Path Planning and Control of a Differential Drive Using Computer Vision

By:

PARAS SAVNANI (16BME117)

MECHANICAL ENGINEERING DEPARTMENT
INSTITUTE OF TECHNOLOGY
NIRMA UNIVERSITY
AHMEDABAD -382481
MAY 2020

Path Planning and Control of a Differential Drive Using Computer Vision

Submitted in partial fulfillment of the requirements for the award of degree of Bachelor of Technology Mechanical Engineering

BACHELOR OF TECHNOLOGY
IN
MECHANICAL ENGINEERING

Submitted by:

PARAS SAVNANI (16BME117)

Guided by**:**

Prof. Shruti Bhatt (Department Guide)
Mr. Arjun Bhasin (Industry Guide)

MECHANICAL ENGINEERING DEPARTMENT
INSTITUTE OF TECHNOLOGY
NIRMA UNIVERSITY
AHMEDABAD -382481
YEAR: 2019-2020

Declaration

This is to certify that

- The thesis comprises my original work towards the degree of Bachelor of Technology in Mechanical Engineering at Nirma University and has not been submitted elsewhere for degree.
- Due acknowledgement has been made in the text to all other material used.

Sign.
Paras Savnani
(16BME117)

Undertaking for Originality of the Work

I, Paras Savnani (16BME117) undertake that the Major Project entitled ("Path Planning and Control Of a Differential Drive Using Computer Vision") submitted by me, towards the partial fulfillment of the requirements for the degree of Bachelor of Technology in Mechanical Engineering of Nirma University, Ahmedabad, is the original work carried out by me. I give assurance that no attempt of plagiarism has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

Signature of Student

Date:
Place: Ahmedabad

Endorsed by
(Signature of Guide)

Certificate

TO WHOMSOEVER IT MAY CONCERN

This is to certify that, Mr. Paras Savnani, student of Mechanical engineering, 8th Semester of, Institute of Technology, Nirma University has satisfactorily completed the project report titled “Path planning and Control Of a Differential Drive Using Computer Vision”.

Date:

Department Guide
Prof. Shruti Bhatt,
Assistant Professor,
Department of Mechanical Engineering,
Institute of Technology
Nirma University, Ahmedabad

Industry Guide
Mr. Arjun Bhasin,
Chief Technical Officer (CTO),
Fero.ai, Ahmedabad/Dubai

Dr. V. J. Lakhera
Head and Professor,
Department of Mechanical Engineering,
Institute of Technology
Nirma University, Ahmedabad

Approval Sheet

The Project entitled **Path planning and Control of a Differential Drive Robot Using Computer Vision** by **Paras Savnani (16BME117)** is approved for the degree of Bachelor of Technology in Mechanical Engineering.

Examiners

____

____

____

Date: ____________

Place: ___________

Acknowledgments

It is my pleasure to express my sincere gratitude to people who were always ready to help whenever needed, and guide throughout my project. I thank everyone who have directly or indirectly provided their valuable time and guidance, without whom this project wouldn’t have been possible.

I would like to thank my project guide **Prof. Shruti Bhatt** (Assistant Professor, Mechanical Engineering Department, Institute of Technology, Nirma University) for her precious advice and help in determining objectives and for guidance throughout the project. I also thank **Mr. Arjun Bhasin** for helping me with his industrial expertise and assistance. I am very thankful to **Fero.ai**, for providing and allowing to use resources required for this project and motivating students like me to conduct such projects.

I appreciatively acknowledge contribution of all writers, researchers, makers, programmers, developers, bloggers who helped to make this project a success. A special thanks to **Mr. Adrian Rosebrock** (Writer at “pyimagesearch”) for his priceless articles and tutorials, which helped me to understand concepts of Image processing and Computer Vision in Python. Also, I would like to thank **Coursera** for providing me free online courses on motion planning and path planning.

Sign. __________
Paras Savnani
(16BME117)

Date:
Place: Ahmedabad

Abstract

Path planning is the study of generating the shortest path between the source and the destination. It is a subset of motion planning study which is a term used in robotics to find a sequence of valid configurations that moves a robot from source to destination. Further, Computer Vision is the study of cameras, the processing of images and videography to extract useful information from the environment. The project will cover all the aspects from the design stage to the implementation stage and also it is a combination of interdisciplinary subjects such as design, electronics and programming.

This project aims to combine the image processing with path planning techniques to extract an obstacle course using a camera and process the image to segment obstacles. Then, we test various path planning algorithms to get the shortest path. Once the path is generated, the path is smoothened using spline interpolation. Further, the differential drive motion planning equations are used to navigate the drive on the path and appropriate control algorithm is applied to mitigate the errors.

Objective of this project is not only to successfully track but to enhance tracking ability of the system by improving algorithms and control system. Efforts will be towards achieving tracking at higher framerate and at higher resolution then currently achieved by some people. During this project the obstacle detection part will be carried out using several Computer Vision algorithms and image processing techniques. The developed system has many applications in advance systems including defense, surveillance, monitoring, humanoid robots, etc. Also a method to evaluate performance of the system has to be developed. The scope of the present work involves the conceptual design and assembly of the tank drive robot, its wireless control and a python “Pygame” simulation of the robot navigating different paths on different obstacle courses.

Table of Contents

List of Figures

List of tables

Chapter 1
Introduction

1.1 Company Profile

Fero aims to spearhead a new generation of technological transformation by shaping the future of digital freight with the use of revolutionary automation systems such as narrow Artificial Intelligence, Machine Learning, Speech Recognition, Computer Vision & Blockchain to eliminate the need for human interaction in the logistics industry worldwide. Fero has created the world's 1st Ai voice agent that will replace monotonous human interaction and optimize transactions in the freight and logistics. As of now, Fero is researching mainly in computer vision technologies, robotics and path planning techniques. For more information, visit: www.fero.ai.

Mission:
Optimization and automation of freight transactions globally

Vision:
Bring unification and Artificial Intelligence enabled coordination to the global freight ecosystem

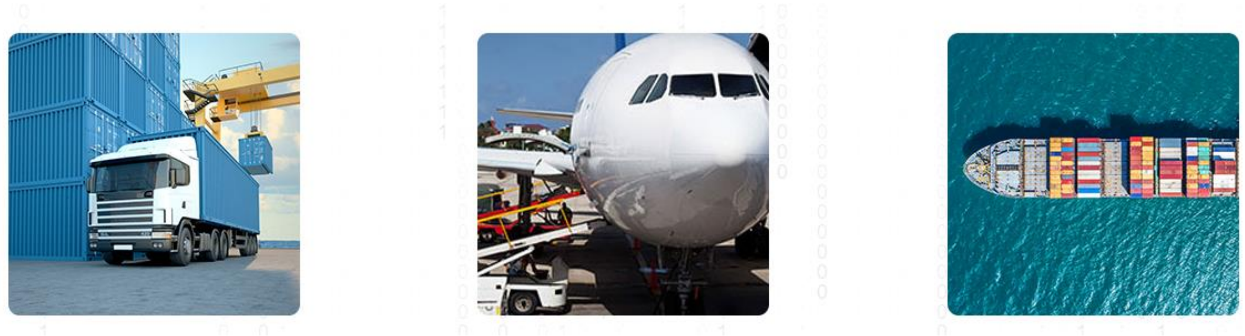


Figure1.1: Fero's Scope of Work

This company is a logistics based company and has mainly two departments i.e IT and CV (Computer Vision). Currently, there is a lot of research going on in the computer vision department and this project is also allocated under that department. It includes the application of computer vision and motion planning. As per company's scope, it is a research project which would be applied to real world scenarios in warehouse management, freight logistics, freight route optimization and shipment tracking etc.

1.2 Motion planning

Motion Planning is a term used in robotics to define a class of problems in which the robot has to reach the goal configuration from the start configuration within a stipulated time period. It is also known as the piano mover's problem/ navigation problem. It is a widely researched field in robotics. From mobile robots to aerial robotics to industrial manipulators, motion planning is an essential problem to tackle. It also includes the subfield of path planning, which helps us to find an optimal path between the source and the destination. Apart from robotics, this field has widespread applications in industry layout planning, trajectory planning, google maps etc.

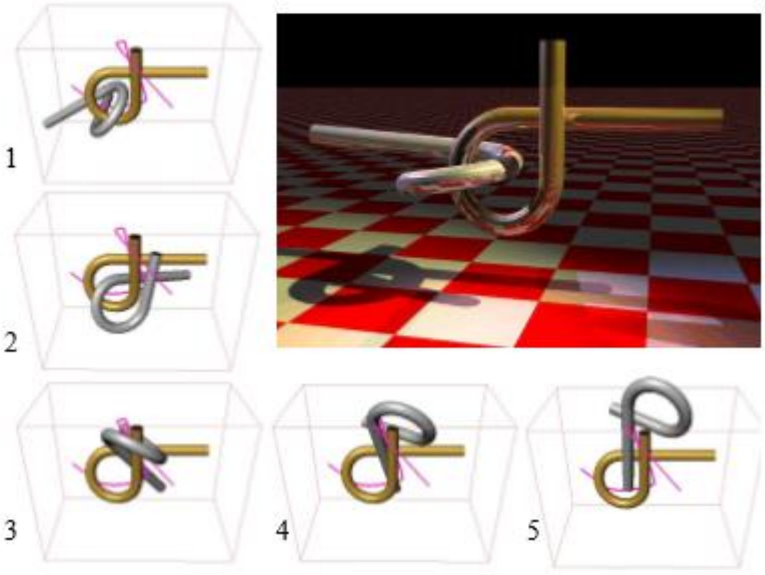


Figure 1.2: Alpha 1 puzzle [1]

In this report we evaluate the different techniques for path planning and apply it to the robot navigation problem. Path planning mainly depends upon how well a graph is generated and searched. For graph generation we have very good techniques like visibility graph, voronoi diagram, cell decomposition etc. and for graph search, algorithms like Dijkstra's, A* etc. provide much faster results. Apart from these standard techniques we have other probabilistic techniques too like (Randomly-Exploring Rapid Trees) RRT, Probabilistic road map, etc. These techniques are relatively simple to apply but they have a major drawback i.e they are not complete, which means that there is no guarantee that we will find the solutions always. The reliability of the solution will depend upon the number of iterations, thus they are known as probabilistic techniques.

1.3 Computer Vision (CV)

We are using computer vision in our day to day life, in our mobile phones applications, in social media, movies, etc. It also has wide application in medical, industries, video editing, etc. Computer vision is use of computing to manipulate the image. Image can be either previously taken, or a video that is an array of image or a live video stream from camera.

Definition of an Image:

- An image is a two-dimensional function g(x,y), where x and y are the planar coordinates, and the intensity of the image at that level is the amplitude of g.

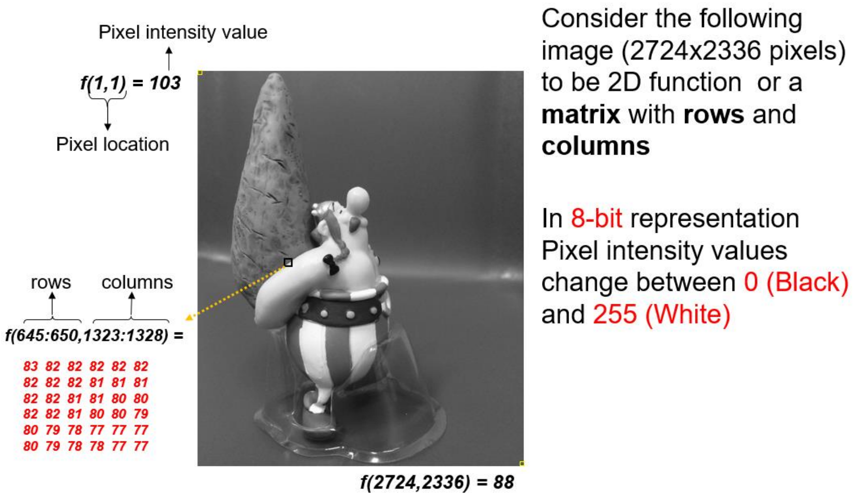
- If (x, y) and the amplitude values of g are finite and discrete quantities, we call the image a digital image. A digital image has a finite number of small elements known as pixels, each of which has a particular location and value in the image.

Image processing:

- Image processing are a set of techniques to perform operations on an image, in order to extract some useful and reliable information from it. We can also enhance an image using these techniques to ascertain distorted features. It is a type of signal processing in which input is an image and output may be image or characteristics/features related with that image.

- Image processing consists of following three steps:

1. Using image acquisition tools to import the image
2. Analyzing and manipulating the image
3. Outputting the resultant altered image or report based on the image analysis.



Consider the following image (2724x2336 pixels) to be 2D function or a **matrix** with **rows** and **columns**

In 8-bit representation Pixel intensity values change between 0 (Black) and 255 (White)

Figure 1.3: Pixel Representation on a grayscale image [2]

There are much more complex functions and algorithms available, especially when computer vision is paired with Machine Learning (ML) and Deep Learning (DL) it gives some awesome results. Lots of tech-giants are working on fusing these amazing technologies including, Google, Apple, Tesla, etc. Computer vision has huge influence in self-driving cars, there is real time environment tracking based on which computers can take decisions. It is fascinating how machines are outperforming the humans. With computer vision and machine learning a car can learn how to drive: in just 3 hours.

CV prepares a data set which acts as input to ML. image received from camera cannot directly be used in ML models, it first needs to be filtered and remove noise. There are lots of ML algorithms that can be applied to in this project, once a physical mechanism is made then it is just software work. In fact, “Haar cascade” is an ML algorithm as explained in detail later. There are algorithms available such as “Sort” and “Deep Sort”, that are based on Global data association and applies deep neuro networks to do association. However, that is out of scope of this project. At the moment a trained model for only “Pedestrian detection” is available, also method for training these models are not published yet. Therefore, it may be a future project.

1.4 Applications

- Mobile Robots
- Military and defense
- Surveillance
- Monitoring
- Baby monitoring and pet monitoring.
- Autonomous vehicles
- Research
  and anywhere where tracking is required.

1.5 Objectives

This project mainly focusses on following;

- Design and Manufacture of the differential drive robot and Selection and integration of hardware.
- Obstacle detection using Computer Vision techniques using an overhead Camera (Visual Surveying).
- Application and optimization of Path Planning Algorithm and Simulation of Robot Control

Chapter 2
Literature review

This chapter discuss work that has been carried out in this field, and how it is applied. There is lots of research being done in the field of computer vision, for various types of detections, tracking, association, recognition, etc. including a steady camera. Also, the field on path planning has a lot of ongoing research to improve the path searching algorithms. The algorithms like visibility graph, Voronoi diagram, Artificial potential fields, bug algorithms have been explored. Following literature review gives a clear insight into latest technological trends and background research behind this project.

2.1 Graph-based Path Planning for Mobile Robots

David T. Wooden (2006): The process of path planning is to decide how to get from one point to another when one does not know what will be along the way. It is a rudimentary problem of robotics. In this paper this problem is discussed, narrowly at the level of dynamic path planning in the plane, as well as at a higher-level, over colored graphs, and mostly within the context of the robot control architecture.

The roadmap construction with a visibility graph based approach for unstructured polygonal environments is known as the oriented visibility graph approach. The ideal quality of this construction derives from the fact that a fixed goal point is insisted upon and allowed for possibly suboptimal paths in the output roadmap. The actual world experiments show the advantages of the suggested method in time-critical outdoor applications where the perception is based solely on stereo-based elevation maps. These maps are converted into the polygons in order to support the use of the planning algorithm. Also, by saving the graph between runs, dynamic updating rules (for addition, removal, or changing polygons) enable the robot to enhance its performance over multiple runs.

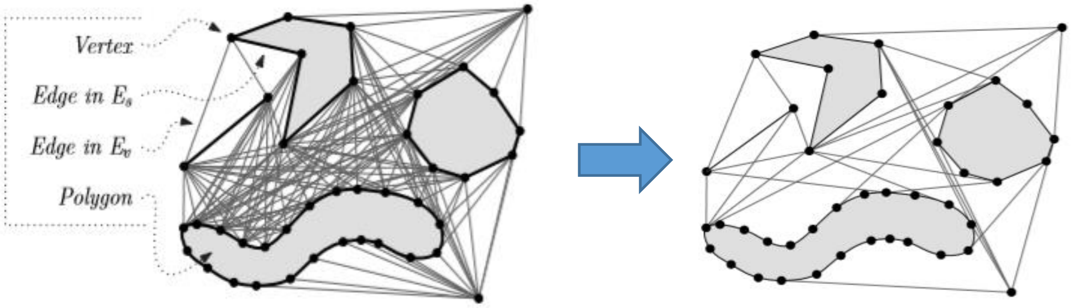


Figure 2.1: Visibility graph generation and reduction [3]

2.2 A Comparative Study of Bug Algorithms for Robot Navigation

This research paper shows a comparative study of the Bug Algorithms, with the goal of comparing them for navigation in robotics. A comparison of selection of different Bug Algorithms in a simulated robot environment where they experience the different types of noise and cases of failure of their on-board sensors. From the software simulation results, one can conclude that the implemented Bug Algorithm's performances are very sensitive to certain types of sensor-noise, which was mostly noticeable when the odometry-drift occurs.

