    ymin: 51
    xmax: 625
    ymax: 420
    id: 56
    Class: "chair"
---
```

Figure 5.4: Detected objects classes and their coordinates in the image shown in figure 5.3.

The appeared uncertainty (probability) in detecting some objects changed due to some factors such as light conditions, camera projection angle and object orientation in the scenery. In addition to the usage of the YOLO-Tiny version which uses small detection models.

5.2 Depth Data

Figure 5.5 illustrate the depth data for the same scenery observed in Figure 5.1. The image data is transformed to distance data. The depth scales is shown to the right of the depth image. The measurements are tested with an empirically (using a meter) and verified with the measured ones.

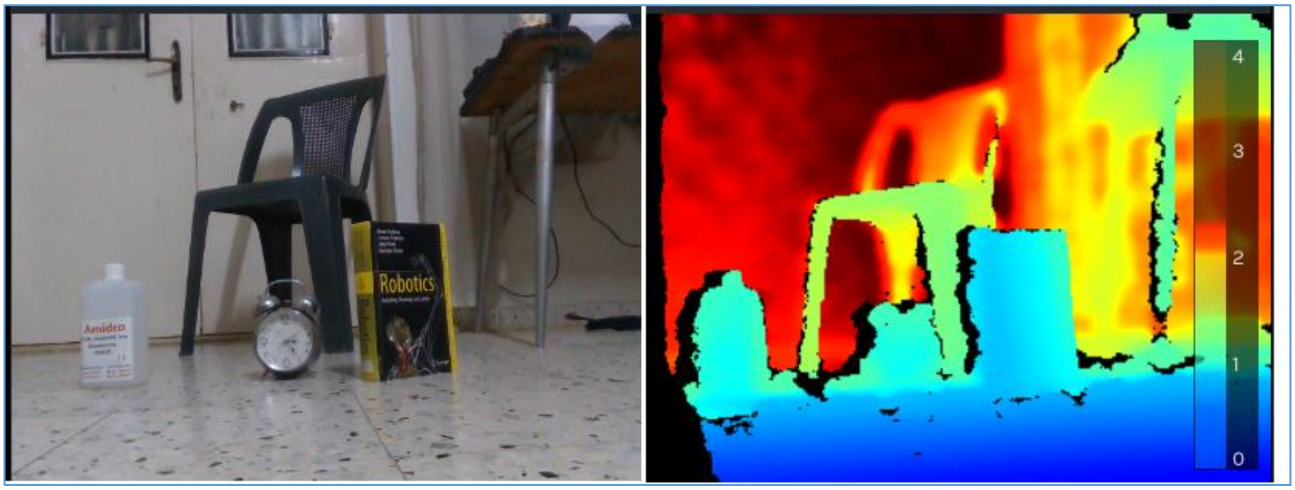


Figure 5.5: Depth Image (Right) in comparison with the original image (Left).

The red pixels represent the farthest distances from the depth sensor, while the blue pixels represent the shortest distances. The depth matrix has the size of **640 x 480**. Thus, it’s not possible to show it here.

5.3 Workspace Mapping

