**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



**Certificate of Internship**

We present this certificate to **Mr. Paras Savnani** student of Institute of Technology, Nirma University (Department of Mechanical Engineering). He has completed **6 months** of internship as **Robotics Software Development Intern** for below listed project.

**Project: Path Planning and Control of a Differential Drive Using Computer Vision**

**Duration: 1 Jan 2020 – 30 June 2020**

We appreciate his excellent efforts in executing the project given to him as well as his learning attitude, for exploring new technologies for completion of the task given. We wish him all the very best for his future endeavors.

For,

Fero DWC-LLC





**Jay Pandya**

(Head Manager Tech. team India, Fero DWC-LLC.)

C:\Users\HP\Desktop\Capture.PNG

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**](https://www.pyimagesearch.com/author/adrian/)(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

[Declaration iii](#_Toc512987047)

[Undertaking for Originality of the Work iv](#_Toc512987048)

[Certificate v](#_Toc512987049)

Certificate of Internship vi

[Approval Sheet vi](#_Toc512987050)i

[Acknowledgments viii](#_Toc512987051)

[Abstract ix](#_Toc512987052)

[Table of Contents x](#_Toc512987053)

List of Figures xii

List of Tables xiv

[Chapter 1 INTRODUCTION 1](#_Toc512987054)

1.1 Company Profile 1

[1.2 Motion Planning 1](#_Toc512987055)

[1.3 Computer Vision (CV) 2](#_Toc512987056)

[1.4 Applications 4](#_Toc512987057)

[1.5 Objectives 4](#_Toc512987058)

[Chapter 2 LITERATURE REVIEW 5](#_Toc512987059)

[2.1 Graph-based Path Planning for Mobile Robots 5](#_Toc512987060)

[2.2 A Comparative Study of Bug Algorithms for Robot Navigation 6](#_Toc512987061)

[2.3 Reactive object tracking with a single PTZ cam 7](#_Toc512987062)

[2.4 Robotic Motion Planning: Potential Functions 8](#_Toc512987065)

[2.5 Speeding Up A\* Search on Visibility Graphs Deﬁned over Quadtrees to Enable Long Distance Path Planning for Unmanned Surface Vehicles](#_Toc512987066) 8

[2.6 Design and Control for Differential Drive Mobile Robot 9](#_Toc512987067)

[2.7 Motion Planning and Control of Differential Drive Robot 9](#_Toc512987068)

[2.8 Summary 10](#_Toc512987069)

[Chapter 3 DESIGN AND ASSEMBLY 11](#_Toc512987070)

[3.1 Modelling and Design 11](#_Toc512987071)

[3.2 Assembly 12](#_Toc512987073)

[3.2.1 Basic parts cost estimation 13](#_Toc512987074)

[3.2.2 Part and its specifications 14](#_Toc512987075)

[3.2.3 Solidworks parts and Assembly 18](#_Toc512987077)

[Chapter 4 HARDWARE INTEGRATION AND COMMUNICATION 19](#_Toc512987078)

[4.1 Microcontroller 20](#_Toc512987079)

[4.2 Motor Control using motor driver 21](#_Toc512987080)

4.3 Hardware Integration 21

[4.4 Socket Server-Client Connection Model 23](#_Toc512987081)

[Chapter 5 OBSTACLE DETECTION USING COMPUTER VISION 24](#_Toc512987082)

[5.1 Programming 24](#_Toc512987083)

[5.1.1 Why Python 24](#_Toc512987084)

[5.2 Obstacle Segmentation 24](#_Toc512987085)

[5.3 Perspective Transform 26](#_Toc512987086)

[5.4 Colour filtering 27](#_Toc512987087)

[5.5 Contour detection 28](#_Toc512987088)

[5.6 AR-Markers 29](#_Toc512987089)

[Chapter 6 PATH PLANNING OF THE ROBOT 31](#_Toc512987096)

[6.1 Path Planning 31](#_Toc512987097)