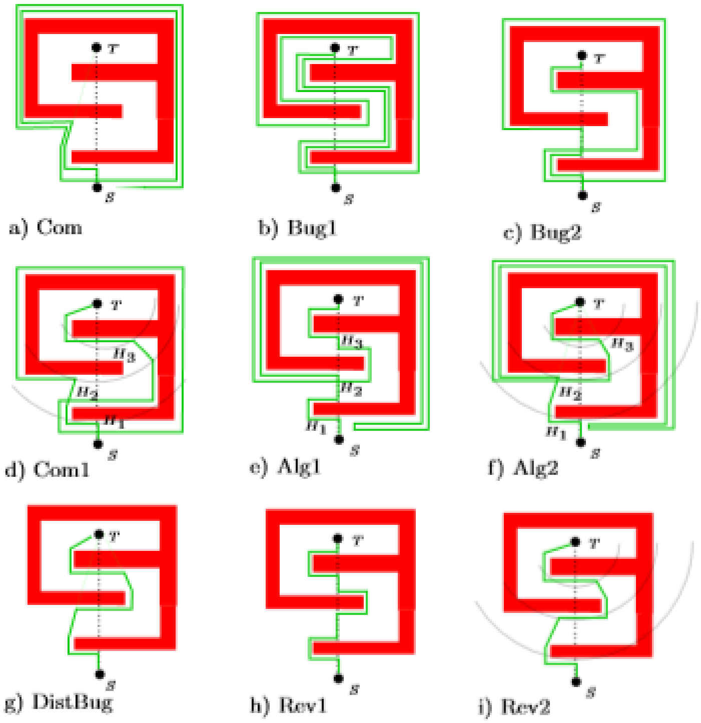
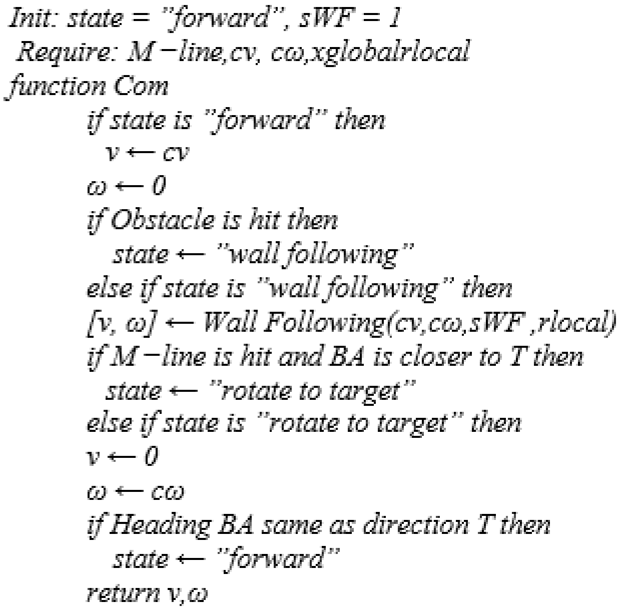


Figure 2.2: Different Bug Algorithms [12]

The idea can be implemented for the robots which are light in weight, the literature study of this paper shows that many of the algorithm variants mainly rely on either a global-localization system or perfect on-board sensors. Also, the simulation experiments evaluated the performance of several Bug Algorithms with varying noise measurements and failure cases, which clearly showed a significant performance degradation of all the algorithms. This shows that the Bug Algorithms can not be implemented on a robot platform used for navigation, which relies only on the on-board sensors without any external help. To enhance these techniques, the experimental results showed that simplicity is a key element, as the most fundamental Bug Algorithm, Com, was the one that was the most resilient to odometry drift and showed most robustness. Another vital element is a loop detection system, where the robot reies on multiple measured variables instead of one, especially in realistic, noise-inducing, environments. Therefore these observations allow bug algorithms to be suitable for the autonomous navigation of small robotic platforms with limited computational resourses.

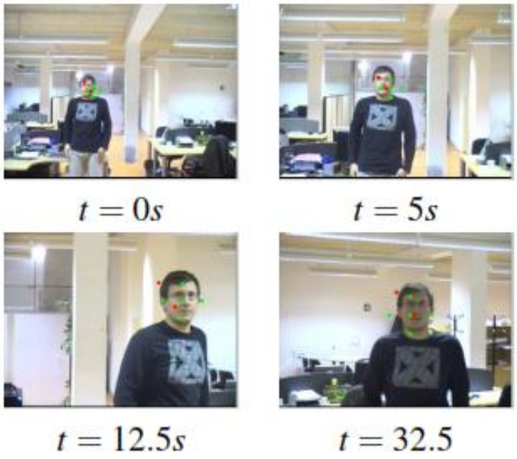
The pseudo-code for the state-machine of Bug2:



```
Init: state = "forward", sWF = 1
 Require: M −line,cv, cω,xglobalrlocal
function Com
          if state is "forward" then
            $v \leftarrow cv$
          $\omega \leftarrow 0$
          if Obstacle is hit then
             state ← "wall following"
          else if state is "wall following" then
          $[v, \omega] \leftarrow$ Wall Following(cv,cω,sWF ,rlocal)
          if M −line is hit and BA is closer to T then
            state ← "rotate to target"
          else if state is "rotate to target" then
          $v \leftarrow 0$
          $\omega \leftarrow c\omega$
          if Heading BA same as direction T then
             state ← "forward"
          return v,ω
```

2.3 Reactive object tracking with a single PTZ came

(Murad Al Haj, 2010) proposes novel method for implementing extended Kalman filter to track position of the object, and find its position and velocity in three dimension. Kalman filter predicts future state of the object. Co-ordinates of this predicted states are used as input to a PID controller to controll pan-tilt and zoom of camera. Pan-Tilt motion keeps object in centre while zoom motion focus on the image . This paper proposes Robust and efficient multipurpose face dection using skin color segmentation. Perforamce perameters such as resolution, framerates, etc. and it effect on tracking effeciency is not mentioned. Here green green dots shows face and two red dots shows orientatino ofa the face, the red dot at centre shows centre of face, while other red dot shows upper let corer of the face.



$t = 0s$ $t = 5s$

$t = 12.5s$ $t = 32.5$

Figure 2.3: Reactive tracking with camera [5]

2.4 Robotic Motion Planning: Potential Functions

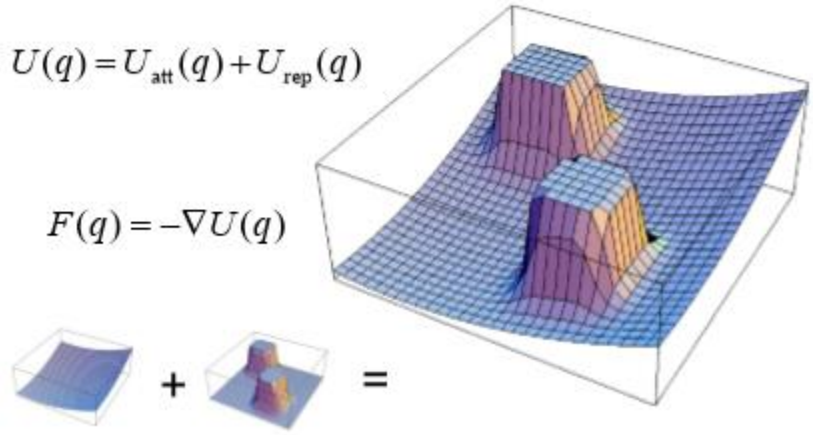
(Howie Choset, RI 16-735):

A simple idea:

- Suppose the goal is a point g∈ℜ
- Suppose the robot is a point r ∈ℜ 2
- Also suppose a “spring” is pulling the robot towards the goal point and away from obstacles
- Similar to the idea of like and dislike charges

Both the bowl and the spring analogies are ways of storing potential energy

- The robot moves to a lower energy point to stabilize itself
- A potential function is a function U(q) : ℜm→ℜ
- The energy is minimized when the robot follows the negative gradient of the potential energy function.
- A vector field over the space of all q values at every point in time, the robot looks at the vector at the point and follows it in that direction.



16-735, Howie Choset, with slides from Ji Yeong Lee, G.D. Hager and Z. Dodds

Figure 2.4: Artificial potential function [6]

2.5 Speeding Up A* Search on Visibility Graphs Defined over Quadtrees to Enable Long Distance Path Planning for Unmanned Surface Vehicles

(Brual C.Shah, Satyandra C. Gupta, 2016): **A*** is a graph traversal and path search algorithm, which is often used due to its completeness, optimality, feasibility and efficiency. The performance of the A* path planning algorithm depends on the estimation of the heuristic cost from the current-state n_I to the goal-state n_G. If the A* path planner underestimates the heuristic, then it has to expand more nodes until it finds an optimal path to the goal. Otherwise, if the path planner overestimates the heuristic, then the h-cost is inadmissible and the paths found are sub-optimal.

2.6 Design and Control for Differential Drive Mobile Robot

The differential drive robot has two fixed powered-wheels mounted on the left and right side of the robot and it has two wheels which are independently driven. If the rotating wheels have the same velocity, the robot moves straight ahead or straight backward. But if one wheel is moving faster than another, the robot follows a curved path along the arc of an instantaneous circle. Finally, if both the wheels are rotating at the same velocity in opposite directions, the robot turns about the centre point of the two driving wheels. This research paper focuses on the design, total mechanical structure, and the velocity control of the differential drive platform. Apart from this, a PID controller based on kinematic modelling is used to control the RPM of the DC motors thus the velocity and rotation of the platform.

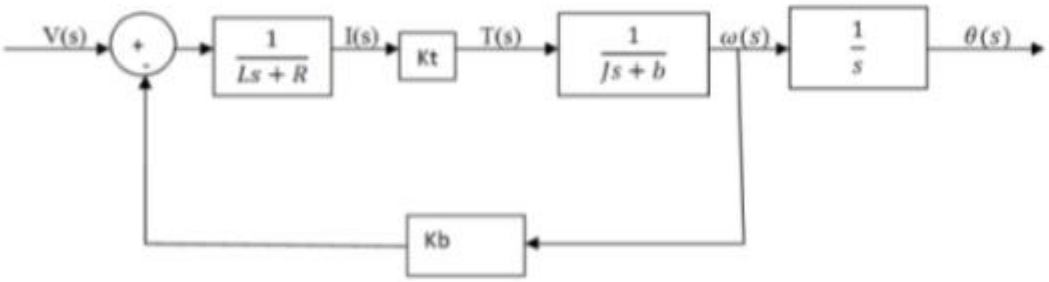


Figure 2.5: Transfer function of the DC motor [7]

2.7 Motion Planning and Control of Differential Drive Robot

(Kaamesh Kothandaraman, 2016): This thesis studies the path planning and control of a differential drive robot along with its implementation. The atan2(tan inverse) function plays a vital role in this thesis. The desired orientation of the robot, anywhere in the plane, can be found out using the atan2 function which gives the orientation towards the destination(goal) without worrying about the quadrant signs. Any position in the workspace can be achieved by controlling the velocity and orientation of the robot precisely. The atan2 function was useful in driving the robot to a point, drawing shapes and pursuing other moving object. Systems such as Vicon camera system, QUARC controller, Xbees and Redbots were used in the experiments. Also, these systems were configured for real-time data communication.

Many different controllers such as PID were tested, but a simple proportional controller provided optimum results compared to other complicated controllers. The robustness of the controller was tested using various tasks such as point-to-point motion and shape formation and the results were used to optimize the controller to enhance the performance. Also, the controller was tested by performing tasks such as obstacle-avoidance, leader follower and pursuer evader.

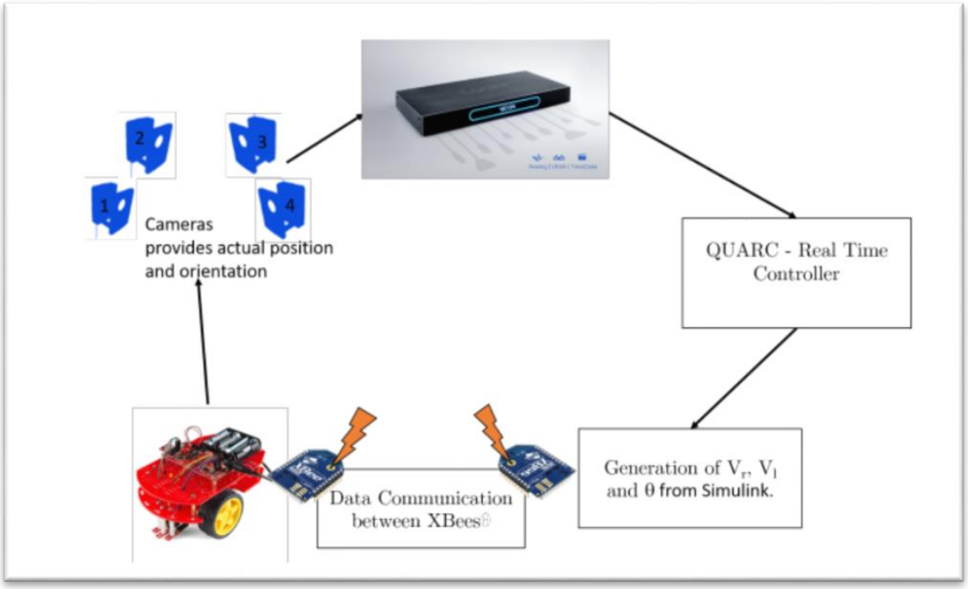


Figure 2.6: System Integration used in the paper [8]

2.8 Summary

After detailed observation of the research in this topic, and study of various path planning algorithms like bug algorithms, Artificial Potential field method, Visibility Graph method, Voronoi Diagram, Cell Decomposition, RRT etc., it was decided to use the visibility graph algorithm with a search algorithm like Dijkstra’s or a heuristic search algorithm like A, because visibility graph gives the shortest path from start to goal (Euclidean distance) and other methods are a bit conservative. Also, there are many problems in other methods like in artificial potential field method there is a big possibility that the robot gets stuck in the local minima of the field, and in RRT (Rapidly-Exploring Random Trees) there is a big chance that robot does not reach the obstacle at all if the course id complex or the iterations are less.

Also, in electronics hardware selection one should focus on a microcontroller that contains the inbuilt facility for communication protocols like Wifi or Bluetooth to avoid integrating additional components, also the motor driver should have 2 or more channels, so that we can control the robot with a single motor driver and thus it can avoid much complexity.

Chapter 3
Design and Assembly

It requires to first to build a concept, based on the concept a design has to be prepared. We are going to make a continuous tracked tank drive differential robot for this project. It gives high traction on every surface because of the tracks. Each track has a driving wheel and a free wheel. Figure 3.1 shows the assembled version of the tracked tank drive.

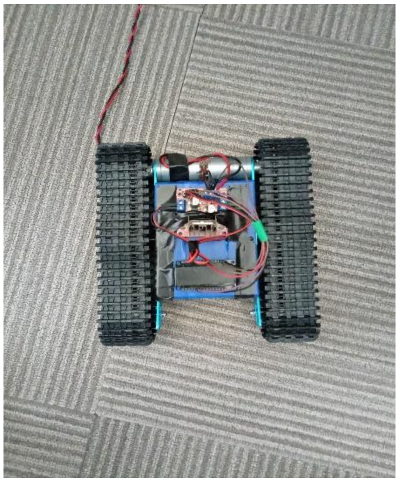


Figure 3.1: Assembled tank drive

3.1 Modelling and Design

Design is made considering size of the system, it should be compact. The system should be light weight because it has to be operated with small motors which have limited torque therefore inertia of moving parts should be less. After considering all conditions finally the design and the 3d model were prepared for the same. The 3d model and its drawings are made in Solidworks (CAD software).

The design is based on availability of the standard components such as the track size and the free wheel and drive wheel dimensions. Here, every part is standardly available so nothing is manufactured to make this drive. Thus a bottom to top approach is used while designing parts. i.e. based on the available part dimensions the whole robot is made. Also, the overall design is kept small to avoid the obstacles easily and navigate the environment without hassle. Furthermore, the electronics is safely mounted on the top of the drive and secured firmly with insulation tape.