The objects in the scenery illustrated in Figure 5.3 are placed as shown in Figure 5.6. This figure is used just to demonstrate the objects coordinates compared to the robot position. Each cell has the dimension of **30cm x 30cm**. The empty cells are given a value of 1, while the obstacles cells are given a value of 1000. The unseen cells (Due to the camera field of view) are treated as obstacles cells. The start cell is the green cell with coordinates (30, 4). And the target cell (red) contains the tennis ball with coordinates (25, 2). Figure 5.7 illustrate the 2D map created by the mapping algorithm with the local costs and distance costs calculated by the ROS node. These data are obtained by the ROS nodes. Figure 5.8 illustrate the total cost matrix as obtained by the ROS node. The total cost matrix is obtained by formula (1.1). It's clear how the mapping algorithm gives the highest cost value to the detected obstacles. Furthermore, all the neighbor cells to the obstacle cell in addition to the hidden cells are also given the highest cost value.

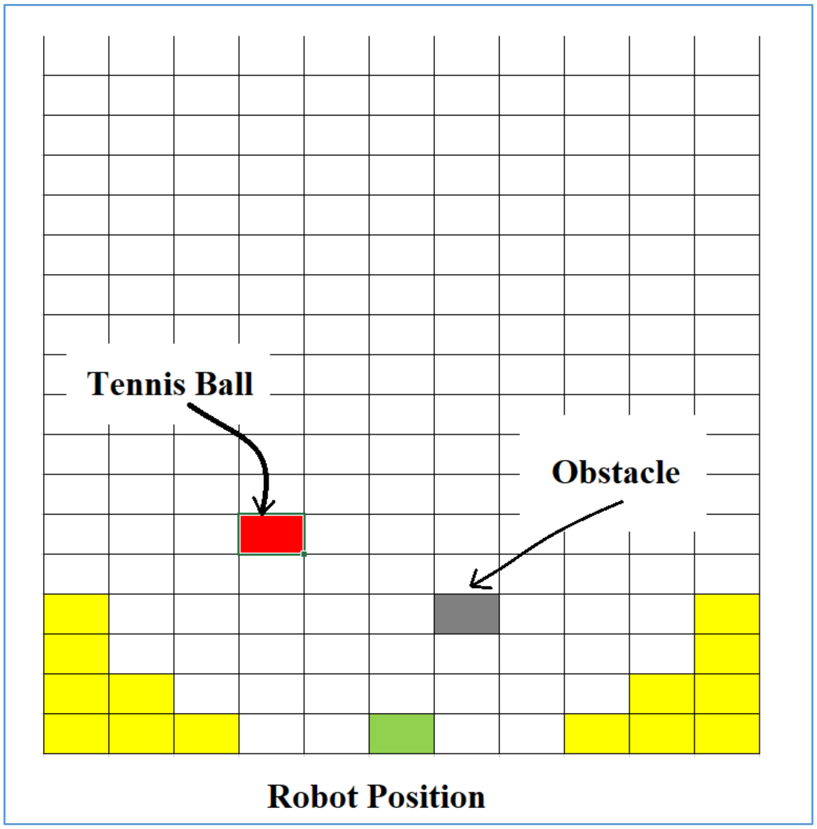
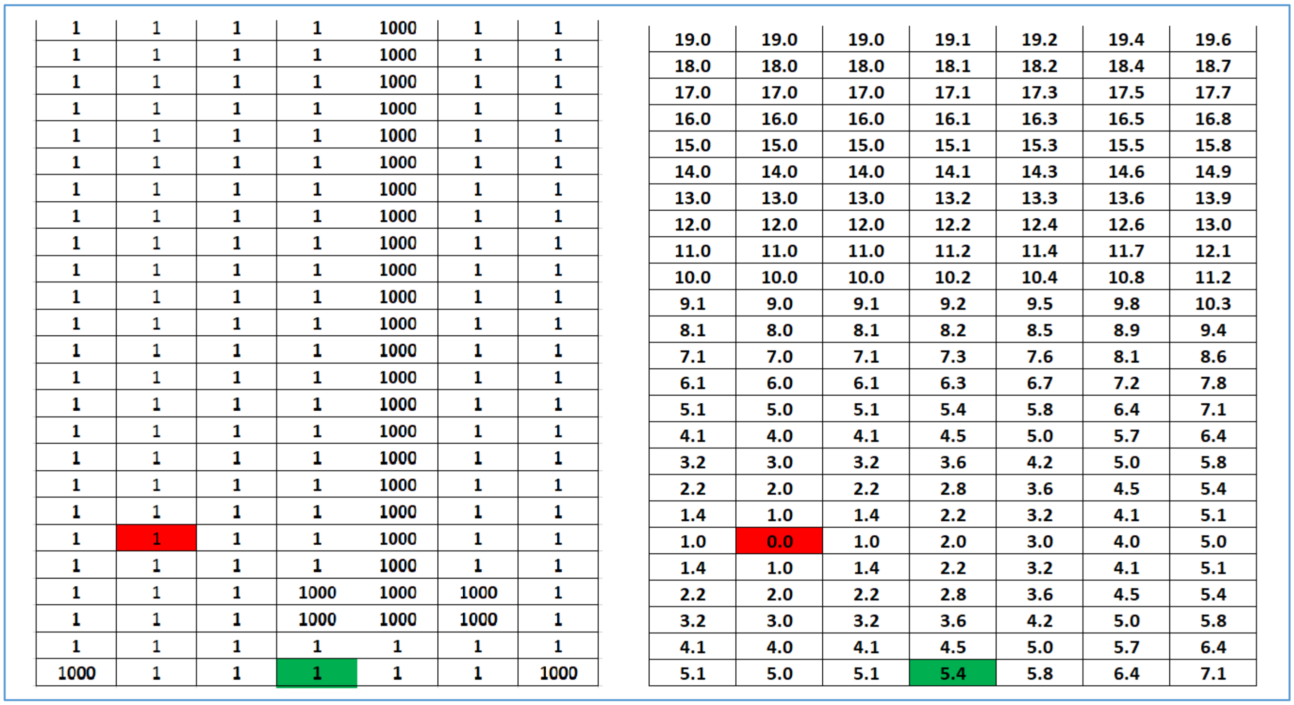


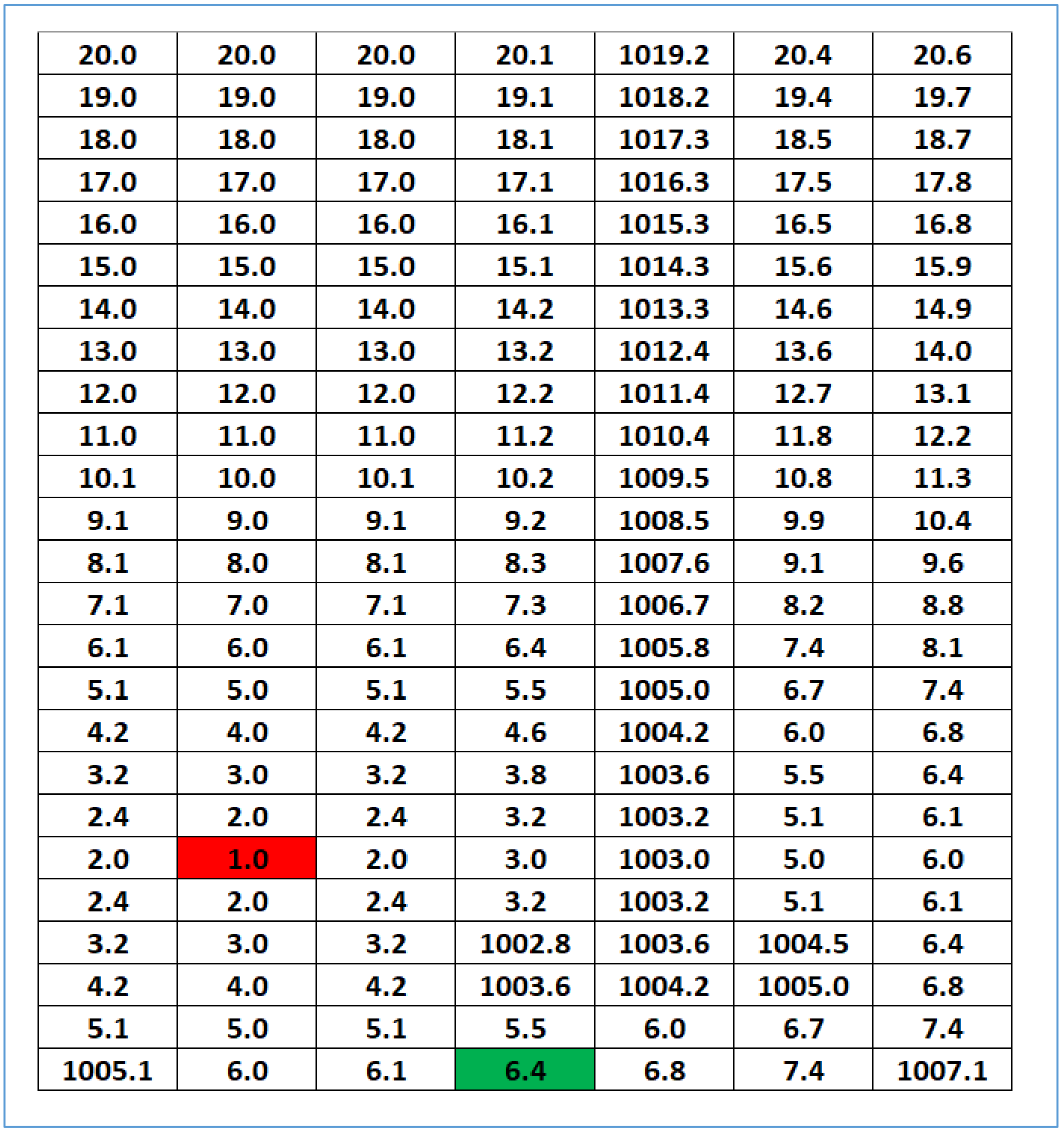
Figure 5.6: Illustration of objects location within the robot workspace.