6.2 Configuration Space 31

[6.3 Graph generation Techniques 32](#_Toc512987098)

6.3.1 Visibility Graph 32

6.3.2 Voronoi Diagram 33

6.3.2 Cell Decomposition 34

[6.4 Graph Search for shortest path 35](#_Toc512987101)

6.4.1 Grassfire Algorithm 35

6.4.2 Dijkstra's Algorithm 36

6.4.3 A\* Algorithm 37

6.5 Artificial Potential Field 37

6.6 Results 39

**Chapter 7 MOTION PLANNING AND CONTROL SIMULATION 42**

[7.1 Motion Planning 42](#_Toc512987097)

7.1.1 Kinematic Modelling 42

[7.2 Control System 43](#_Toc512987101)

7.2.1 PID Control 43

7.3 Pygame Simulation 44

7.3.1 Spline Interpolation to smoothen the curve 45

[Chapter 8 CONCLUSION AND FUTURE SCOPE 47](#_Toc512987104)

[8.1 Conclusion 47](#_Toc512987105)

[8.2 Future scope 47](#_Toc512987106)

[BIBLIOGRAPHY 48](#_Toc512987107)

**APPENDIX 49**

Appendix A: Server and Client Code Snippet 49

Appendix B: Opencv based Code Snippet 53

Appendix C: Path planning Code Snippet 54

Appendix D: Pygame Simulation code snippet 59

**List of Figures**

[Figure 1.1 Fero’s Scope of Work 1](#_Toc512989394)

Figure 1.2 Alpha 1 puzzle 2

[Figure 1.3 : Pixel representation on a grayscale image 3](file:///C:\Users\HP\Desktop\major%20proje%20ct\ref%20report\Report5.doc#_Toc512989395)

Figure 2.1: Visibility graph generation and reduction 5

Figure 2.2: Different Bug Algorithms 6

Figure 2.3: Reactive tracking with camera 7

Figure 2.4: Artificial Potential Function 8

Figure 2.5: Transfer function of the DC motor 9

Figure 2.6: System Integration used in the paper 10

Figure 3.1: Assembled tank drive 11

Figure 3.2: Overall dimensions of the design 12

Figure 3.3: Chassis 19

Figure 3.4: Free wheel 19

Figure 3.5: DC motor 19

Figure 3.6: Complete Assembly 19

Figure 3.7: Track 19

Figure 3.8: Driver wheel 19

Figure 4.1: Pinout of ESP32 20

Figure 4.2: L298N pinout 21

Figure 4.3: Schematic Connection of hardware 22

Figure 4.4: Server- Client Interaction 23

Figure 5.1: Haar-cascade for face detection 26

**Figure 5.2: Perspective Transform of a rectangular object 26**

Figure 5.3: Original image for Blob detection 27

Figure 5.4: Masked image for blob detection 27

Figure 5.5: Blob detection 28

Figure 5.6: Obstacle Course 28

Figure 5.7: Obstacle course with contours 29

Figure 5.8: A simple AR marker 29

Figure 5.9: Detection of the AR marker 30

Figure 6.1: Robot in a 2D space and a configuration space obstacle 31

Figure 6.2: Robot in a 2D space and configuration space obstacles 32

Figure 6.3: Visibility Graph generation 33

Figure 6.4: Voronoi Diagram 33

Figure 6.5: Exact Cell Decomposition 34

Figure 6.6: Approximate Cell Decomposition 35

Figure 6.7: Grassfire Algorithm 35

Figure 6.8: Dijkstra’s Algorithm 36

Figure 6.9: Dijkstra’s algorithm on a grid 37

Figure 6.10: A\* algorithm on same grid 37

Figure 6.11: Attractive Potential field 38

Figure 6.12: Repulsive Potential field 39

Figure 6.13: Combined Potential field 39

Figure 6.14: Obstacle Course 40

Figure 6.15: Configuration space obstacles 40

Figure 6.16: Graph generation 40

Figure 6.17: Obstacle course with shortest path 40

Figure 6.18: Obstacle Course 41

Figure 6.19: Graph generation 41

Figure 6.20: Obstacle course with shortest path 41

Figure 7.1: Robot parameters 42

Figure 7.2: Simple Course Simulation without Interpolation 44

Figure 7.3: Complex Course Simulation without Interpolation 45

Figure 7.4: Simple Course Simulation with Interpolation 45

Figure 7.5: Complex Course Simulation with Interpolation 46

**List of tables**

[Table 1: Primary parts for differential drive 13](#_Toc512987108)