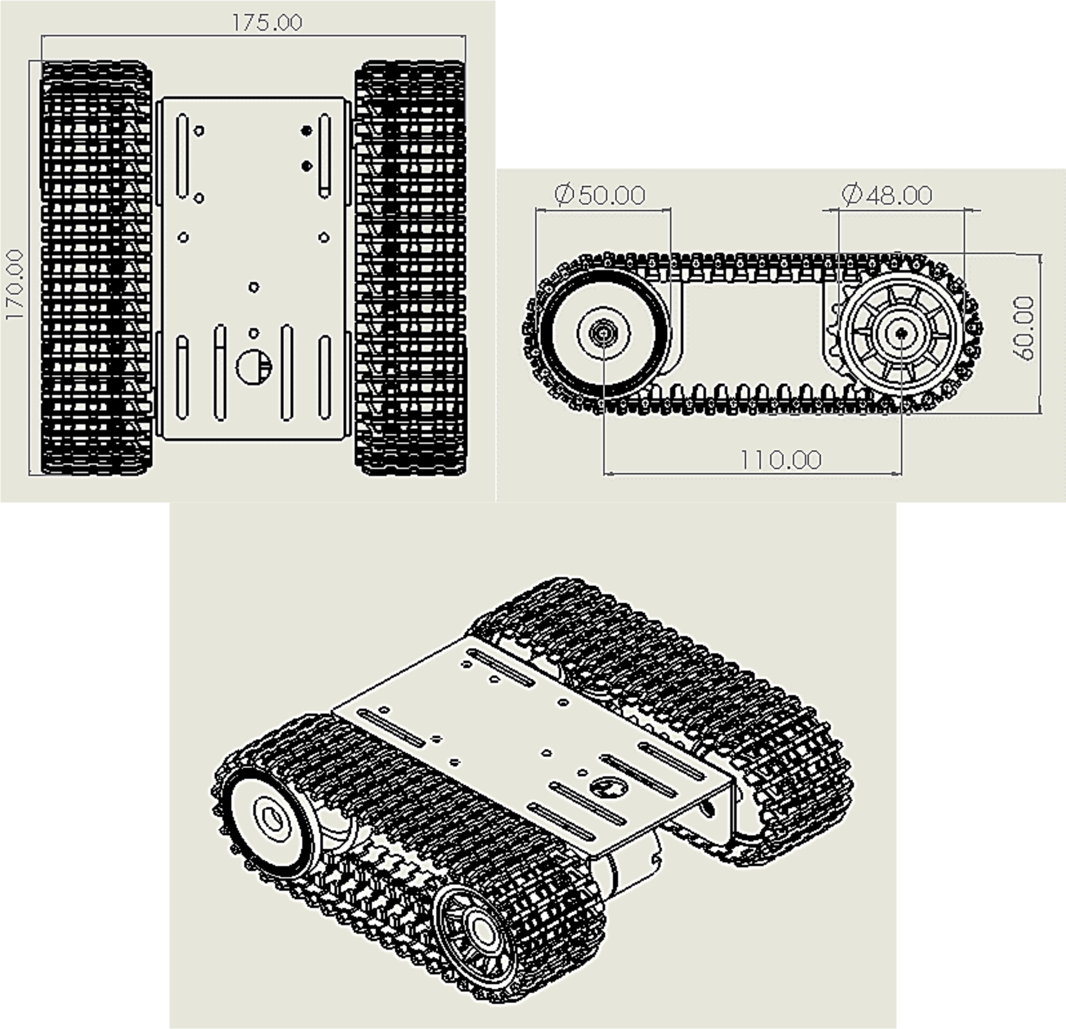


Figure 3.2: Overall dimensions of the design

3.2 Assembly

Following is the basic parts required for the system and their cost estimation.

3.2.1 Basic parts cost estimation

Following are basic parts and cost of the parts that is required for the differential drive setup.

Part name	**Price**	**Qty.**	**Total price**
ESP32 Module	950	1	800
100 RPM DC Motors	150	2	300
Logitech C310 Camera	1800	1	1800
12V 10A SMPS	1300	1	1300
L298 Motor driver	200	1	200
Tank Components	1500	7	1500
Sub total			5900
GST (18%)			1062
Total			6962

Table 1: Primary parts for differential drive

3.2.2 Part and its specifications

Part specifications of all parts used in this project are as follows.

Sr. no.	Name	Part	Specifications
1	ESP32 Module		**CPU**: Xtensa dual-core (or single-core) 32-bit LX6 microprocessor, operating at 160 or 240 MHz and performing at up to 600 DMIPS **Memory**: 520 KiB SRAM **Wireless connectivity**: Wi-Fi: 802.11 b/g/n Bluetooth: v4.2 BR/EDR and BLE (shares the radio with Wi-Fi) **Peripheral interfaces**: 12-bit SAR ADC up to 18 channels 2 × 8-bit DACs 10 × touch sensors (capacitive sensing GPIOs)

			4 × SPI 2 × I^{2}S interfaces 2 × I^{2}C interfaces 3 × UART Motor PWM LED PWM (up to 16 channels) Hall effect sensor Ultra low power analog pre-amplifier
2	100 RPM/12 V DC Motor		**Torque:** 1.2kgcm **Speed:** 12V- 100 RPM **Weight:** 125 gms **Dimensions:** 23X12.2X29 mm **Motor Type:** DC brushed with gearbox

3	Logitech C310 Camera		2.4 GHz Intel® CoreTM2 Duo 2 GB **RAM** 200 MB **hard drive space** **USB 2.0** port 1 Mbps upload speed or higher 1280 x 720 **screen resolution** **Weight:** 250 gms
4	12V 10A SMPS		**Output**: 12V, 10Amp **Input**:180V to 260V AC , 47 to 63Hz LED power indication **DC calibration range**: +-10% of the output voltage **Overload protection**: up to 150% with auto recovery **Dimension**:200 x 98 x 40mm

			Product weight: 625gms
5	L298 Motor driver		**Double H bridge unit** **Chip:** L298N (new ST) **Logic voltage**: 5 V **Unit voltage**: 5 V-35 V **Logic**: 0mA-36mA **Transmission current**: 2 A (Maximum single bridge) Maximum **Power**: 25 W **Size**: 43x43x26mm

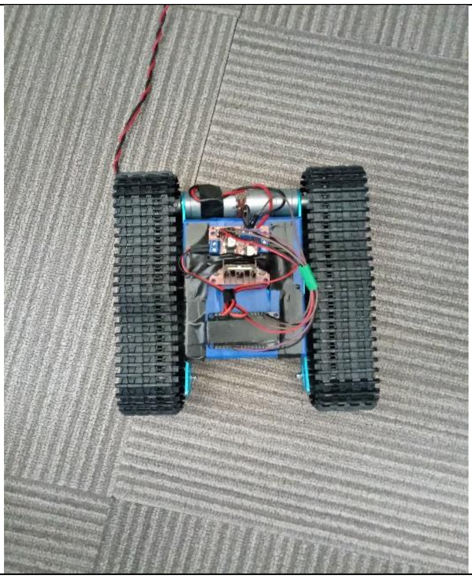
6	Tank Components		**Dimensions mentioned in the “Design” section**

Table 2: Parts and its specifications

And other small parts such as …

- 5V 2A power adapter
- Breadboard
- Solder station
- Jumper Wires
- S.S M3 bolts and nuts
- Screwdriver and spanners
- Solder flux
- Insulation tape
- Wire cutter

3.2.3 Solidworks Parts and Assembly

Following figures depict the individual parts of the differential drive and their connection to the assembly. As mentioned before these are the standard available parts in market and the assembly is a bottom to top approach.

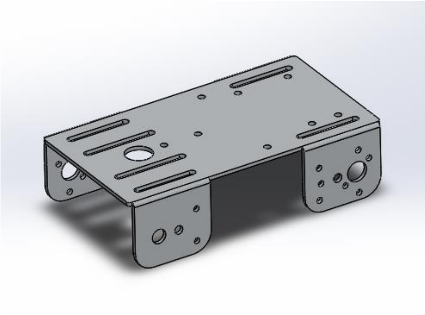


Figure 3.3: Chassis

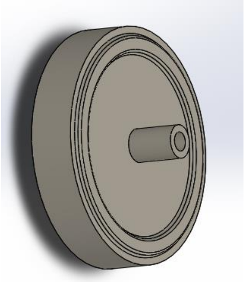


Figure 3.4: Free wheel

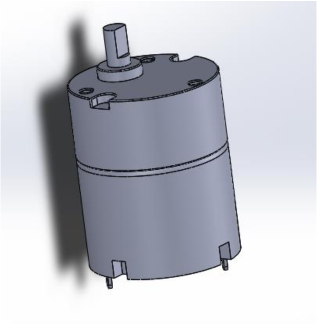


Figure 3.5: DC motor

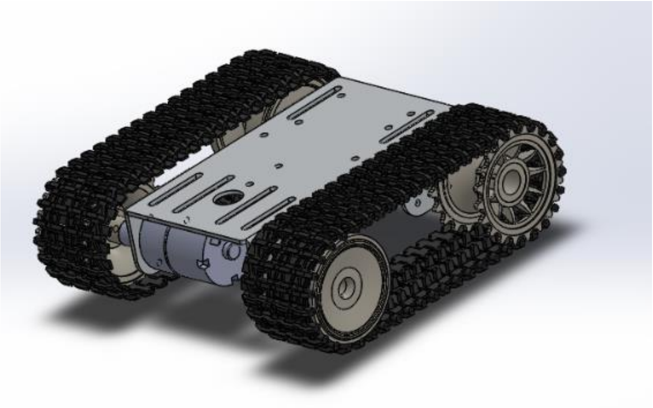


Figure 3.6: Complete Assembly

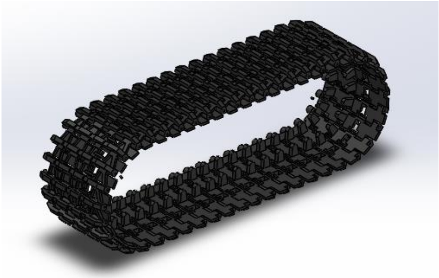


Figure 3.7: Track

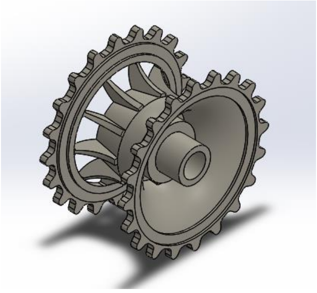


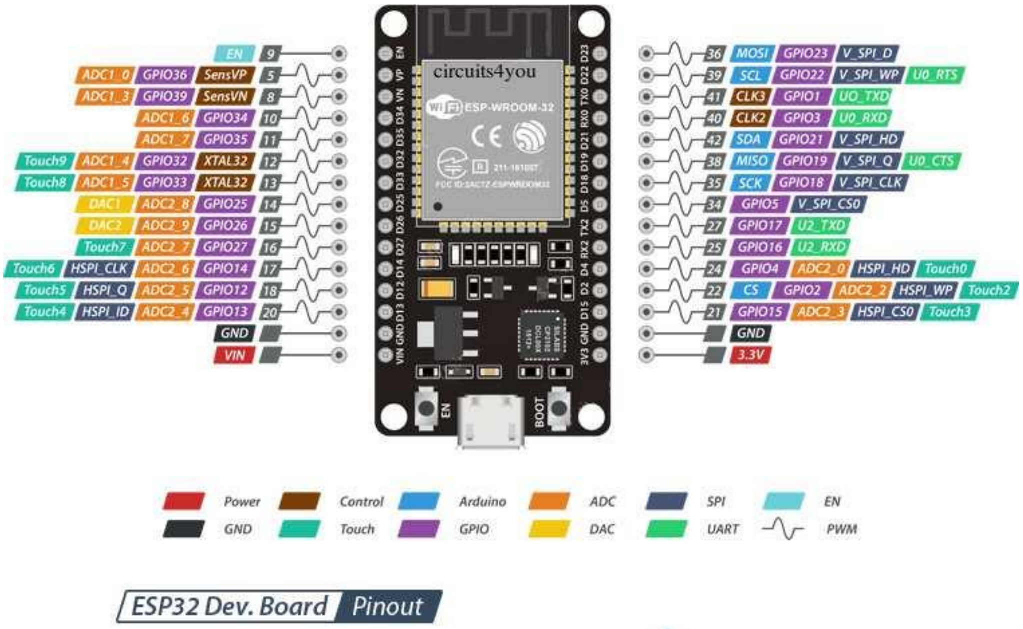
Figure 3.8: Driver wheel

Chapter 4
Hardware Integration and Communication

This chapter describes required hardware components, their connections, layout architecture, and the communication layer between the controller and the motor driver. This chapter will focus on the working of the controller and motor control. A client-server protocol is also established between the laptop and the controller.

4.1 Microcontroller

A microcontroller is a small chip having onboard processor, RAM, IO pins, Memory etc. The microcontroller controls the system by executing the commands written in the code and issuing them to the motor driver. Furthermore, it is also responsible for the processing and the calculation required to run the drive. As mentioned in the previous chapter, we will be using ESP 32 as our main controller as it has both Wi-Fi and Bluetooth connectivity to receive and transmit signals. This module is very handy and is widely used in IOT based applications. It combines the functionality of different modules in one and is very easy to program. We can program it using the Arduino Ide and importing the ESP package in it. It also has many GPIO (General Purpose Input-Output) pins, UART support, SPI, I2C etc. Conclusively it is a perfect board for this application.



EN 9
ADC1_0 GPIO36 SensVP 5
ADC1_3 GPIO39 SensVN 8
ADC1_6 GPIO34 10
ADC1_7 GPIO35 11
Touch9 ADC1_4 GPIO32 XTAL32 12
Touch8 ADC1_5 GPIO33 XTAL32 13
DAC1 ADC2_8 GPIO25 14
DAC2 ADC2_9 GPIO26 15
Touch7 ADC2_7 GPIO27 16
Touch6 HSPI_CLK ADC2_6 GPIO14 17
Touch5 HSPI_Q ADC2_5 GPIO12 18
Touch4 HSPI_ID ADC2_4 GPIO13 20
GND
VIN
36 MOSI GPIO23 V_SPI_D
39 SCL GPIO22 V_SPI_WP U0_RTS
41 CLK3 GPIO1 U0_TXD
40 CLK2 GPIO3 U0_RXD
42 SDA GPIO21 V_SPI_HD
38 MISO GPIO19 V_SPI_Q U0_CTS
35 SCK GPIO18 V_SPI_CLK
34 GPIO5 V_SPI_CS0
27 GPIO17 U2_TXD
25 GPIO16 U2_RXD
24 GPIO4 ADC2_0 HSPI_HD Touch0
22 CS GPIO2 ADC2_2 HSPI_WP Touch2
21 GPIO15 ADC2_3 HSPI_CS0 Touch3
GND
3.3V
circuits4you
ESP-WROOM-32
EN VP VN D34 D35 D32 D33 D25 D26 D27 D14 D12 D13 GND VIN
D23 D22 TX0 RX0 D21 D19 D18 D5 TX2 RX2 D4 D2 D15 GND 3V3
EN
BOOT
Power
GND
Control
Touch
Arduino
GPIO
ADC
DAC
SPI
UART
EN
PWM
ESP32 Dev. Board Pinout

Figure 4.1: Pinout of ESP32 [9]

4.2 Motor Control using Motor Driver

Motor Driver: A motor driver is an electronic circuit which is used to run the motor at desired speed and in the desired direction. Motor drivers come in many varieties, but we are going to use the DC motor driver. It is able to operate the motor in both directions and also it provides a continuous spectrum of intermediate RPM's between zero and the maximum RPM. It basically works on the principle of Pulse Width Modulation signal being fed to the on-board MOSFETS to drive the motors with different speeds and for direction reversal, it uses an H-bridge. High resolution PWM signals are desirable for smooth control as they allow for more interpolations between the zero and maximum RPM range of the motor, thus providing superior control with greater accuracy. Furthermore, the motor driver should be carefully chosen according to the current requirements of the motors to be used, it may be possible that motor requires high current but the driver is unable to provide it.

Here, we are going to use L298N motor driver. It is a dual channel motor driver i.e it allows us to control 2 motors simultaneously thus it is perfect for our application. Also it can handle 3A at 35V. Furthermore, it also has PWM pins to provide intermediate voltages.

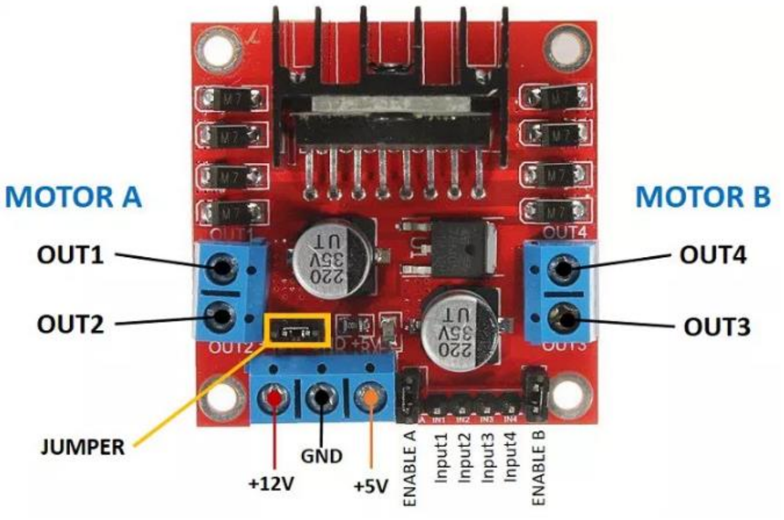


Figure 4.2: L298N pinout [10]

4.3 Hardware Integration

Figure 4.3 shows the connections between the controller, motor-driver, power supply and the motor. The motor driver has a two terminal blocks mounted at each side for each motor.

OUT1 and **OUT2** denotes the positive and negative terminal of motor 1

OUT3 and **OUT4** denotes the positive and negative terminals of motor 2.

12V: The +12V terminal is the power supply

GND: Ground terminal

+5V: Provides 5V if jumper is removed. If jumper is present, it acts as 5V output.

Jumper: If we supply more than 12V, you should remove the jumper, also if jumper is in place it uses the motor power supply to provide to the onboard chip.