1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1	1000	1	1
1	1	1	1000	1000	1000	1
1	1	1	1000	1000	1000	1
1	1	1	1	1	1	1
1000	1	1	1	1	1	1000

19.0	19.0	19.0	19.1	19.2	19.4	19.6
18.0	18.0	18.0	18.1	18.2	18.4	18.7
17.0	17.0	17.0	17.1	17.3	17.5	17.7
16.0	16.0	16.0	16.1	16.3	16.5	16.8
15.0	15.0	15.0	15.1	15.3	15.5	15.8
14.0	14.0	14.0	14.1	14.3	14.6	14.9
13.0	13.0	13.0	13.2	13.3	13.6	13.9
12.0	12.0	12.0	12.2	12.4	12.6	13.0
11.0	11.0	11.0	11.2	11.4	11.7	12.1
10.0	10.0	10.0	10.2	10.4	10.8	11.2
9.1	9.0	9.1	9.2	9.5	9.8	10.3
8.1	8.0	8.1	8.2	8.5	8.9	9.4
7.1	7.0	7.1	7.3	7.6	8.1	8.6
6.1	6.0	6.1	6.3	6.7	7.2	7.8
5.1	5.0	5.1	5.4	5.8	6.4	7.1
4.1	4.0	4.1	4.5	5.0	5.7	6.4
3.2	3.0	3.2	3.6	4.2	5.0	5.8
2.2	2.0	2.2	2.8	3.6	4.5	5.4
1.4	1.0	1.4	2.2	3.2	4.1	5.1
1.0	0.0	1.0	2.0	3.0	4.0	5.0
1.4	1.0	1.4	2.2	3.2	4.1	5.1
2.2	2.0	2.2	2.8	3.6	4.5	5.4
3.2	3.0	3.2	3.6	4.2	5.0	5.8
4.1	4.0	4.1	4.5	5.0	5.7	6.4
5.1	5.0	5.1	5.4	5.8	6.4	7.1

Figure 5.7: Local cost matrix (left) and the distance cost matrix (Right).

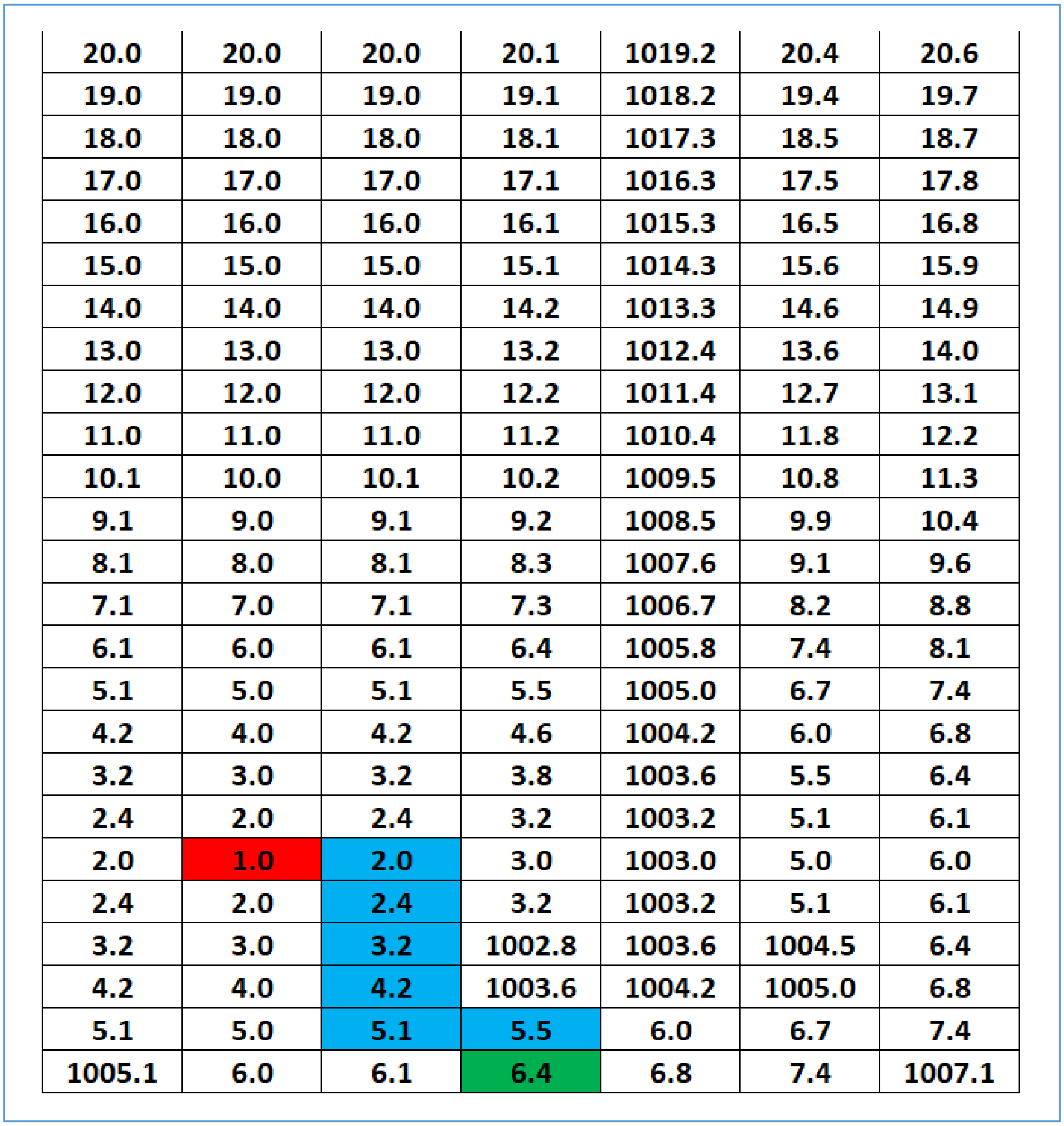


20.0	20.0	20.0	20.1	1019.2	20.4	20.6
19.0	19.0	19.0	19.1	1018.2	19.4	19.7
18.0	18.0	18.0	18.1	1017.3	18.5	18.7
17.0	17.0	17.0	17.1	1016.3	17.5	17.8
16.0	16.0	16.0	16.1	1015.3	16.5	16.8
15.0	15.0	15.0	15.1	1014.3	15.6	15.9
14.0	14.0	14.0	14.2	1013.3	14.6	14.9
13.0	13.0	13.0	13.2	1012.4	13.6	14.0
12.0	12.0	12.0	12.2	1011.4	12.7	13.1
11.0	11.0	11.0	11.2	1010.4	11.8	12.2
10.1	10.0	10.1	10.2	1009.5	10.8	11.3
9.1	9.0	9.1	9.2	1008.5	9.9	10.4
8.1	8.0	8.1	8.3	1007.6	9.1	9.6
7.1	7.0	7.1	7.3	1006.7	8.2	8.8
6.1	6.0	6.1	6.4	1005.8	7.4	8.1
5.1	5.0	5.1	5.5	1005.0	6.7	7.4
4.2	4.0	4.2	4.6	1004.2	6.0	6.8
3.2	3.0	3.2	3.8	1003.6	5.5	6.4
2.4	2.0	2.4	3.2	1003.2	5.1	6.1
2.0	1.0	2.0	3.0	1003.0	5.0	6.0
2.4	2.0	2.4	3.2	1003.2	5.1	6.1
3.2	3.0	3.2	1002.8	1003.6	1004.5	6.4
4.2	4.0	4.2	1003.6	1004.2	1005.0	6.8
5.1	5.0	5.1	5.5	6.0	6.7	7.4
1005.1	6.0	6.1	6.4	6.8	7.4	1007.1

Figure 5.8: Total cost matrix as calculated by the ROS node.