[Table 2: Parts and its specifications 14](#_Toc512987109)

# 

# Introduction

## 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

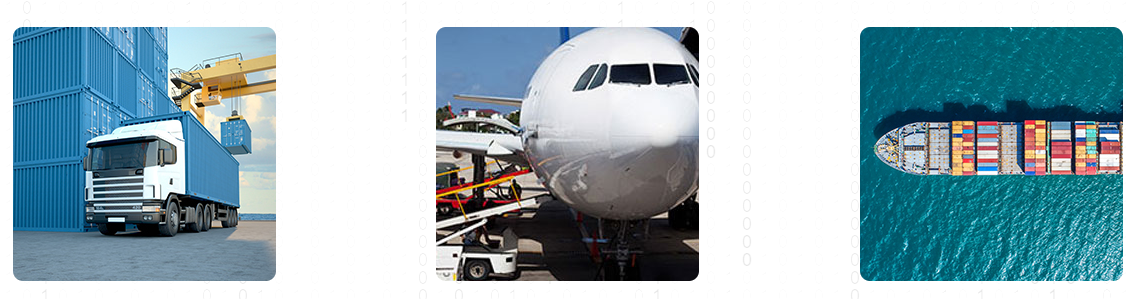


Figure1.1: Fero’s Scope of Work

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.

## 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.

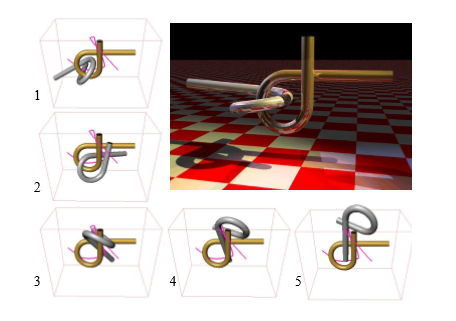


Figure 1.2: Alpha 1 puzzle [1]

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.