- If a HIGH signal is sent to the enable 1 pin, motor 1 is ready to be controlled with the maximum speed
- If a LOW signal is sent to the enable 1 pin, motor 1 turns off.
- If a PWM signal is sent to the motor driver, we can control the RPM(Rotations Per Minute) of the motor. The motor speed is directly proportional to the duty cycle. But, if a signal lower than a threshold is sent the motors might make a continuous buzzing sound.

The DC motor requires a big jump in current to move, so the motors should be powered using an external power source from the ESP32. We are using a 12V/ 10A SMPS (Switch mode Power Supply) to power the motors. The switch between the battery holder and the motor driver is optional, but it is very handy to cut and apply power. This way you don’t need to constantly connect and then disconnect the wiring to save power. Also a 0.1uF ceramic capacitor is soldered to the positive and negative terminals of the DC motor, as shown in the diagram to help smooth out any voltage spikes.

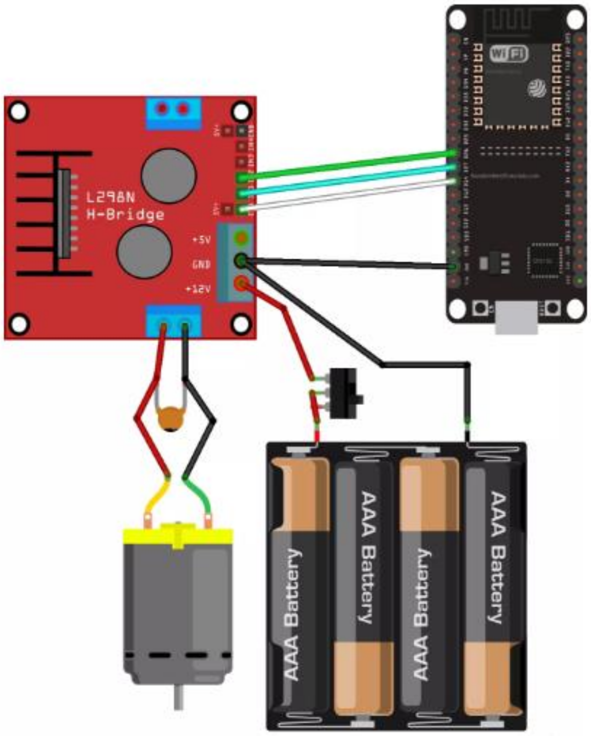


Figure 4.3: Schematic Connection of hardware

4.4 Socket Server-Client Connection Model

Now, to transmit the velocity values nad the angles to the robot we need some sort of connection betweeen the robot and the Pc on which the image will be processed. So, we use a Server- Client Model to transmit the data. In a server-client architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and delivers the data packets requested back to the client. Socket is the endpoint of a bi-directional communications channel between the server and the client. Sockets communicate within a process, between processes on the same machine, or between processes on different machines. For any communication with a remote program, we have to connect through a socket port.

In this project, the Pc is the socket server and the ESP32 board is the client and an end to end encrypted connection is setup between them via the mobile hotspot of the Pc. Here, the Client sends the request to the server to send the data and waits till it receives the velocity and angle values from the Pc. The Pc calculates these values by tracking the robot through the overhead camera and applying the motion planning and control algorithms on it.

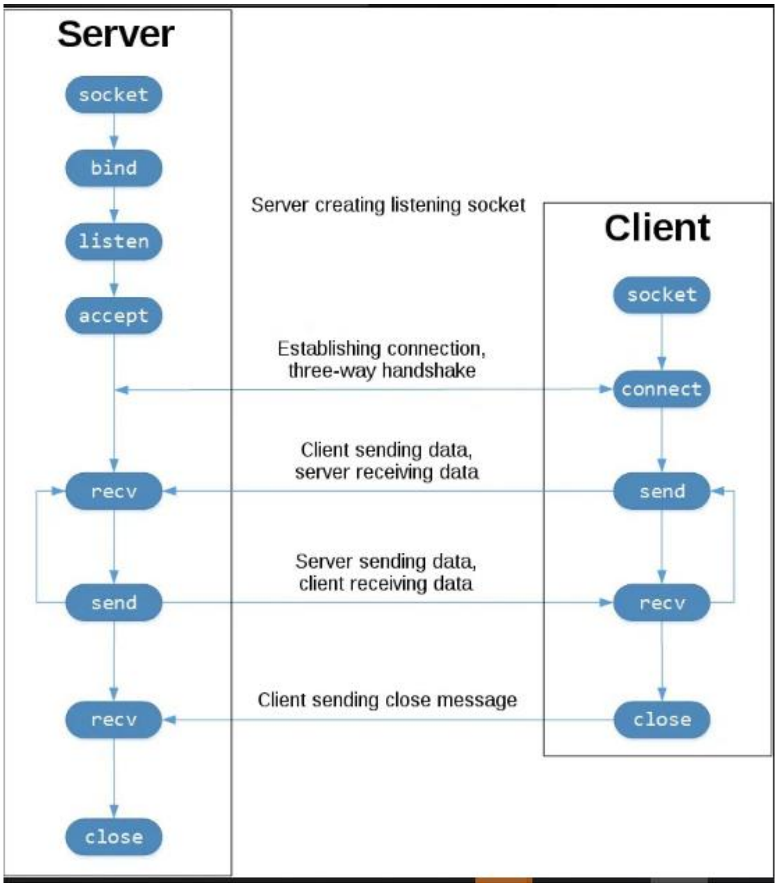


Figure 4.4: Server Client Interaction

Chapter 5
Obstacle Detection Using Computer Vision

Now that the body is ready, it needs life. In order to receive images, process the image and control the whole system it needs to be programmed. Programing is soul of this project. Several ways of tracking the objects are prepared and results are observed and compared in order to improve performance of the system. This chapter explores the various image processing techniques to segment the robot and the obstacles from the environment.

5.1 Programming

Python is used as primary language for programming and image processing. **C** language is used for programming ESP32.

5.1.1 Python programming

Python is chosen because of its superiority with simplicity. Following are some points which were considered.

1. **Simplicity**
   Technically python is much slower as compare to other programming languages like C, C++, Java, etc. But it is simple, its syntax is easy and natural. Writing code in python feels like writing a story. Anyone with basic programming knowledge can master Python.

2. **Libraries**
   It has a huge collection of standard libraries. There is a library of almost everything. Also making library is very simple. That’s why its library collection is enormous and growing every second.

3. **Integration**
   Python can be integrated with other languages like C, C++, etc. In fact, Open CV (Open Source computer vision) python library is having its source code in C/C++.

4. **Community**
   Because of its simplicity and flexibility everyone is using python and so am I. It has huge community and support. The first Google search engine was completely made in python. Many tech giant uses python as primary programming language. Therefore, it is having huge active community.

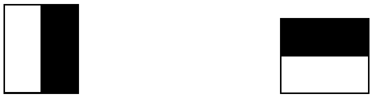
5.2 Obstacle Segmentation:

There are many algorithms to identify an object from image, Haar cascade is one of them. Haar-cascade is a Machine Learning algorithm that match features and identifies weather the object is there in image or not. Any machine learning algorithm needs training and for training it needs data. To train a dataset with machine learning first we need huge set of classified data. More data, better will be the training. Generally, data is divided into three sets 1) Training dataset 2) Test dataset and 3) Cross validation set. The model is trained with training set and checked with other two. This is basic process of training a machine learning algorithm.

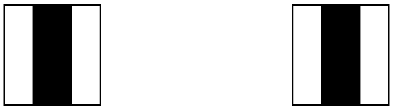
During training of model it identifies Features of the face. Once model is trained and we give an input image for classification, it matches the features of face with input image. If sufficient amount of features are matched, then it classifies the image as positive and indicates that there is a face in the image else it gives negative. It is similar to how humans identifies things. If we want to teach a kid to identify an apple, we first show him/her a pictures of apple and tell "This is an apple", then we show pictures that is not apple and tell "This is not apple". Now training is done. Similarly, In case of computers it requires both positive and negative dataset ("face" and "no face" images), since computers are less smart it requires large amount of training data, In machine learning it is said that **"The winner is not one who has best algorithm, The winner is one who has best data".** Humans requires few images to identify and object; after seeing only one picture of apple we can identify an apple; however, computers are becoming smart due to rapid technology advancement new algorithms are coming every day, now it is also possible for computers to learn from very few training example, it is being called **"One-shot learning"**.

Humans can identify features, but computers only understand "0" and "1". We can find feature that if an image has nose, eyes, lips, etc. there must be a face in image. For computers there are basically three kind of features as shown below.

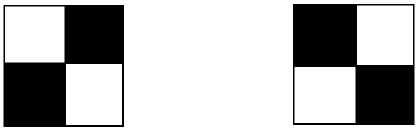
1. Edge feature



2. Line feature



3. Rectangular feature



A feature corresponds to subtraction of pixels below black area from pixels below white area, it gives a value and that is a feature of an image. When an input image is provided features are extracted from the image and then it is matched with training data, features are extracted in form of raster scan; starting from top left corner to bottom right. As stated before even a small resolution has large amount of features, in haar cascade we extract features in several steps. In first scan we extract some features, the portion where face features are matched, there is probability of being a face and the portion where face features are not matched for example, if there is a flat wall or window in background, in that portion face features are not found hence probability of being a face in such area is very low. In second scan the area in which features are matched in previous scan is only scanned, and other portion is discarded. This saves lots of time and computational power, that is the reason why it is called a cascade method. Basic illustration of the process is shown below. It should be noted that it only detects faces available in image, facial recognition is a whole different thing from facial detection.

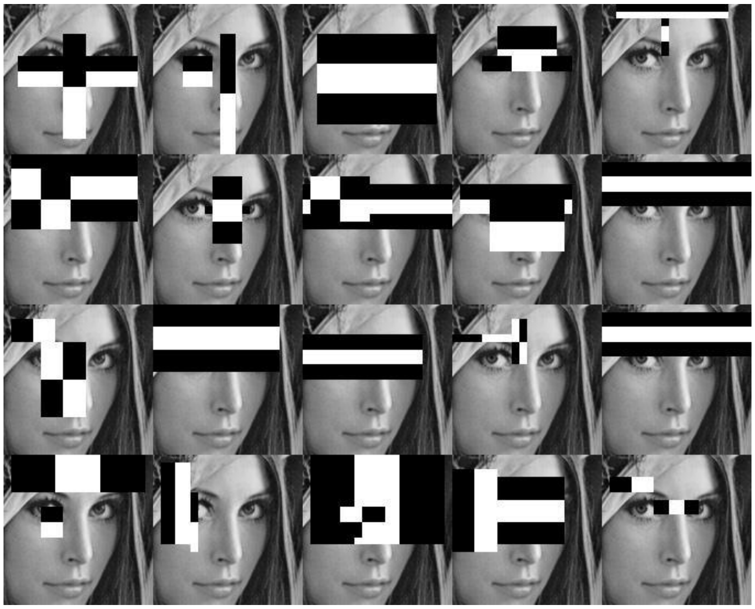


Figure 5.1: Haar-cascade for face detection [2]

5.3 Perspective Transform

Perspective Transform is a technique to eliminate the errors in the placing of camera to track the robot. It is basically a transformation matrix to map/superimpose the original points to new points. If the camera is mounted at an angle, or if the mounting distance change it may affect the distance scaling between pixels and the real world distances. So, we use the Opencv's "cv2.warpPerspective( )" function to transform the obstacle course using its corner points to the computer screen to a previously defined aspect ratio.

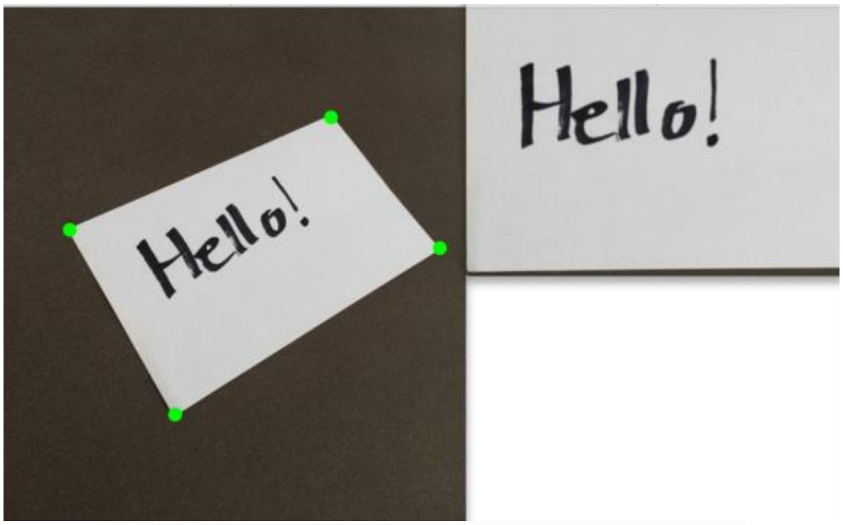


Figure 5.2: Perspective Transform of a rectangular object [2]

5.4 Color filtering

In color filter a threshold is provided, if value of pixel falls between given range then pixel value is high (White) else low (Black). First we need to convert image from BGR (Blue Green Red) to HSV. HSV format is more analogous to human eye, we don’t see colors like how much portion is red, blue, how much is green. We see colors in terms of color, its brightness and illumination.

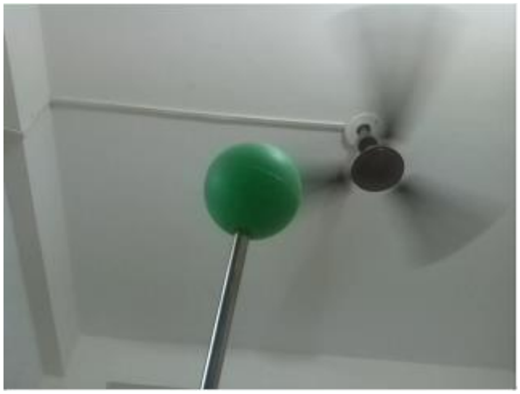


Figure 5.3: Original image for blob detection

Then a threshold is applied that outputs a binary image, the pixels whose values fall within a range are set to 1(White). Also, there is lots of noise in the image because image also containing of background that may have similar color to tone as object to be found. These noise can be seen inform of small dots. To remove this noise some blur, dilution and erosion is applied. After applying threshold and noise removal image can be seen below.

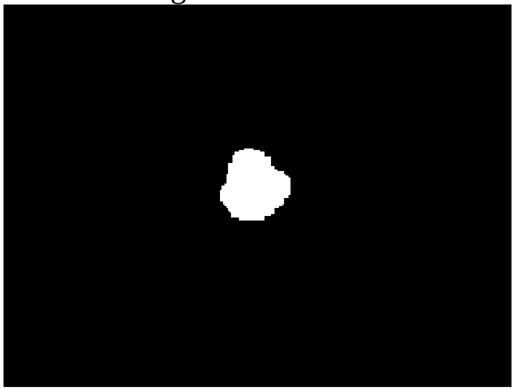


Figure 5.4: Masked image for blob detection

Now largest cluster represents out object. If background is of similar color as the object, it may give false positive results. The method finds largest cluster of white pixels.

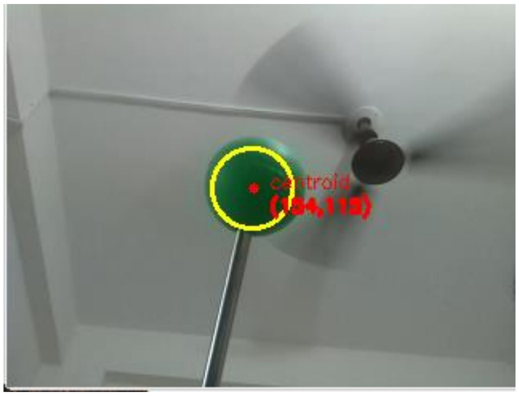


Figure 5.5: Blob detection

5.5 Contour Detection

A contour can be explained as a curve joining all the continuous points(along the boundary), having same color or intensity. The contours are a very useful tool for shape analysis and object detection and recognition. We need contour detection to separate the obstacles from the background, so that we can draw the visibility graph using the obstacle corners. Also, to account for any convexity and continuity in obstacles, we fit a 4-sided polygon to each obstacle using its extreme points, so as to simplify the construction. In figure 5.7 the green polygons surrounding the different shaped obstacles are the fitted contours.

The green contours are made using the minimum bounding area, so it considers the rotation also. The function used is cv2.minAreaRect(). It returns a Box2D structure which contains following details - (top-left corner(x, y), (width, height), angle of rotation). But to draw this rectangle, we need 4 corners of the rectangle. It is obtained by the function cv2.boxPoints().