5.4 Path Finding with A* Search Algorithm

After the total cost matrix are obtained by formula (1.1). The A* algorithm searched for the optimal free-obstacles path which illustrated by the blue cells in Figure 5.9. It’s clear how the algorithm searched for the cells with the minimum cost, and avoiding all the cells with higher cost till it reached the target cell.



20.0	20.0	20.0	20.1	1019.2	20.4	20.6
19.0	19.0	19.0	19.1	1018.2	19.4	19.7
18.0	18.0	18.0	18.1	1017.3	18.5	18.7
17.0	17.0	17.0	17.1	1016.3	17.5	17.8
16.0	16.0	16.0	16.1	1015.3	16.5	16.8
15.0	15.0	15.0	15.1	1014.3	15.6	15.9
14.0	14.0	14.0	14.2	1013.3	14.6	14.9
13.0	13.0	13.0	13.2	1012.4	13.6	14.0
12.0	12.0	12.0	12.2	1011.4	12.7	13.1
11.0	11.0	11.0	11.2	1010.4	11.8	12.2
10.1	10.0	10.1	10.2	1009.5	10.8	11.3
9.1	9.0	9.1	9.2	1008.5	9.9	10.4
8.1	8.0	8.1	8.3	1007.6	9.1	9.6
7.1	7.0	7.1	7.3	1006.7	8.2	8.8
6.1	6.0	6.1	6.4	1005.8	7.4	8.1
5.1	5.0	5.1	5.5	1005.0	6.7	7.4
4.2	4.0	4.2	4.6	1004.2	6.0	6.8
3.2	3.0	3.2	3.8	1003.6	5.5	6.4
2.4	2.0	2.4	3.2	1003.2	5.1	6.1
2.0	1.0	2.0	3.0	1003.0	5.0	6.0
2.4	2.0	2.4	3.2	1003.2	5.1	6.1
3.2	3.0	3.2	1002.8	1003.6	1004.5	6.4
4.2	4.0	4.2	1003.6	1004.2	1005.0	6.8
5.1	5.0	5.1	5.5	6.0	6.7	7.4
1005.1	6.0	6.1	6.4	6.8	7.4	1007.1

Figure 5.9: Total cost matrix with the free-obstacles path (Blue cells).

5.5 Path Planning between Wye Points Through Cartesian Polynomials

The Figures bellow displays the interpolation between each two adjacent cells within the selected path. The obtained path consists from the following cells:

$$(30,4), (29,4), (29,3), (28,3), (27,3), (26,3), (25,3), (25,2).$$

These coordinates are converted to metric value for the interpolation. After obtaining the heading direction at each Wey point for each cell based on formula (3.20), the configuration for each cell is:

$$(\mathbf{30}, \mathbf{4}, \mathbf{90^{o}}), (\mathbf{29}, \mathbf{4}, \mathbf{180^{o}}), (\mathbf{29}, \mathbf{3}, \mathbf{90^{o}}), (\mathbf{28}, \mathbf{3}, \mathbf{90^{o}}), (\mathbf{27}, \mathbf{3}, \mathbf{90^{o}}), (\mathbf{26}, \mathbf{3}, \mathbf{90^{o}}), (\mathbf{25}, \mathbf{3}, \mathbf{180^{o}}), (\mathbf{25}, \mathbf{2}, \mathbf{180^{o}}).$$

Convert to metric unit:

$$(-\mathbf{9}, \mathbf{1.2}, \mathbf{90^{o}}), (-\mathbf{8.7}, \mathbf{1.2}, \mathbf{180^{o}}), (-\mathbf{8.7}, \mathbf{0.9}, \mathbf{90^{o}}), (-\mathbf{8.4}, \mathbf{0.9}, \mathbf{90^{o}}), (-\mathbf{8.1}, \mathbf{0.9}, \mathbf{90^{o}}),$$
$$(-\mathbf{7.8}, \mathbf{0.9}, \mathbf{90^{o}}), (-\mathbf{7.5}, \mathbf{0.9}, \mathbf{180^{o}}), (-\mathbf{7.8}, \mathbf{0.6}, \mathbf{180^{o}}).$$

The interpolations are as follow:

Between (30,4) & (29,4) are presented in Figure 5.10-a.
Between (29,4) & (29,3) are presented in Figure 5.10-b.
Between (29,3) & (28,3) are presented in Figure 5.10-c.
Between (28,3) & (27,3) are presented in Figure 5.10-d.
Between (27,3) & (26,3) are presented in Figure 5.10-e.
Between (26,3) & (25,3) are presented in Figure 5.10-f.
Between (25,3) & (25,2) are presented in Figure 5.10-g.

The total path arcing is illustrated in Figure 5.11. It’s clear how A* path is converted to an arc path with no discontinuities. Thus, the robot will make an arc path maneuvering in order to turn around sharp corner. The robot will not have to stop and rotate in order to change its course. The corner curvature related to the values of the constants k_i & k_f.