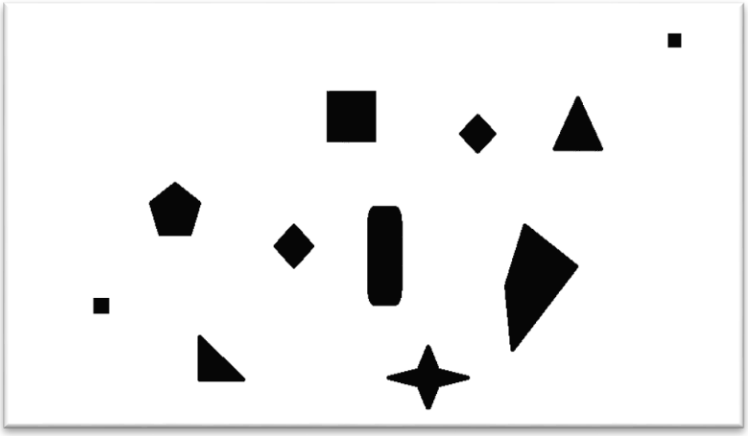


Figure 5.6: Obstacle Course

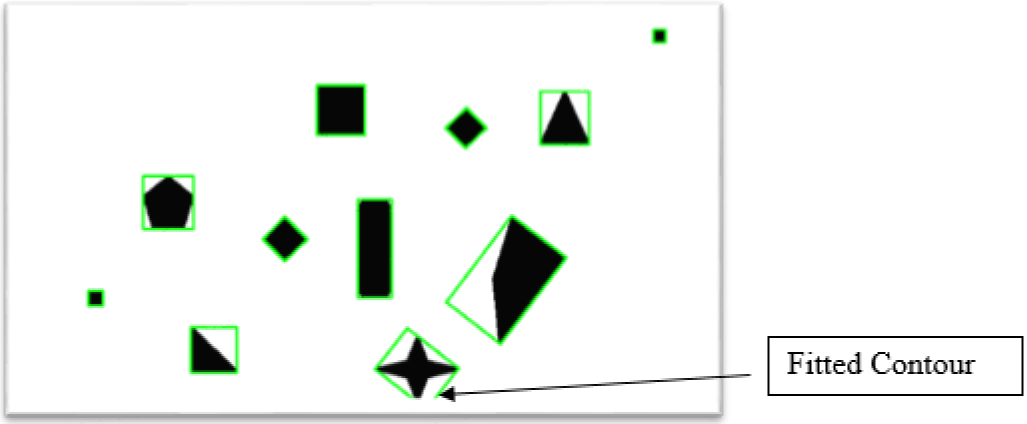


Figure 5.7: Obstacle course with contours

5.6 Augmented Reality (AR) Marker

Augmented reality (AR) is a live, direct or indirect, view of a physical, real-world environment whose elements are augmented by computer-generated sensory input such as sound, video, graphics or GPS data. Different AR markers are the shapes on images that can be easily detected by a camera. Most markers are black and white, but colors can also be used as long as the contrast between them can be properly recognized by a camera. A simple AR marker can consist of one or more basic shapes made with black squares above a white background. More complex markers can be created using simple images that are still read properly by a camera.

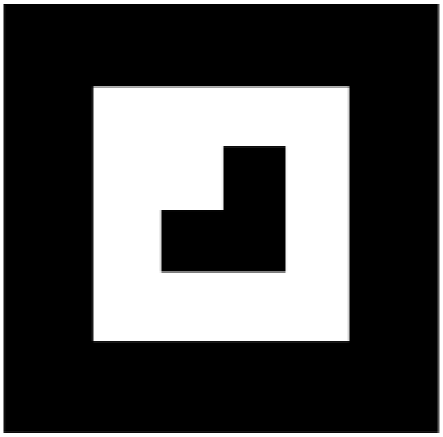


Figure 5.8: A simple AR marker

Augmented Reality markers are used to find the SIFT keypoints with descriptors in the image and as we know them beforehand due to the pattern of the AR marker, it helps to pin point a certain region precisely in the image. Here, it is attached on the top of the robot to distinguish the robot from the obstacles in the environment.

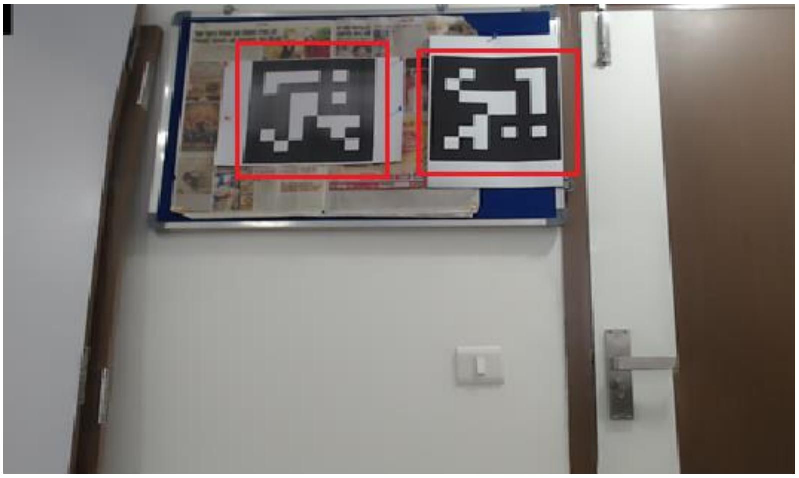


Figure 5.9: Detection of the AR marker

Chapter 6
Path planning of the robot

6.1 Path Planning

A rudimentary need in robotics is to have algorithms that convert high-level specifications of tasks from humans into low-level descriptions of how to move. The terms motion planning and path planning are often used for these kinds of problems. Path planning is a task of finding a continuous path from the start point to the goal point. When path planning for the robot is done, it is assumed to be a point object traversing on the path, thus its movements are considered in motion planning of the robot. There are many techniques to generate the paths. Some techniques are simpler than others, but not all techniques are complete (i.e not all techniques guarantee to find a path, for example RRT (Rapidly-Expanding Random Tress etc.)). To study the topic of path planning we require the understanding of what a configuration space is?

6.2 Configuration Space

The robot can take various discrete positions on a graph, which we can enumerate and these can be connected by edges to make the path continuous. Configuration space is a handy mathematical and conceptual tool which was developed to help us think about some kinds of problems in a unified framework. Conclusively, the configuration space of a robot is the set of all configurations and or positions that the robot can attain. Suppose we have a 2D space with a single robot and a single obstacle. Now, for simplicity we assume that the robot has only 2 Degree of Freedom (DOF) i.e translation in X and Y. Now, the region covered by the obstacle is the set of all possible configurations that the robot cannot attain. And conversely, the set of all possible configurations the robot can attain is known as freespace. Now Figure__ shows the configuration space obstacle, by considering the robot and the obstacle dimensions. The configuration space obstacle is derived using the ***Minkowski sum*** of the obstacle. The Minkowski sum of two sets of position vectors *A* and *B* in Euclidean space is formed by adding each vector in *A* to each vector in *B*, i.e

$$A + B = \{a + b \mid a \epsilon A, b \epsilon B\}$$

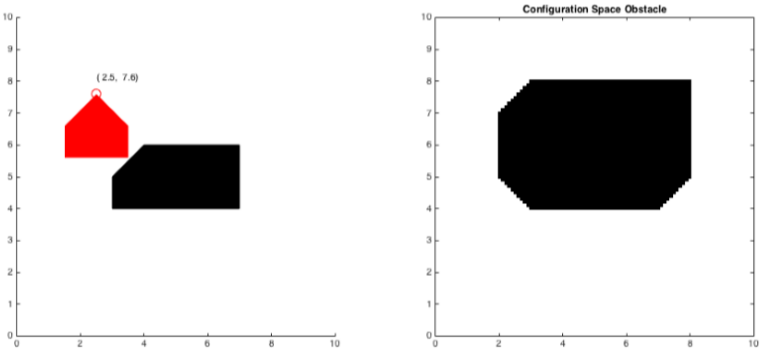


Figure 6.1: Robot in a 2D space and a configuration space obstacle [11]

In case of multiple obstacles in space, we can say that the union of all the configuration space obstacles helps to form the configuration space. Here, the white part is the free space and the black area is the space where the robot cannot go.

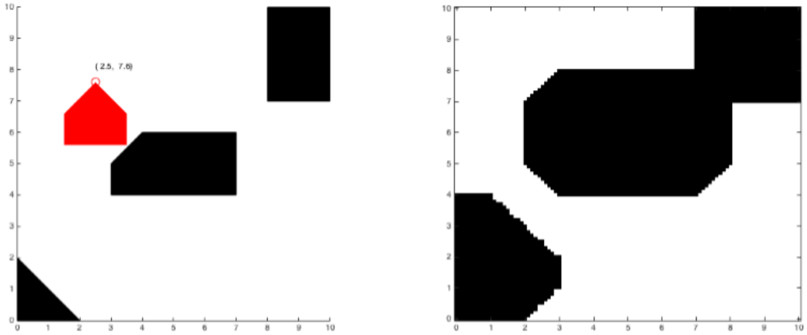


Figure 6.2: Robot in a 2D space and configuration space obstacles [11]

6.3 Graph Generation Techniques

6.3.1 Visibility Graph

A visibility graph is a graph of intervisible locations, typically for a set of points and obstacles in the Euclidean plane. Each node in the graph represents a point location, and each edge represents a visible connection between them. That is, if the line segment connecting two locations does not pass through any obstacle, an edge is drawn between them in the graph. When the set of locations lies in a line, this can be understood as an ordered series. Here, a set of n disjoint-polygonal obstacles are given in the plane, and two points start point and the goal point that lie outside of the obstacles. We have to determine the shortest path from start to goal that avoids the interiors of the obstacles. To solve this problem, the shortest path between any two points in the graph is that, which avoids a set of polygonal obstacles and forms a polygonal curve, also whose vertices are either vertices of the obstacles or the points start position and goal position.

To construct the visibility graph we must determine which vertices are visible from a given vertex. To do this:

- Input a series of vertices n_i, whose edges do not intersect (total of N)
- Join the given vertex n_g to all other vertices n_i (which has a total of N-1).
- Furthermore, check if the line segment $n_g n_i$ intersects the given N edges.
- Keep repeating the procedure for every vertex (total of N).
- This algorithm has a time complexity of $O(n^3)$.

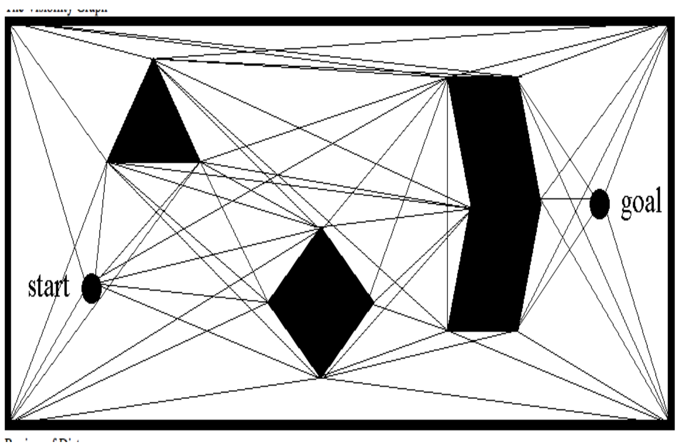


Figure 6.3: Visibility Graph Generation

6.3.2 Voronoi Diagram

On a plane, for a set of sites (points in that 2D space with similar properties) the Voronoi diagram partitions the space based on the minimal distance to each site.

To construct a voronoi diagram, follow the below steps:

- First draw straight lines connecting upper and lower boundaries in the map from the vertices of all obstacles.
- Then, find mid points of the line segments generated by the obstacle vertices and boundaries or by consecutive vertices.
- Finally, connect all the mid points to produce voronoi diagram.

As we can see, Voronoi diagram contains the edges of the path which are distant from the obstacle edges. So, it is a bit conservative algorithm, so it gives sub-optimal paths, but it is a safe algorithm, thus reducing the chances of collision.

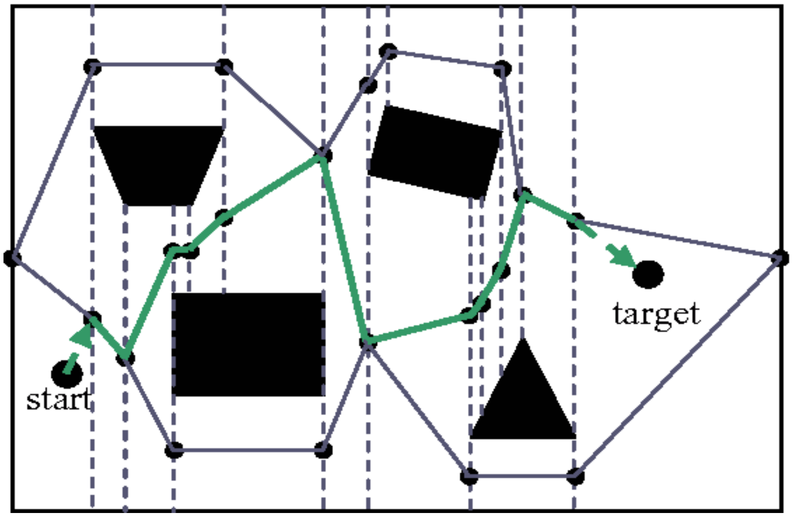


Figure 6.4: Voronoi Diagram

6.3.3 Cell Decomposition

Cell Decomposition is dividing the obstacle course into smaller sub-divisions of similar shape. A fixed resolution is decided that would be used to divide the course, it could be a pixel too. Therefore when the decomposition is uniform it is known as exact cell decomposition. An example of this method is illustrated below.

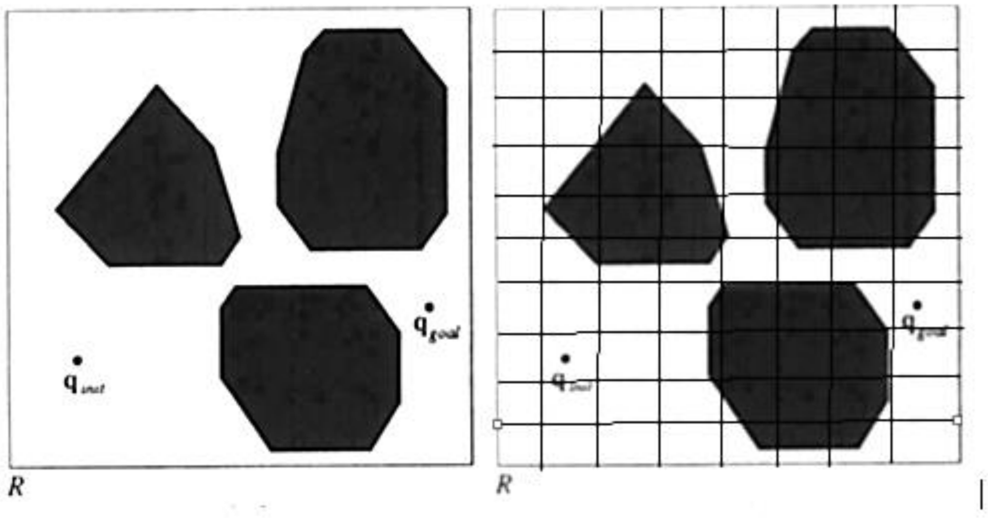


Figure 6.5: Exact Cell Decomposition

A better approach to conserve time and reduce time complexity in the search algorithm would be to decompose these cells unequally and keep bigger cell sizes where there are no obstacles and smaller cell sizes for obstacles. It can use an iterative/recursive method to continue subdividing the cells until any one of the following scenarios occurs:

- Each cell in the grid lies either fully in the free space or fully in the Configuration space-obstacle region
- An arbitrary limit resolution is reached, i.e the cell can’t be further divided.

Once a cell fulfils one of these criteria, it stops decomposing. This technique is called a "quadtree" decomposition, because a cell is divided further into four smaller cells of the same shape every time it gets decomposed. Therefore, after the decomposition step, the free path can then easily be found by following the adjacent cells that lie in the free space.

This would significantly improve the algorithm run time by reducing the nodes and keeping the path same. This is known as approximate cell decomposition. An example of this method is shown below.

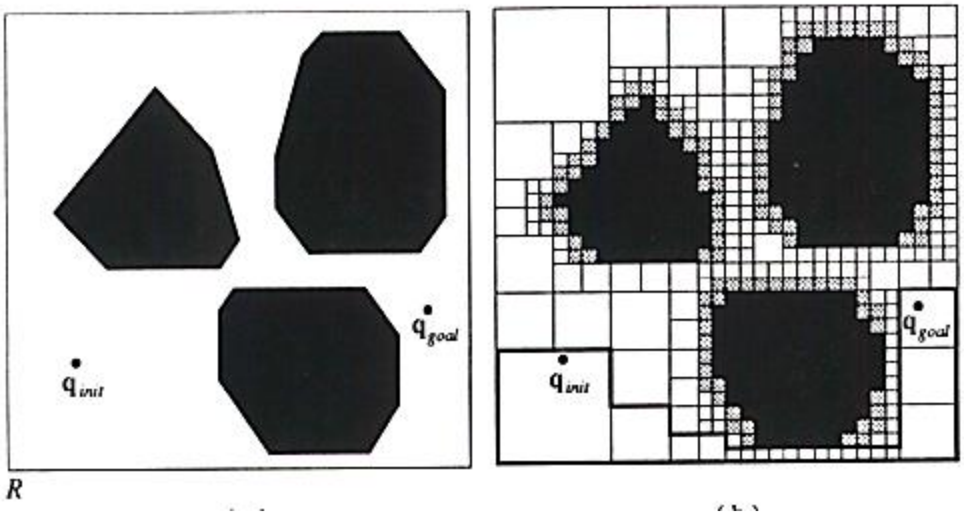


Figure 6.6: Approximate cell decomposition

6.4 Graph Search for the Shortest Path

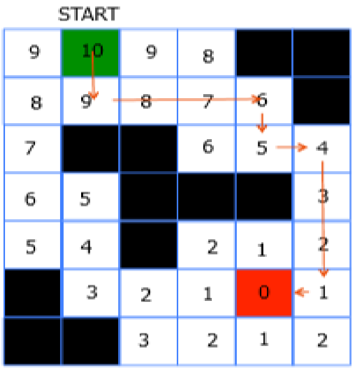
6.4.1 Grassfire Algorithm

This algorithm works like fire spreads. It starts from the goal node and expands towards the start node. At first mark the goal node with 0 value, then give its adjacent nodes a +1 value. On each iteration find all unmarked nodes adjacent to the marked nodes and give them the value distance value +1. Continue this until the start node is reached and once the start node is reached backtrack the goal node to find the path.

The computational effort required to run this algorithm increase linearly with the number of edges, which can expressed as:

$$O(|V|) \tag{6.1}$$

Where $|V|$ is the number of nodes in the graph.



9	10	9	8	■	■
8	9	8	7	6	■
7	■	■	6	5	4
6	5	■	■	■	3
5	4	■	2	1	2
■	3	2	1	0	1
■	■	3	2	1	2

Figure 6.7: Grassfire algorithm [CMP]

6.4.2 Dijkstra’s Algorithm

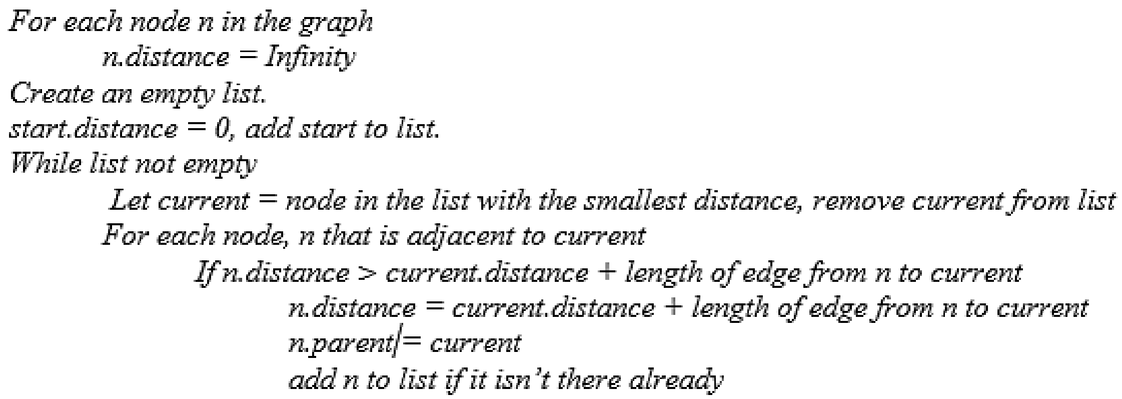
Dijkstra’s algorithm is quite an efficient algorithm to find the shortest path in the graph. It is different from minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph. This algorithm uses a greedy approach, in the sense that we find the next best solution hoping that the end result is the best solution for the whole problem.

Here, we need to store the path distance of every vertex. It can be stored in an array of size v (number of vertices). Also along with the length of the shortest path, we want to get the shortest path. For this, we map each vertex to the vertex that last updated its path length. Once the algorithm is complete, we can easily backtrack from the destination vertex to the source vertex to ascertain the path. To improve the algorithm, a minimum priority queue can be used to efficiently receive the vertex with least path distance. Normally, the time-complexity of this algorithm is $O(n^2)$, but the computational complexity with priority queue data structure is reduced to :

$$O((|V| + |E|)log|V|) \tag{6.2}$$

|V| denotes the number of nodes in the graph and |E| denotes the number of edges.

Pseudo Code for Dijkstra’s Algorithm:



```
For each node n in the graph
        n.distance = Infinity
Create an empty list.
start.distance = 0, add start to list.
While list not empty
        Let current = node in the list with the smallest distance, remove current from list
        For each node, n that is adjacent to current
                If n.distance > current.distance + length of edge from n to current
                        n.distance = current.distance + length of edge from n to current
                        n.parent = current
                        add n to list if it isn't there already
```

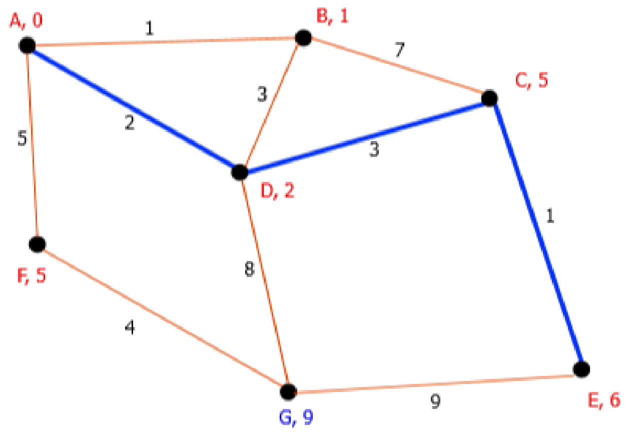


Figure 6.8: Dijkstra’s Algorithm [CMP]

6.4.3 A* Algorithm

A* algorithm is an artificial intelligence algorithm. It is an important searching algorithm that searches for the shortest path between the initial and the final state. A* is an extension of Dijkstra's algorithm and it tries to improve the performance of Dijkstra by introducing a heuristic function. It has mainly three parameters i.e.

$$\mathbf{F = G + H} \tag{6.3}$$

- F is the total cost of the node.
- G is the distance between the current node and the start node.
- H is the heuristic — estimated distance from the current node to the end node.

The way that the algorithm makes its decisions is by taking the f-value into account. The algorithm selects the smallest f-valued cell and moves to that cell. This process continues until the algorithm reaches its goal cell. To explain it doesn't explore nodes in every direction equally, instead it explores in the direction of the goal and if it encounters an obstacle, it follows the other node which can reach towards the goal.

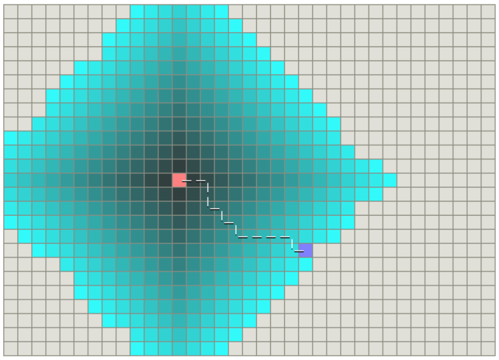


Figure 6.9: Dijkstra's Algorithm on a grid

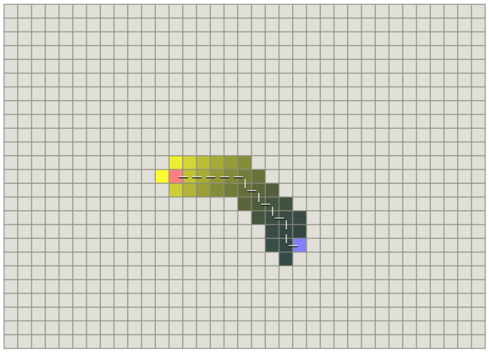


Figure 6.10: A* Algorithm on the same grid

6.5 Artificial Potential Field

The basic thinking behind is to construct a smooth function over the extent of the configuration space which has higher values when the robot is near an obstacle and lower values when it is further away from them. We also want this function to have it's lowest value at the desired goal location and it's value should increase as we move to configurations that are further away. If we can construct such a function we can use it's gradient to guide the robot to the desired configuration.

Constructing an Attractive Potential Field:

An attractive potential function, f_a(x), can be constructed by considering the distance between the current position of the robot, x = $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, and the desired goal location, x_g = $\begin{pmatrix} x_1^g \\ x_2^g \end{pmatrix}$ as follows:

$$f_a(\mathrm{x}) = \psi(\left\|x - x_g\right\|^2) \tag{6.4}$$

Here ψ is simply a constant scaling parameter.

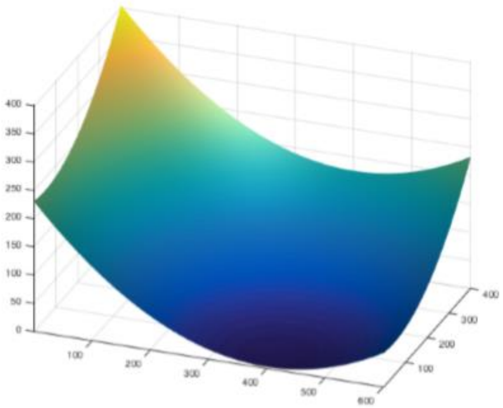


Figure 6.11: Attractive Potential field [6]

A repulsive potential function in the plane, f_r(x), can be constructed based on a function, ρ(x), that returns the distance to the closest obstacle from a given point in configuration space, x.

$$f_r(\mathrm{x}) = \left\{ \begin{array}{ll} \eta(\frac{1}{\rho(\mathrm{x})} - \frac{1}{d_0})^2 & if\ \rho(\mathrm{x}) \leq d_0 \\ 0 & if\ \rho(\mathrm{x}) > d_0 \end{array} \right\} \tag{6.5}$$

Here, η is a constant scaling parameter and d_0 is the parameter which controls the effectiveness of the repulsive potential.

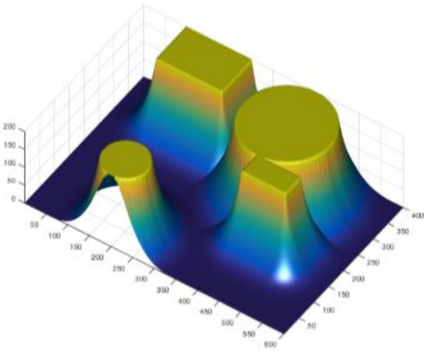


Figure 6.12: Repulsive potential field [6]

Combined potential function with gradient based control strategy:

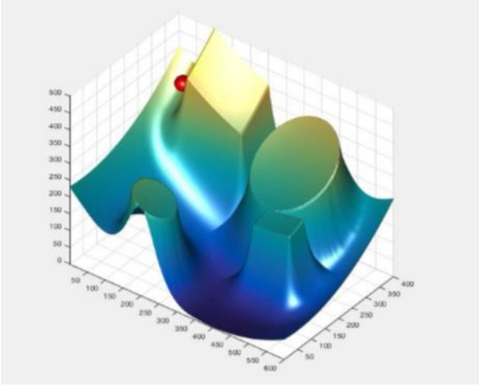


Figure 6.13: Combined Potential field [6]

6.6 Results

The visibility graph algorithm was used to generate the paths on the obstacle course because it is some good advantages over other algorithms such as:

- **Completeness:** This means that the robot will always find a path between the start and the goal no matter the complexity of the course, given that the path exists. On the contrary algorithms like Rapidly-Exploring Random Trees (RRT) and Probabilistic Road Maps (PRM) fail to find a path when the course becomes sufficiently complex or when the number of iterations are less.

- **Relatively Optimal Path:** Visibility Graphs generate relatively shorter paths in comparison to voronoi diagram and cell decomposition because it uses obstacle corners to generate edges which makes it less conservative, but efficient.

- **Dynamic Environment:** Visibility graphs are suitable for dynamic environments, i.e they can be used when the obstacles position is changed or some new obstacles are added.

Although we know that A* algorithm is faster than Dijkstra's, but when we applied both to the system, they gave almost same results and further A* took more memory than Dijkstra, so we used Dijkstra to compute the shortest path. But, in more complex systems and bigger architectures A* should definitely be preferred over the Dijkstra's algorithm.

Below figures depict the overall process of finding the shortest path. Initially, the obstacle course is feed in system, this will be received from the camera, and then the obstacles are segmented using the contour detection. Following step involves taking the Minkowski sum of the obstacle to make the configuration space. Further, the visibility graph is generated using the configuration space obstacles. And lastly the Dijkstra's algorithm was applied to find the
Shortest path in the graph.

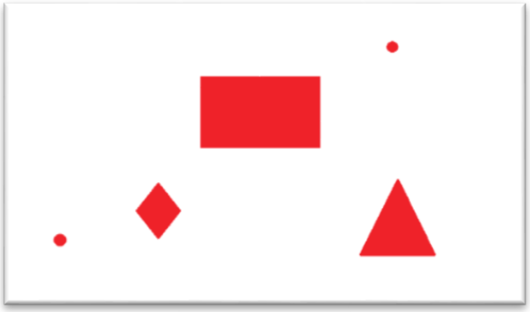


Figure 6.14: Obstacle Course

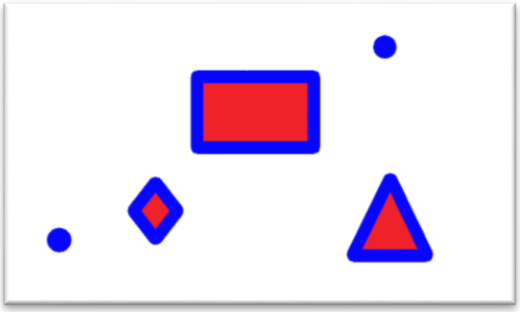


Figure 6.15: Configuration space obstacles

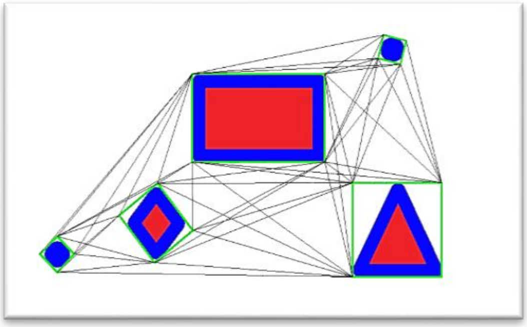


Figure 6.16: Graph generat

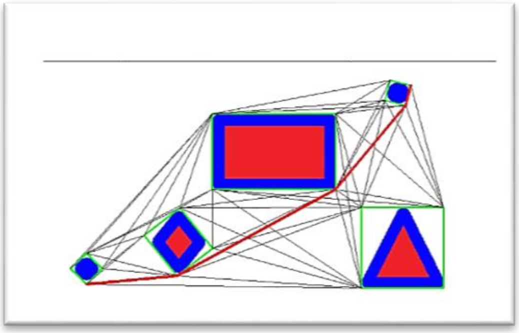


Figure 6.17: Obstacle Course with shortest path

Results on a more complex course:

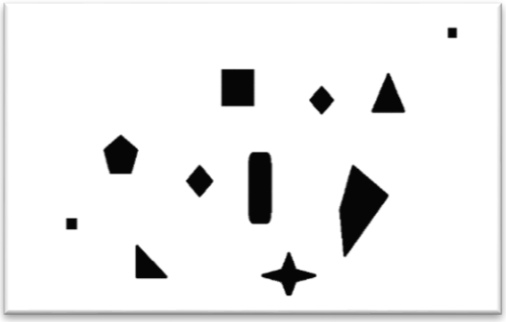


Figure 6.18: Obstacle Course

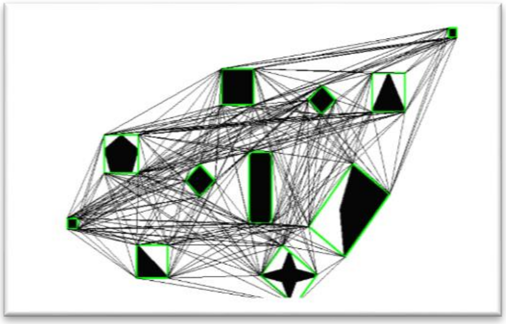


Figure 6.19: Graph generation

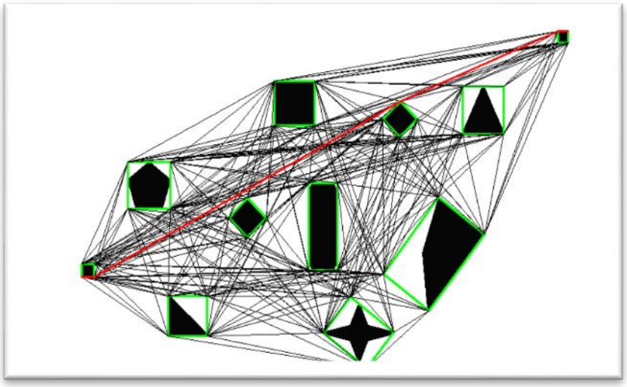


Figure 6.20: Obstacle Course with shortest path

Chapter 7
Motion Planning and Control Simulation

7.1 Motion Planning

Motion Planning is the study of the robot motion, i.e how the robot will traverse a given path. In the study of the motion planning the robot velocity curves and the acceleration curves are studied and the robot positions are decided based on temporal values. A differential drive robot has two controllable wheels mounted on its chassis as shown in figure 7.1. To control and maneuver any differential drive robot in a plane, the robot should be given a linear velocity V and a heading θ. Also, the path of the robot can be planned by controlling the velocity and orientation of the robot.