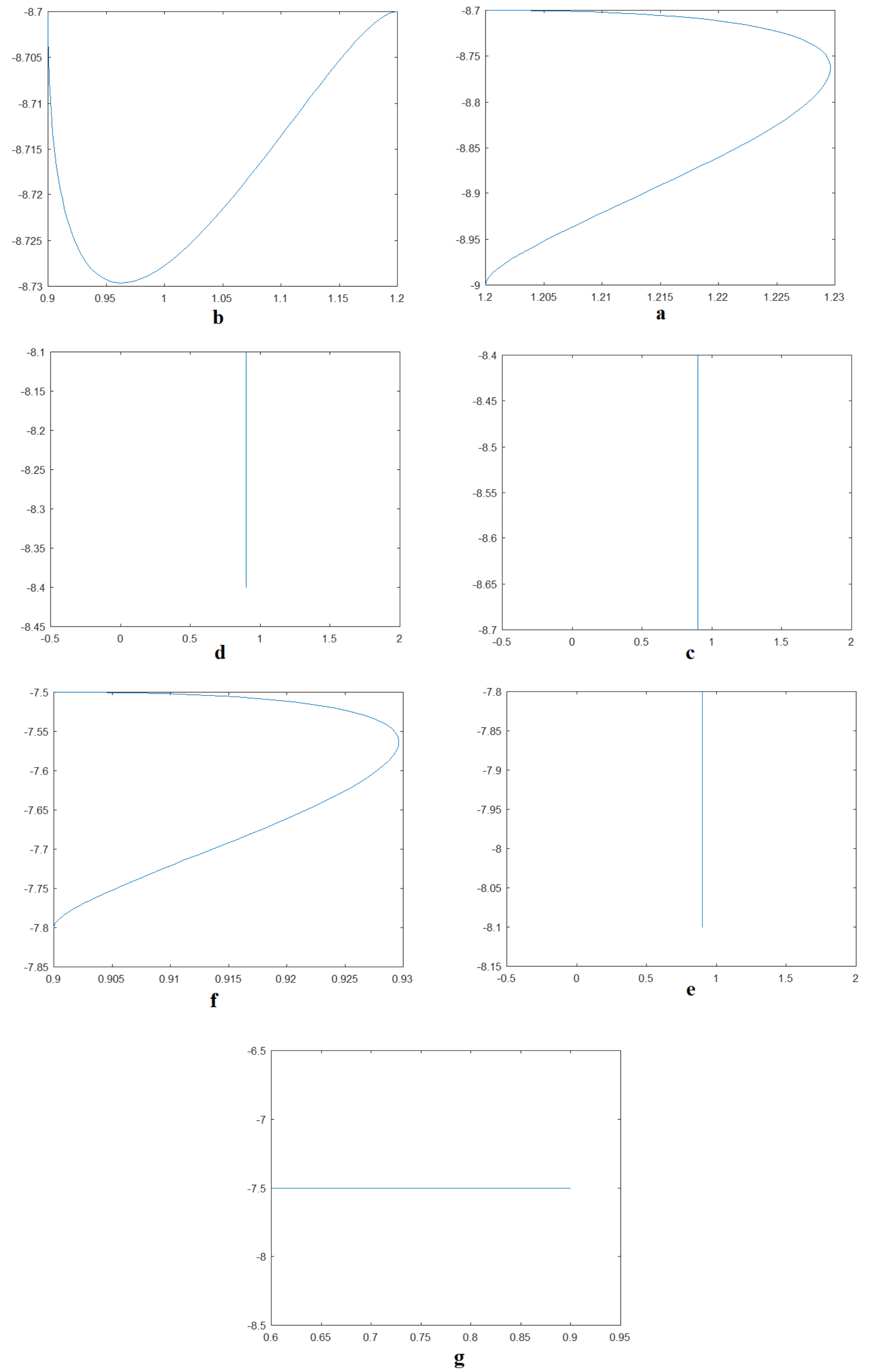


***Figure 5.10: Interpolation between each two adjacent cells.* $K_i = K_f = 0.2$.**

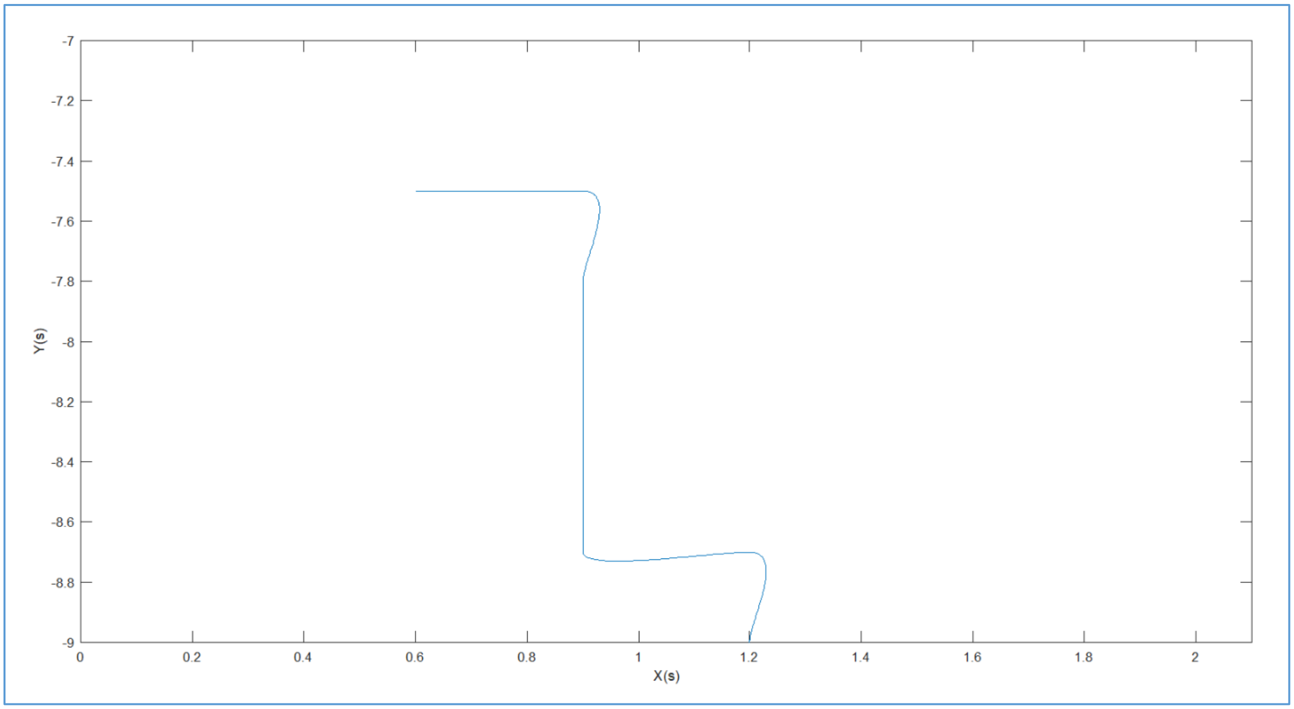


Figure 5.11: Total A* path after arcing.

5.6 Robot Trajectories

After generating the total path, the trajectories are obtained as illustrated in chapter 3. Figure 5.12 illustrate the total velocity profile along the path. The total travelling time along the path is determined according to the number of cells to reach the target cell. The travelling between two adjacent cells is designed to be 2 seconds. Thus, the travelling time is equal to the number of A* cells multiplied by 2. Accordingly, for the given path above. The total travelling time is 14 seconds. It should be noted that the robot velocity at each Wey point is choosen to be 0.05 m/s. and specified as a boundary condition. its clear how the velocity profile satisfy this boundary condition at each Wey point at time intervals : 2, 4, 6, 8, 10 & 12 sec. . Figure 5.13 & 5.14 represents the velocity for each wheel.