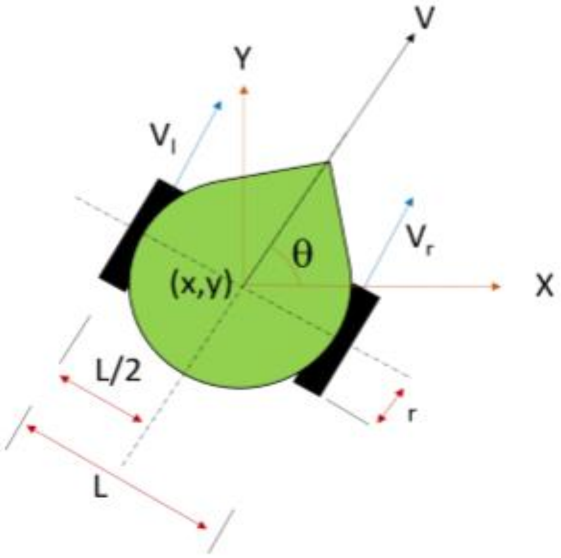


Figure 7.1: Robot parameters [8]

7.1.1 Kinematic modelling of the Robot

The inputs required for the motion of a mobile robot are the linear velocity (V) and the orientation θ. The rate of change of position of robot in x-direction is $\dot{x}$ and that in y-direction is $\dot{y}$ and the angular velocity are given by:

$$V = \frac{V_R + V_L}{2} \tag{7.1}$$

Now,

$$\dot{x} = V\cos(\theta) \tag{7.2}$$

$$\dot{y} = V\sin(\theta) \tag{7.3}$$

$$\dot{\theta} = \omega = \frac{V_R - V_L}{L} \tag{7.4}$$

Substituting the linear velocity V into $\dot{x}$ and $\dot{y}$ we get:

$$V = \sqrt{\dot{x}^2 + \dot{y}^2} \tag{7.5}$$

$$V = \frac{V_R+V_L}{2} \tag{7.6}$$

The individual velocities, V_R and V_L, can now be calculated using the equations:

$$\mathbf{V_R = V + \frac{L}{2}\omega} \tag{7.7}$$

$$\mathbf{V_L = V - \frac{L}{2}\omega} \tag{7.8}$$

The outputs, V_R and V_L, can now be used to generate the output $\dot{x}$, $\dot{y}$, ω. The actual orientation θ is fed back for error calculation, making this system a closed loop system. This error is feed into a PID controller and appropriate output is received.

7.2 Control System

7.2.1 Proportional Integral Derivative (PID) Control

PID controller is often used to produce optimum control function. Here, the PID controller is used to calculate the error between the actual position and the achieved position. This error is feed into the controller and it provides the optimum control function. The values of K_p, K_I, K_d are obtained using trial and error method. But, it is not a reliable method to compute these values, they can be computed more accurately by apps like PID tuner in Matlab by feeding the transfer function of the system.

K_p: 0.38
K_I: 0.54
K_d: 0.76

$$P = K_p e_p(t) + K_p K_D \frac{de_p(t)}{dt} + P(0) \tag{7.9}$$

In proportional controller output is directly proportional to input. In high speed application system may not behave as required due to physical characteristics of the system such as inertia, lag in response, friction, etc. Ideally it is required that system should be critically damped but it may not be attained. To change and optimize the control function other controllers are required.

In PID, the proportional term considers the current size of e(t) only at the time of the controller calculation, while the integral term considers the history of the error, in simple terms, for how long and how far the measured process variable has been from the set point during the span of the time. Integration is a process of continual summing. Integration of error over time means that the complete controller error history up to the present time is summed up.

Derivative control gives fast response as output is proportional to rate of change of input. While integral reduces steady state error as error is accumulated over time.

7.3 Pygame Simulation

Pygame module in python is used for creating games and animation in python. It is the Python wrapper for the SDL library, which stands for Simple DirectMedia Layer. SDL provides cross-platform access to your system's underlying multimedia hardware components, such as sound, video, mouse, keyboard, and joystick. This project uses Pygame to simulate the robot kinematic movements on the shortest path generated by the Dijkstra's algorithm. There is a perpetually running while loop, in which all the pygame events occur.

The code has two main functions i.e get_angle() and drive(), these functions help to get the desired angle of the robot at each iteration and the drive function provides the instantaneous X and Y velocities. The obstacle course and the shortest paths are loaded from the visibility graph code from previous chapter. Figure 7.2 and Figure 7.3 shows the robot car simulation, notice the path is not a smooth path. The circles drawn on the images shows the sharp angle turns in both the obstacle courses. Now, in real world scenarios the car cannot turn about its geometric centre, due to large turning radius the below paths are not feasible in real life. So, to solve this problem, I have applied the spline interpolation between the discrete points obtained from the algorithm.

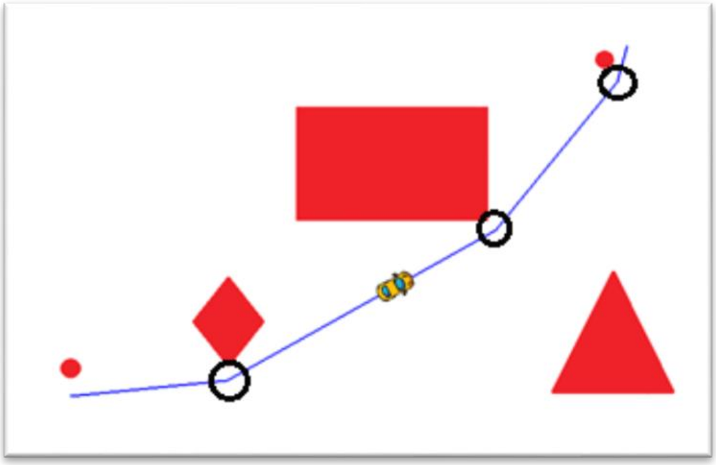


Figure 7.2: Simple Course Simulation without Interpolation

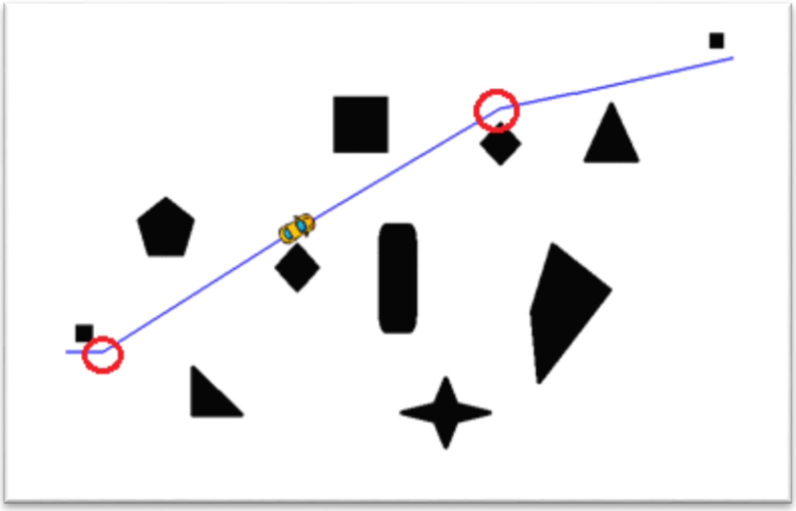


Figure 7.3: Complex Course Simulation without Interpolation

7.3.1 Spline Interpolation to Smoothen the Curve

What is a Spline?

A spline curve is a special function defined piecewise by polynomials. Splines are functions that perform an interpolation by passing exactly through the observation points. It is a very useful technique often used in Computer Graphics and Computer Aided Design (CAD). Spline Interpolation is usually preferred over the convention polynomial interpolation because the interpolation error can be made small even for low degree polynomials for the spline.

For this use-case, SciPy python module's interpolation function is used. The splines were generated with a data of 100 points and the interpolation method was selected according to the deviation from the original curve. Figure 7.4 shows the interpolation for simple geometry pygame simulation and Figure 7.5 shows the interpolation for complex geometry Pygame simulation.

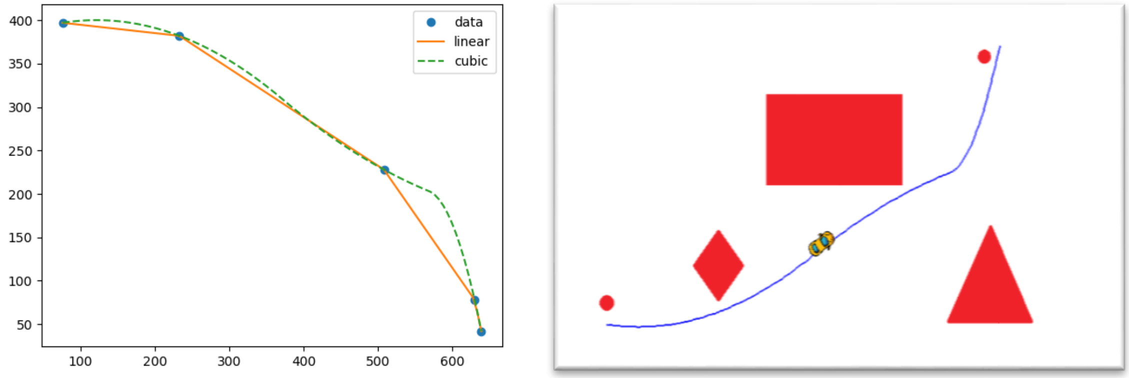


Figure 7.4: Simple Course Simulation with Interpolation

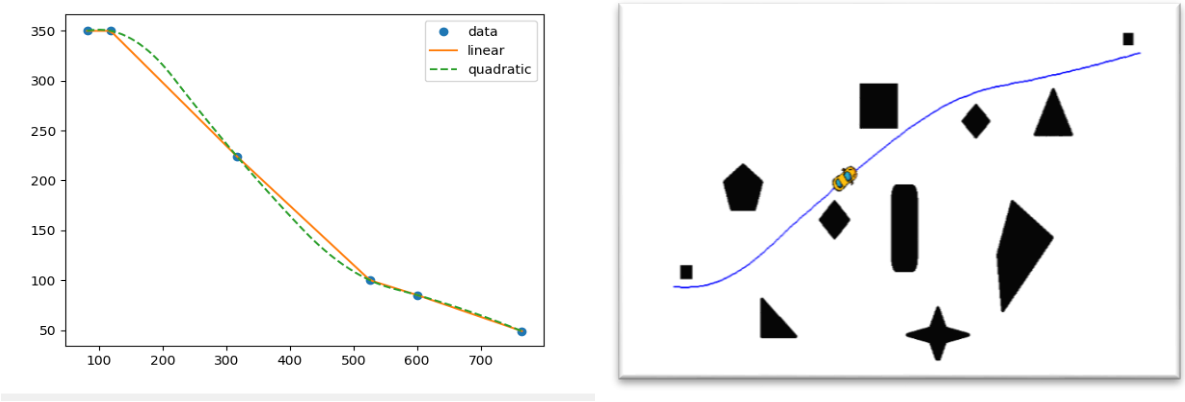


Figure 7.5: Complex Course Simulation with Interpolation

Chapter 8
Conclusion and Future Scope

8.1 Conclusion

This project implements the designing, motion planning and the control of the differential drive robot. The drive is designed using suitable constraints and the hardware selection and integration requirements are also discussed in this report. Also, the costing of the chosen parts is considered to check the economic viability of the project. Furthermore, the object segmentation and separation is carried out to build the obstacle course. The best approach to segment the obstacles is by using the perspective transform and then by applying the thresholding and contour detection on the field. Next step involves carrying out the path planning on the segmented course. After thorough review analysis and testing, visibility graph method was chosen because of its completeness and relative efficiency over other algorithms and therefore Dijkstra's search algorithm was applied on the generated graph. This step provides us with a shortest path in terms of Euclidean distance from start to goal. We used Dijkstra, instead of A* because there was not much difference in the performance for the sparse graphs generated for our courses and Dijkstra has an easy implementation code. Finally the kinematic equations of the drive were used to guide it on the shortest path found. The velocities and the angles found out using the kinematic constraints are transmitted to the drive using the socket server-client protocol. This helps to control the robot in real time i.e the simultaneous movement robot and transmission of drive parameters. Robot movements are very complex for a continuous path, so to visualize these movements a Pygame simulation (a python api for creating simulation) is made. This simulation will be very useful to compare the ideal and practical trajectories of the robot.

8.2 Future scope

Path planning is a broad field which is applied from Google Maps, Aerial robotics, Mobile Robotics to Industrial Manipulators. To expand my work, I would like to improve the path planning algorithm discussed in this report. This can be done by applying Deep convolutional Q-Learning (Q-learning is a reinforcement learning technique) to train the agent which is deployed in the obstacle field and finally the optimal path is found by training the neural network. Therefore, my aim is to enhance the path search algorithm using deep learning model, also to show the real time working (visual surveying) of the robot with PID control.

Also to expand, the camera can be deployed on the drones to achieve real time visual surveying of the obstacles on a much larger and outdoor areas. Multiple Drones can cover a huge area thus making this system adaptable to any kind of terrain.

Bibliography

[1] Steven M. LaValle, University of Illinois, *Planning Algorithms*

[2] Rosebrock, D. A. (2016). Practical Python and OpenCV: *An Introductory, Example Driven Guide to Image Processing and Computer Vision*. pyimagesearch.

[3] Subir Kumar Ghosh, *Visibility-based Robot Path Planning* -School of Technology & Computer Science Tata Institute of Fundamental Research.

[4] K. N. McGuire1∗, G.C.H.E. de Croon1 and K. Tuyls2 Delft University of Technology, The Netherlands, University of Liverpool, United Kingdom, *A Comparative Study of Bug Algorithms for Robot Navigation.*

[5] S.N. Fry, M. B. (2000). *Tracking of flying insects using pan-tilt cameras*. (p.9). Elceveir.

[6] *Robotic Motion Planning: Potential Functions* - Robotics Institute 16-735 http://voronoi.sbp.ri.cmu.edu/~motion, Howie Choset

[7] Cherry Myint, Nu Nu Win, (IJSETR) Volume 5, Issue 9, September 2016

   a. *Position and Velocity control for Two-Wheel Differential Drive Mobile Robot*

[8] Kaamesh Kothandaraman, *Motion Planning and Control of Differential Drive Robot-* Wright State University.

[9] *ESP32 Datasheet:* https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[10] *L298N Motor driver Datasheet:* https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf

[11] David T. Wooden(2006), *Graph-based Path Planning for Mobile Robots*

[12] Marcell Missura1, Daniel D. Lee2, and Maren Bennewitz, Minimal Construct: *Efficient Shortest Path Finding for Mobile Robots in Polygonal Maps*

[13] Bista, D. (2016). *Understanding and Design of an Arduino-based.* Verginia Commonwealth University.