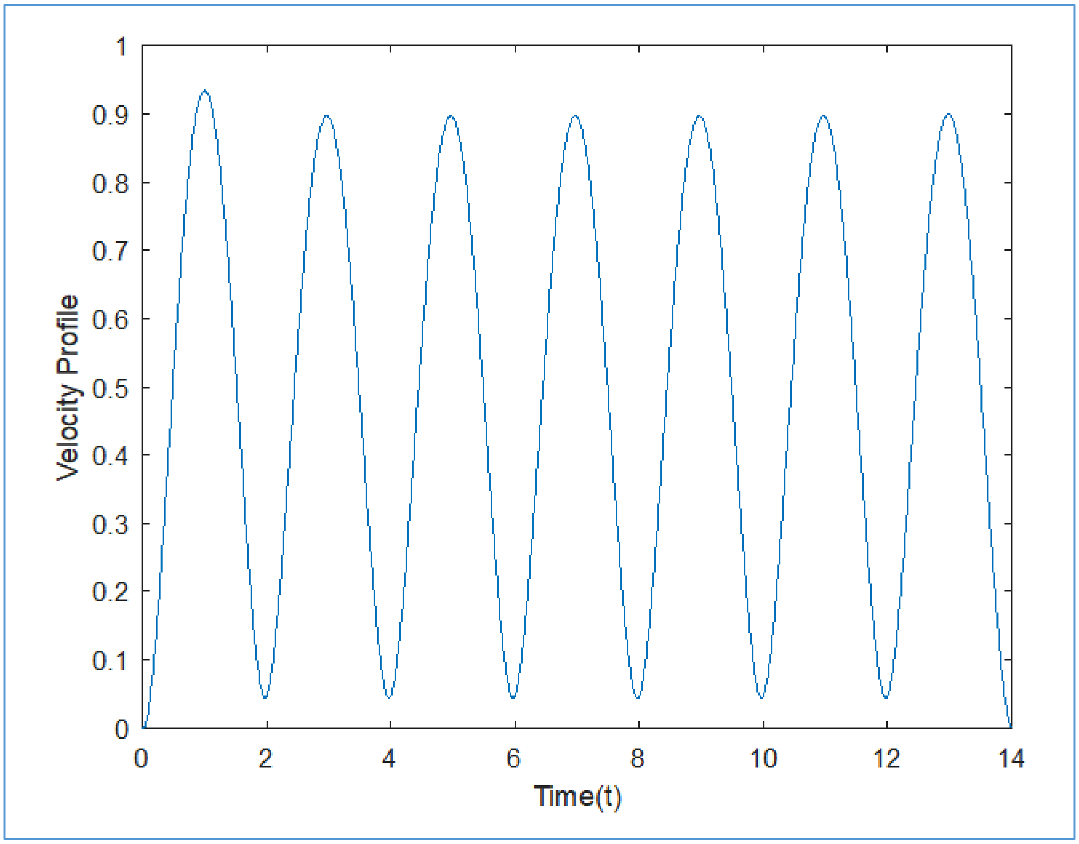


Figure 5.12: Total velocity profile.

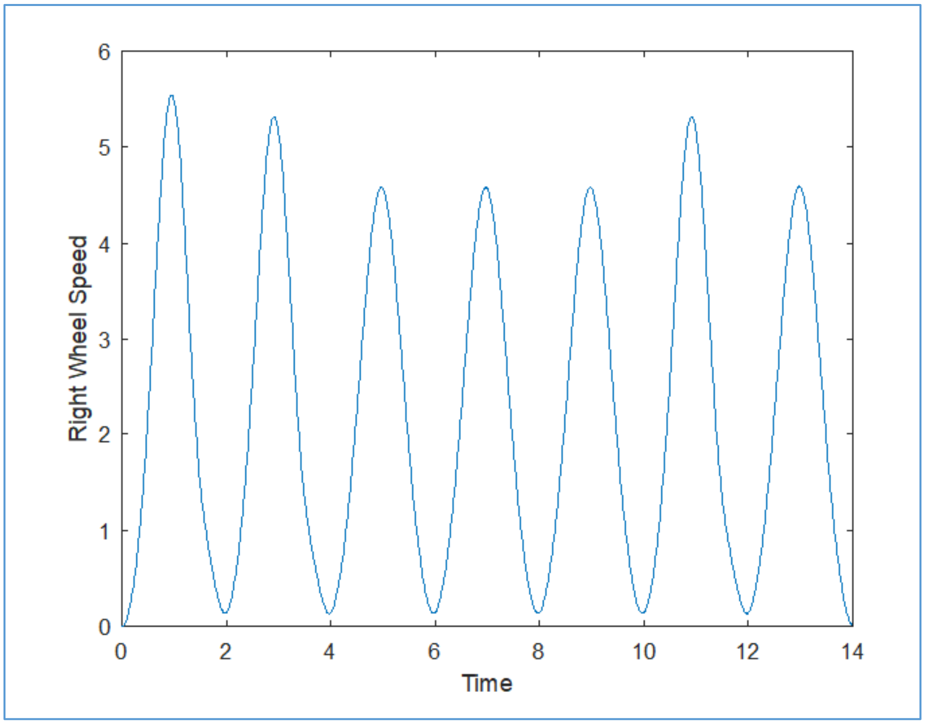


Figure 5.13: Right wheel angular velocity.

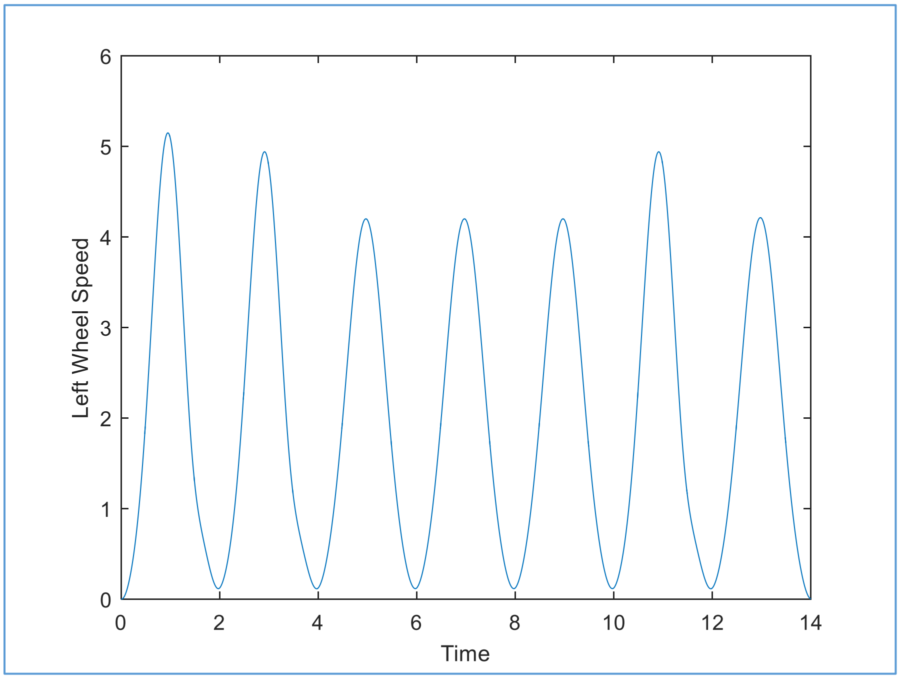


Figure 5.14: Left wheel angular velocity.

5.7 Motion Control

Figure 5.15 represent the transient response of a step input (RPM) of the closed loop system. The figure show that the OS% is 2%, this is due to the presence of the numerator in the closed loop transfer function (in equation 3.26). This increase in the OS% has no significant effect on the robot trajectory.

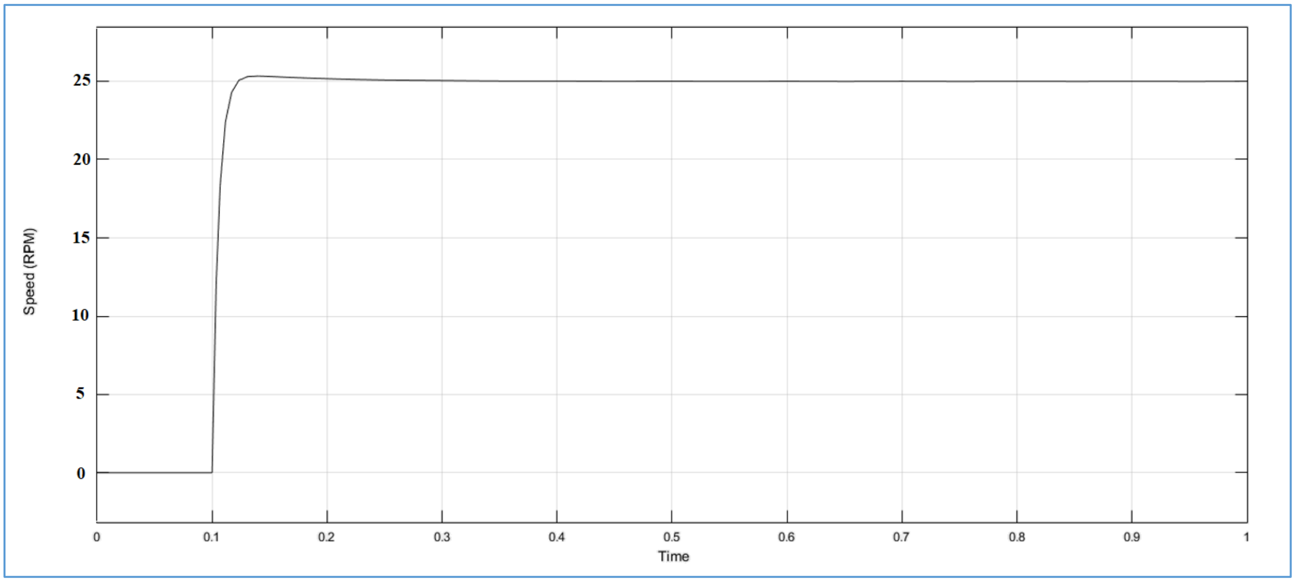


Figure 5.15: Step response of the closed loop system.

6

Conclusion and Future Research Topic

6.1 Conclusion

In this work, the proposed method of developing and implementing a standalone tennis collecting mobile robot using ROS platform was performed and deployed on a single board computer. The practical results show that the RPI4B is capable of performing all of the robot algorithms within acceptable constraints without the use of external resources. The robot is able to perceive its environment, detect objects and classify them, and generate the optimal obstacle-free path towards the target tennis ball.

However, many constraints and limitations appeared during the implementation process. For example, the RPI4B required a longer execution time than a standard PC to perform vision tasks. The extra time is due to the RPI4B hardware resource limitations, as well as the limitation on utilizing hardware acceleration abilities. In addition, there is incompatibility between the ROS software and the RPI4B OS, as well as between the ROS and other hardware's SDK. This problem was solved by installing Ubuntu Mate.

Furthermore, the A* algorithm was used to help the robot to find the shortest obstacle-free path by taking the distance as the optimization criteria. The RPI4B was able to execute the A* algorithm quickly, allowing the robot to determine the most optimal path to the target tennis ball. Also, the path obtained was curved to allow the robot to maneuver around corners while moving. Other algorithms were also in charge of determining the wheel's trajectory and control law. The algorithms were written in C++ language to ensure real-time data communication between the RPI4B and the low-level controller while also utilizing fewer computational resources. As an outcome, the robot could finish the tasks in a reasonable amount of time. Finally, it was demonstrated that the proposed low-cost solution was capable of running ROS-based mobile robot algorithms.

6.2 Future Work

The implementation of a ROS based mobile robot on a single board computer has been demonstrated to achieve a real low cost mobile robot. However, there are several significant areas to be explored in this work:

1. Minimizing the execution time imposed by the YOLO algorithm by exploiting the parallel execution capability of the ARM processor. To exploit this feature, the source code of the algorithm must be redesigned in way to support multithreading. This will allow the algorithm to be distributed on the four CPU cores. Which significantly improve the overall performance.

2. Minimizing the YOLO algorithm execution time using hardware acceleration. Which mean to exploit the graphical processing unit where the YOLO offloaded to GPU in order to accelerate it. This gives a more performance and efficiency than if it executed only on the CPU.

3. Minimizing the execution time for the whole system by overclocking the RPI4B CPU. This method requires the RPI4B to be efficiently equipped with an appropriate CPU cooler, in order to prevent overheating. The proposed method allow the user to overclock the CPU to 1.8 MHz safely.

4. Integration of neural convolutional network sticks with the raspberry pi.

5. Large implementation of workspace mapping theories using LIDAR and depth image sensors.

6. Add the feature of follow me mode, which allow the tennis robot to follow the player and collect the tennis balls which are detected in the player path.

References

[1] Chang, Y.H., Chung, P.L. and Lin, H.W., 2018, April. Deep learning for object identification in ROS-based mobile robots. In *2018 IEEE International Conference on Applied System Invention (ICASI)* (pp. 66-69). IEEE.

[2] Koledoye, M.A., De Martini, D., Carvani, M. and Facchinetti, T., 2017. Design of a mobile robot for air ducts exploration. *Robotics*, *6*(4), p.26.