[14] Tiago P. do Nascimento , Member, IEEE, Gabriel F. Basso , Carlos E.T.D orea , and Luiz Marcos G. Gonc¸alves , Member, IEEEVOL. 24, NO. 4, AUGUST 2019, *Perception-Driven Motion Control Based on Stochastic Nonlinear Model Predictive Controllers,*

Appendix

Appendix A: Server and Client Code Snippet

Client Side Code:

```
#include "WiFi.h"


// Setting SSID and password of the network
const char* ssid = "PARAS";
const char* password =  "123456789";

WiFiServer wifiServer(80);

// Motor 1 Pins definition
int motor1Pin1 = 27;
int motor1Pin2 = 26;
int enable1Pin = 14;

// Motor 2 Pins definition
int motor2Pin1 = 18;
int motor2Pin2 = 19;
int enable2Pin = 15;

// Initializing the PWM properties
const int freq = 30000;
const int pwmChannel = 0;
const int resolution = 8;
int dutyCycle = 500;

void setup()
{
  // sets the pins as outputs:
  pinMode(motor1Pin1, OUTPUT);
  pinMode(motor1Pin2, OUTPUT);
  pinMode(enable1Pin, OUTPUT);

  pinMode(motor2Pin1, OUTPUT);
  pinMode(motor2Pin2, OUTPUT);
  pinMode(enable2Pin, OUTPUT);


  ledcSetup(pwmChannel, freq, resolution);

  // connecting the channel to the GPIO pin used for enable
  ledcAttachPin(enable1Pin, pwmChannel);
  ledcAttachPin(enable2Pin, pwmChannel);
  Serial.begin(115200);

 //Connecting to the wifi network

  delay(2000);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.println("Connecting to WiFi..");
  }

  Serial.println("Connected to the WiFi network");
  Serial.println(WiFi.localIP());

  wifiServer.begin();

  // testing
  Serial.print("Testing DC Motor...");
}

void loop()
{

   WiFiClient client = wifiServer.available();

  if (client)
  {

    while (client.connected())
    {

      while (client.available()>0)
      {
        char c = client.read();
        Serial.println(c);

          if (c == 'w')
          {
             // Move the Robot ahead at max speed
              Serial.println("Moving Forward");
              digitalWrite(motor1Pin1, LOW);
              digitalWrite(motor1Pin2, HIGH);
              digitalWrite(motor2Pin1, LOW);
              digitalWrite(motor2Pin2, HIGH);
              delay(50);

          }
          if (c == 's')
          {

            // Move the Robot backwards at maximum speed
            Serial.println("Moving Backwards");
            digitalWrite(motor1Pin1, HIGH);
            digitalWrite(motor1Pin2, LOW);
            digitalWrite(motor2Pin1, HIGH);
            digitalWrite(motor2Pin2, LOW);
            delay(50);
          }
          if (c == 'a')
          {

            // Rotate the robot anticlockwise
            Serial.println("Rotating anticlockwise");
            digitalWrite(motor1Pin1, LOW);
            digitalWrite(motor1Pin2, HIGH);
            digitalWrite(motor2Pin1, HIGH);
            digitalWrite(motor2Pin2, LOW);
            delay(10);
          }

         if (c == 'd')
         {
          // Rotate the robot clockwise
            Serial.println("Rotating clockwise");
            digitalWrite(motor1Pin1, HIGH);
            digitalWrite(motor1Pin2, LOW);
            digitalWrite(motor2Pin1, LOW);
            digitalWrite(motor2Pin2, HIGH);
            delay(10);
         }
        if (c == 'k')
        {
         // Stop the robot
            Serial.println("Stopping the robot");
            digitalWrite(motor1Pin1, LOW);
            digitalWrite(motor1Pin2, LOW);
            digitalWrite(motor2Pin1, LOW);
            digitalWrite(motor2Pin2, LOW);
            delay(50);
        }
      }

      delay(10);
    }

    client.stop();
    Serial.println("Client disconnected");
```

```
  }
}
```

Server Side Code:

```
import socket
import pygame
import sys

sock = socket.socket()

host = "192.168.137.189"
port = 80

sock.connect((host, port))

pygame.init()

pygame.display.set_caption(u'Robot Control')

pygame.display.set_mode((400, 400))

while True:
    event = pygame.event.wait()
    if event.type == pygame.QUIT:
        break

    if event.type in (pygame.KEYDOWN, pygame.KEYUP):
        key_name = pygame.key.name(event.key)

        if event.type == pygame.KEYDOWN:
            print (u'"{}" key pressed'.format(key_name))
            message = key_name.encode()
            sock.send(message)

        elif event.type == pygame.KEYUP:
            print (u'"{}" key released'.format(key_name))
            message = 'k'.encode()
            sock.send(message)

pygame.quit()
sock.close()
sys.exit()
```

Appendix B: Opencv based Code Snippet

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
from scipy.spatial import distance


cap = cv2.VideoCapture(0)

try:
    while(1):
        _, frame = cap.read()

        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        # blur = cv2.GaussianBlur(gray,(1,1),1000)

        ret, thresh = cv2.threshold(gray, 50, 255,cv2.THRESH_BINARY_INV)

    #    _, binary = cv2.threshold(gray, 225, 255, cv2.THRESH_BINARY_INV)

        contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

        for i in range(len(contours)):
            cnt = contours[i]

            rect = cv2.minAreaRect(cnt)

            box = cv2.boxPoints(rect)
            print(box)
            box = np.int0(box)
            frame = cv2.drawContours(frame, [box] , -1, (0,255,0), 2)

        cv2.imshow('frame', frame)
        cv2.imshow('thresh', thresh)

        k = cv2.waitKey(1)
        if k == 27:
            break
except Exception as E:
    print('ERROR OCCURED')
    print (E)

finally:
    cap.release()
    cv2.destroyAllWindows()
```

Appendix C: Path planning Code Snippet

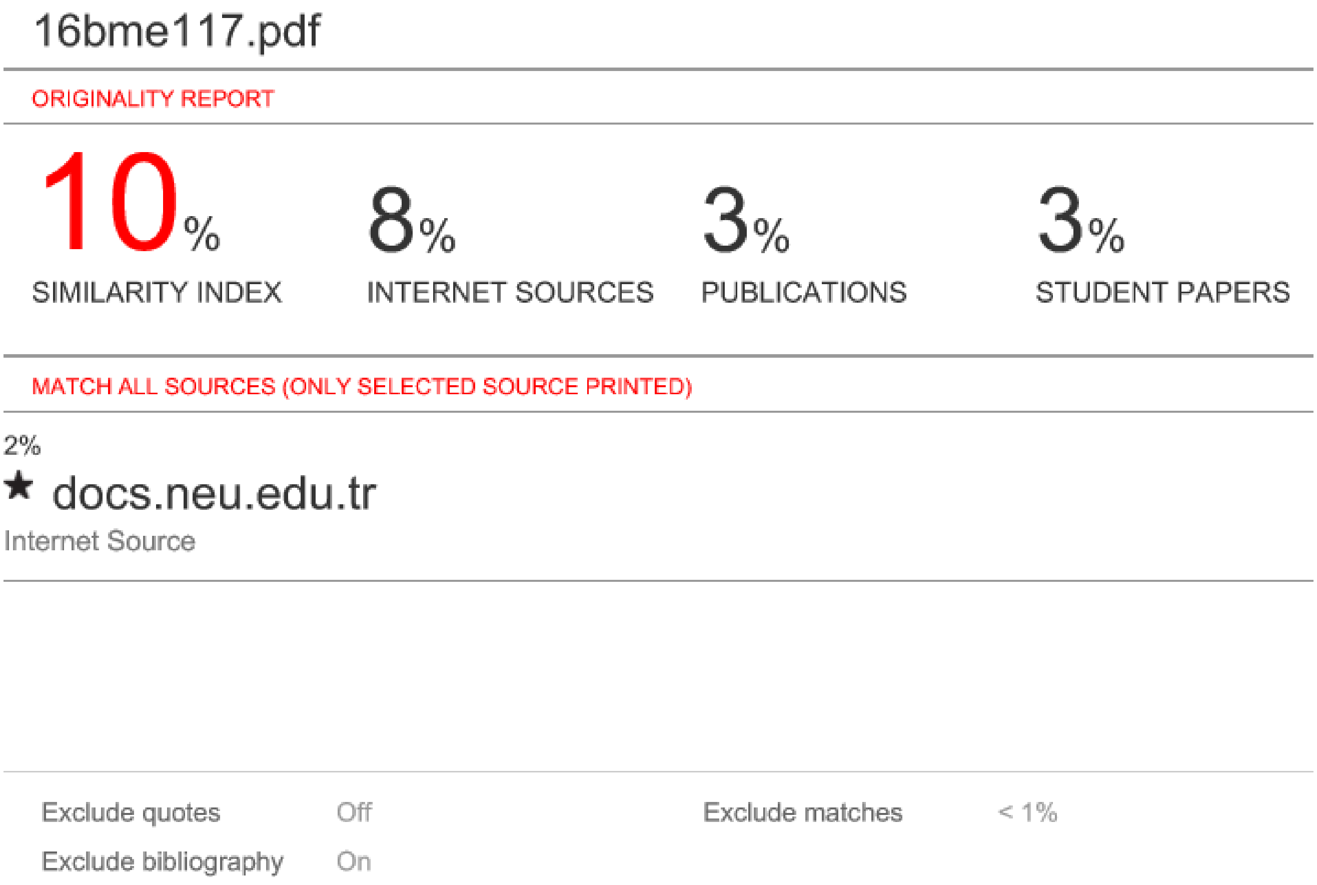
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
#from shapely import geometry

#________________________________________________________________________________

## Function to test the line intersection

def line_intersection(line1, line2):

    from shapely import geometry
    line1 = geometry.LineString(line1)
    line2 = geometry.LineString(line2)
    x = str(line1.intersection(line2))
    if 'POINT' in x:
        return True
    else:
        return False

#_______________________________________________________________________________

## Graph generation function (Creates Adjacency list form final_path_edges)

def graph_generation(final_path_edges):

    graph = {}
    for path_edge in final_path_edges:
        vertex = path_edge[0]
        graph[vertex] = {}

        if len(graph[vertex]) == 0:
            for path_edge in final_path_edges:
                if path_edge[0] == vertex:
                    distance = int(((path_edge[1][1] - path_edge[0][1])**2 + (path_edge[1][0] -
path_edge[0][0])**2)**0.5)
                    graph[vertex].update({path_edge[1]:(distance)})
                    graph[vertex].update({path_edge[1]:(distance)})

    return graph

#________________________________________________________________________________

## Obstacle Inflation Function

def obstacle_inflation(image, contours):
    for i in range(len(contours)):
        for e in contours[i]:
            image = cv2.circle(image, tuple(e[0]), 10, (255,0,0), -1)
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            ret,thresh = cv2.threshold(gray,150,255,cv2.THRESH_BINARY_INV)

    inflated_contours,_ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
    return inflated_contours

#__________________________________________________________________________________

## Dijkstra's shortest path Algorithm

Def Dijkstra(graph_structure, start_node,goal_node):

    Shortest_distance = {}
    Predecessor = {}
    Unseen_nodes = graph
    infinity = 10000000
    Path = []
    for node in Unseen_nodes:
        Shortest_distance[node] = infinity
    Shortest_distance[start_node] = 0
    while Unseen_nodes:
        min_node = None
        for node in Unseen_nodes:
            if min_node is None:
                minNode = node
            elif Shortest_distance[node] < Shortest_distance[min_node]:
                min_node = node

        for child_node, weight in graph_structure[min_node].items():
            if weight + Shortest_distance[min_node] < Shortest_distance[child_node]:
                Shortest_distance[child_node] = weight + Shortest_distance[min_node]
                Predecessor[child_node] = min_node
        Unseen_nodes.pop(min_node)
    current_node = goal_node
    while current_node != start_node:
        try:
            path.insert(0,current_node)
            current_node = Predecessor[current_node]
        except KeyError:
            print('Path cannot be reached')
            break
#     print(Shortest_distance)
    path.insert(0,start_node)
    if Shortest_distance[goal_node] != infinity:
       return Shortest_distance[goal_node], path



#____________________________________________________________________________________
```

```
if __name__=='__main__':

    image = cv2.imread('C:\\Users\\HP\\Desktop\\major proje ct\\vis_graph\\obstacle course3.png')
    # rols, cols,_ = image.shape
    # print(rols, cols)
    # image = cv2.resize(image, (800,400), interpolation = cv2.INTER_AREA)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
#    cv2.imshow('gray',gray)

    ret,thresh = cv2.threshold(gray,150,255,cv2.THRESH_BINARY_INV)
    contours,_ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

#    points = np.zeros((1,2))  # dummy starting point

    nodes = []            # nodes
    obstacles = []        # set of obstacles
    path_edges = []       # edges between nodes
    obstacle_edges = []   # obstacle edges
    final_path_edges = [] # final total edges

    inflated_contours = obstacle_inflation(image, contours)
    # cv2.imwrite('image.png',image)

    for i in range(len(inflated_contours)):
        cnt = inflated_contours[i]
        rect = cv2.minAreaRect(cnt)
        box = cv2.boxPoints(rect)

#     clockwise points in box :


#        2 _____ 3
#         |         |
#         |         |
#         |______|
#        1         4

        box = np.int0(box)
        obstacles.append(box)
#        points = np.append(points, box, axis=0)
#        print(box)

        image = cv2.drawContours(image, [box] , -1, (0,255,0), 2)
#        image1 = cv2.drawContours(image, cnt , -1, (0,255,0), 2)

#     Create nodes list

    for obstacle in obstacles:
        obstacle_edges.append([[tuple(obstacle[0]),tuple(obstacle[1])],
[tuple(obstacle[1]),tuple(obstacle[2])], [tuple(obstacle[2]),tuple(obstacle[3])],
[tuple(obstacle[3]),tuple(obstacle[0])]])
```

```
        for i in obstacle:
            nodes.append(i)

#    Create path_edges list
#    Find all edges in the graph

    for node in nodes:
        for obstacle in obstacles:
            if node in obstacle:
                continue
            else:
                for i in obstacle:
                    path_edge = [tuple(node), tuple(i)]
                    path_edges.append(path_edge)

#                    cv2.line(image, path_edge[0], path_edge[1], (0, 0 ,0), 1)    # edges including intersections

#    Find visible edges in the graph

    for path_edge in path_edges:
        counter = 0

        for obstacle in obstacle_edges:
            for obstacle_edge in obstacle:
                if line_intersection(path_edge, obstacle_edge):
                    counter += 1
        if counter == 4:
            final_path_edges.append([path_edge[0], path_edge[1]])
#              print(path_edge[0], path_edge[1])
            cv2.line(image, path_edge[0], path_edge[1], (0, 0, 0), 1)    # edges excluding intersections

# To add obstacle edges as paths in final_path_edges

    for obstacle in obstacle_edges:
        for obstacle_edge in obstacle:
            reverse = [obstacle_edge[1], obstacle_edge[0]]
            final_path_edges.append(obstacle_edge)
            final_path_edges.append(reverse)

    # print(final_path_edges)

    graph = graph_generation(final_path_edges)

    shortest_distance, path = dijkstra_algo(graph,(81, 350),(525,174))

    for i in range(len(path)-1):
        cv2.line(image, path[i], path[i+1], (0, 0, 255), 3)

    print('Shortest path = ', path)
    print()
    print('Shortest distance = ', shortest_distance)
    # cv2.imshow('image',image)
    plt.imshow(image)
    plt.show()
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Appendix D: Pygame Simulation Code Snippet

```
import cv2
import numpy as np
import pygame
import sys
import math
import time
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d


def robot(robot_image,x, y,angle):
    orig_rect = robot_image.get_rect()
    rot_image = pygame.transform.rotate(robot_image, angle)
    rot_rect = orig_rect.copy()
    rot_rect.center = rot_image.get_rect().center
    rot_image = rot_image.subsurface(rot_rect).copy()

    # To update pos of the robot
    screen.blit(rot_image, (x, y))

def get_angle(sp, i):
   s = math.atan2(sp[i+1][1]-sp[i][1], sp[i+1][0]-sp[i][0])
   angle = -s*180/math.pi
   # angle = round(angle,2)
   return angle

def drive(sp, i):
    global angle_change
    global x_change
    global y_change
    angle_change = 1
    x_change = (sp[i+1][0]-sp[i][0])/5
    y_change = (sp[i+1][1]-sp[i][1])/5
    x_distance = sp[i+1][0]-sp[i][0]
    y_distance = sp[i+1][1]-sp[i][1]
    return angle_change, x_change, y_change, x_distance, y_distance

def spline_fit(sp):
    x = []
    y = []
    nsp = []
    for i in range(len(sp)):
        x.append(sp[i][0])
        y.append(sp[i][1])

    f1 = interp1d(x, y, kind ='linear')
    f2 = interp1d(x, y, kind='quadratic')

    xnew = np.linspace(sp[0][0], sp[-1][0], num=100, endpoint=True)

    for (i,j) in zip(xnew, f2(xnew)):
        i = round(i, 3)
        j = round(j, 3)
        nsp.append((i,j))

    return nsp
#__________________________________________________________

pygame.init()
image = cv2.imread('obstacle course3.png')
rows, cols, _ = image.shape

sp = [(81, 350), (118, 350), (317, 224), (525, 174)]#[(81, 350), (118, 350), (317, 224), (526, 100), (601,
85), (765, 49)] #[(76, 397), (233, 382), (508, 228), (630, 78), (639, 42)]

sp = spline_fit(sp)

screen = pygame.display.set_mode((cols, rows))
screen.fill((0,0,0))
# pixarr = pygame.PixelArray(screen)
# pixarr[30][50] = (255,255,255)
course = pygame.image.load('obstacle course3.png')

# Robot Variables
robot_image = pygame.image.load('car.png')
size = (40, 40)
robot_image = pygame.transform.scale(robot_image, size)

## Starting parameters
x = sp[0][0]- size[0]/2
y = sp[0][1]- size[1]/2
angle = 0
final_angle = get_angle(sp, 0)
angle_change, x_change, y_change, x_distance, y_distance = drive(sp, 0)

time.sleep(2)

## Simulation Loop
while True:

    screen.blit(course, (0, 0))
    # End Condition
    if x >= (sp[0][0] - size[0]/2) + sp[-1][0]- sp[0][0]:
        x = (sp[0][0] - size[0]/2) + sp[-1][0]- sp[0][0]
        y = (sp[0][1]- size[1]/2) +  sp[-1][1]- sp[0][1]

    for e in range(len(sp)-1):
        pygame.draw.line(screen, (0, 0, 255), sp[e], sp[e+1], 2)

    for i in range(len(sp)-1):
        if x < sp[i+1][0] - size[0]/2:
            break

    final_angle = get_angle(sp, i)
    # print(final_angle)
    angle_change, x_change, y_change, x_distance, y_distance = drive(sp, i)

## Angle Rotation and drive translation

    if angle != final_angle:
        if abs(final_angle - angle) >= 1:
            if angle < final_angle:
                angle += angle_change
            else:
                angle -= angle_change

        else:
            if angle < final_angle:
                angle += (final_angle - angle)
            else:
                angle -= (angle- final_angle)
    else:
        if x_change > 0:
            x += x_change
        else:
            x -= x_change
        if y_change > 0:
            y -= y_change
        else:
            y += y_change

    robot(robot_image, x, y, angle)

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
    pygame.display.update()
```



16bme117.pdf

ORIGINALITY REPORT

SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS
10%	8%	3%	3%

MATCH ALL SOURCES (ONLY SELECTED SOURCE PRINTED)

2%

★ docs.neu.edu.tr

Internet Source

Exclude quotes	Off	Exclude matches	< 1%
Exclude bibliography	On		