[3] Gatesichapakorn, S., Takamatsu, J. and Ruchanurucks, M., 2019, January. ROS based autonomous mobile robot navigation using 2D LiDAR and RGB-D camera. In *2019 First international symposium on instrumentation, control, artificial intelligence, and robotics (ICA-SYMP)* (pp. 151-154). IEEE.

[4] Köseoğlu, M., Çelik, O.M. and Pektaş, Ö., 2017, September. Design of an autonomous mobile robot based on ROS. In *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)* (pp. 1-5). IEEE.

[5] Peng, J., Ye, H., He, Q., Qin, Y., Wan, Z. and Lu, J., 2021. Design of Smart Home Service Robot Based on ROS. *Mobile Information Systems*, *2021*.

[6] Priyandoko, G., Wei, C.K. and Achmad, M.S.H., 2018. Human following on ROS framework a mobile robot. *Sinergi*, *22*(2), pp.77-82.

[7] Hasegawa, K., Takasaki, K., Nishizawa, M., Ishikawa, R., Kawamura, K. and Togawa, N., 2019, December. Implementation of a ros-based autonomous vehicle on an fpga board. In *2019 International Conference on Field-Programmable Technology (ICFPT)* (pp. 457-460). IEEE.

[8] Abd Rahman, N.A., Sahari, K.S.M., Jalal, M.F.A., Rahman, A.A., Abd Adziz, M.I. and Hassan, M.Z., 2020, April. Mobile robot for radiation mapping in indoor environment. In *IOP Conference Series: Materials Science and Engineering* (Vol. 785, No. 1, p. 012021). IOP Publishing.

[9] Maiti, T.K., 2021, March. ROS on arm processor embedded with FPGA for improvement of robotic computing. In *2021 International Symposium on Devices, Circuits and Systems (ISDCS)* (pp. 1-4). IEEE.

[10] Dong, M., Chen, D., Ye, N. and Chen, X., 2021, March. Design of Ackerman Mobile Robot System Based on ROS and Lidar. In *Journal of Physics: Conference Series* (Vol. 1838, No. 1, p. 012073). IOP Publishing.

[11] Aagela, H., Al-Nesf, M. and Holmes, V., 2017, September. An Asus_xtion_probased indoor MAPPING using a Raspberry Pi with Turtlebot robot Turtlebot robot. In *2017 23rd International Conference on Automation and Computing (ICAC)* (pp. 1-5). IEEE.

[12] Sankar, S. and Tsai, C.Y., 2019. ROS-based human detection and tracking from a wireless controlled mobile robot using kinect. *Applied System Innovation*, *2*(1), p.5.

[13] Rivai, M., Hutabarat, D. and Nafis, Z.M.J., 2020. 2D mapping using Omni-directional mobile robot equipped with LiDAR. *Telkomnika*, *18*(3), pp.1467-1474.

[14] Zhi, L. and Xuesong, M., 2018, October. Navigation and control system of mobile robot based on ROS. In *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)* (pp. 368-372). IEEE.

[15] Chen, H.K. and Dai, J.M., 2016, July. An intelligent tennis ball collecting vehicle using smart phone touch-based interface. In *2016 International Symposium on Computer, Consumer and Control (IS3C)* (pp. 362-365). IEEE.

[16] Perera, D.M., Menaka, G.M.D., Surasinghe, W.V.K.M., Madusanka, D.K. and Lalitharathne, T.D., 2019, December. Development of a Vision Aided Automated Ball Retrieving Robot for Tennis Training Sessions. In *2019 14th Conference on Industrial and Information Systems (ICIIS)* (pp. 378-383). IEEE.

[17] Nie, H., Cai, G., Liu, B., Hu, X., Yang, F., Ni, J. and Hou, X., 2019, September. On Development of An Autonomous Ball Collecting Wheeled Mobile Robot. In *2019 3rd Conference on Vehicle Control and Intelligence (CVCI)* (pp. 1-5). IEEE.

[18] Quigley, M., Gerkey, B., & Smart, W. D. (2016). *Programming Robots with ROS: A Practical Introduction to the Robot Operating System* (1st ed.). Pp 3-7. O'Reilly Media.

[19] *ROS/Tutorials - ROS Wiki*. (2021). Ros.Org. http://wiki.ros.org/ROS/Tutorials

[20] Fairchild, C. and Harman, T.L., 2017. *ROS Robotics By Example: Learning to control wheeled, limbed, and flying robots using ROS Kinetic Kame*. Packt Publishing Ltd.

[21] Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2008). *Robotics: Modelling, Planning and Control (Advanced Textbooks in Control and Signal Processing)* (1st ed. 2009 ed.). Springer. Pp 607-608 & Pp 489-493.

[22] Liu, X. and Gong, D., 2011, April. A comparative study of A-star algorithms for search and rescue in perfect maze. In *2011 international conference on electric information and control engineering* (pp. 24-27). IEEE.

[23] Girshick, R., Donahue, J., Darrell, T. and Malik, J., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 580-587).

[24] Girshick, R., 2015. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision* (pp. 1440-1448).

[25] Ren, S., He, K., Girshick, R. and Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, *28*.

[26] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).

[27] *Introduction to yolo algorithm for object detection*. Section. (n.d.). Retrieved January 3, 2022, from https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/

[28] Sharbati, Abu Omar & Sultan. (2018). "Tennis-Balls Collecting Robot". Palestine Polytechnic University.

[29] Nise, N. S. (2010). *Control Systems Engineering* (6th ed.). Wiley.

[30] Intel, "Intel RealSense D400 Series Product Family" , Revision 005, January 2019.

[31] *Raspberry Pi Documentation*. (2022). Raspberry Pi. https://www.raspberrypi.com/documentation/

[32] *Ubuntu MATE for Raspberry Pi*. (n.d.). Ubuntu Mate. Retrieved 2022, from https://ubuntu mate.org/raspberry-pi/

[33] HiTechnic, "HiTechnic FIRST Motor Controller Specification", Specification Rev1.4, 2008.

[34] Fan, X. (2015). *Real-Time Embedded Systems: Design Principles and Engineering Practices* (1st ed.). Newnes.

[35] *ROS.org*. (n.d.). Wiki ROS. Retrieved May 1, 2021, from http://wiki.ros.org/noetic/Installation/Ubuntu

[36] *Git Hub*. (n.d.). Intel Realsense-ROS. Retrieved July 6, 2021, from https://github.com/

[37] *Git Hub*. (n.d.). Darknet-ROS. Retrieved April 10, 2021, from https://github.com/leggedrobotics/darknet_ros

[38] Team, U. M. A. T. E. (n.d.). *Raspberry Pi*. Ubuntu MATE. Retrieved June 7, 2021, from https://ubuntu-mate.org/raspberry.