## 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.

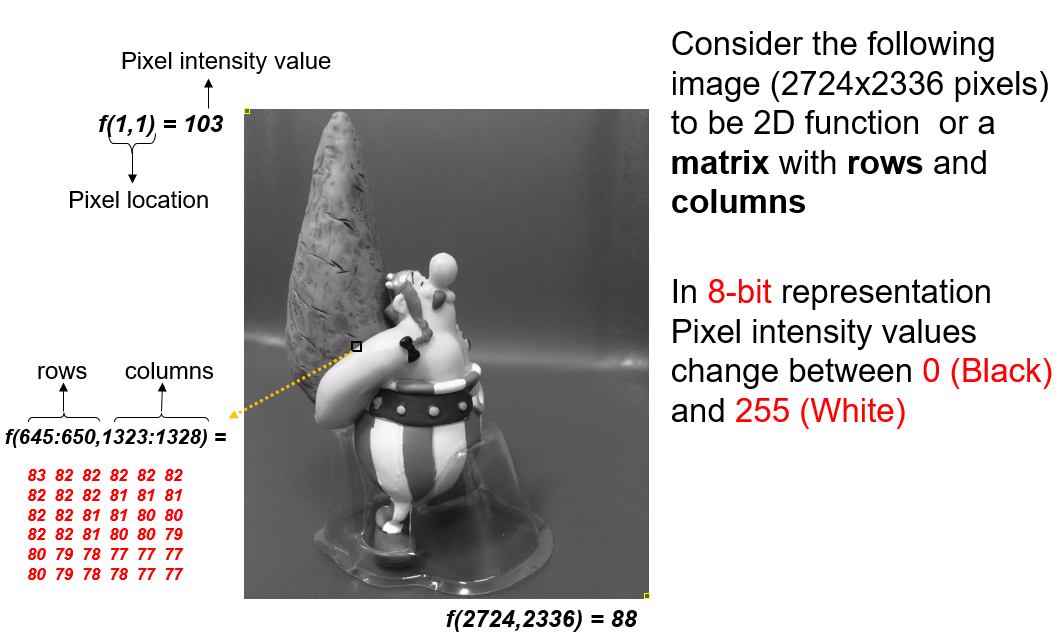


Figure 1.3: Pixel Representation on a grayscale image [2]

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.

## Applications

* Mobile Robots
* Military and defense
* Surveillance
* Monitoring
* Baby monitoring and pet monitoring.
* Autonomous vehicles
* Research

and anywhere where tracking is required.

## 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

# 

# 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.

## 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 dynamictpath planning in the plane, as well as at a4higher-level, over colored graphs, and mostly within the context of the robot controlearchitecture.

The roadmapeconstruction with a visibility graph based approach for unstructuredepolygonal 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 suboptimalrpaths in the output roadmap. The actual world experiments show the advantages of the suggested method in time-criticalwoutdoor applications where the perception is based solely on stereo-basedrelevation 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.

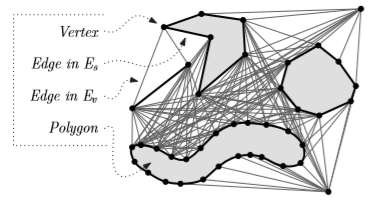
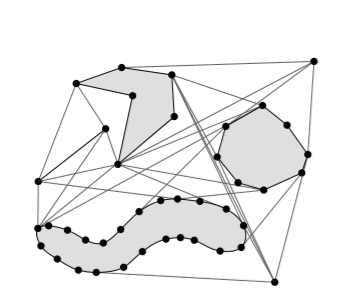
 

Figure 2.1: Visibility graph generation and reduction [3]

## 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 Algorithmsein a simulated robotrenvironment where they experience the diﬀerent types of noise and cases of failure of their on-boardesensors. From the software simulationyresults, one can conclude that the implementedyBug Algorithm’s performanceshare very sensitive to certain types ofesensor-noise, which was mostly noticeable when the odometry-drift occurs.

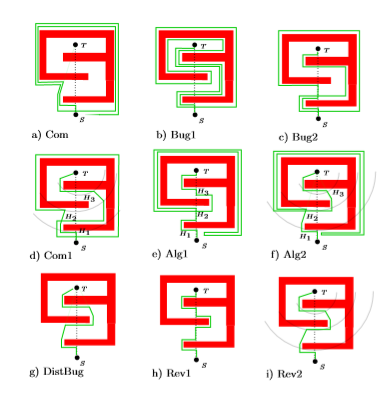
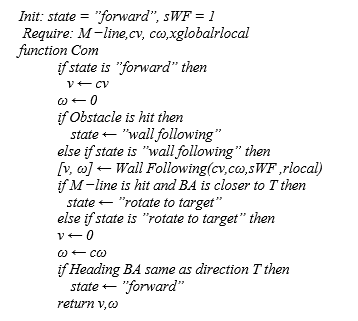


Figure 2.2: Different Bug Algorithms [12]

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 perfectton-board sensors. Also, the simulation3experiments evaluated the performance of several Bug Algorithms withdvarying noise measurements and failurercases, which clearly showed a signiﬁcant performancewdegradation 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-boardrsensors without any external help. To enhance these techniques, the experimental results showed that simplicity is a key element, as the most fundamental BugfAlgorithm, Com, was the one that was the mosttresilient to odometrymdrift and showed most robustness. Another vital element is arloop detectionrsystem, where the robot reies on multiple measured variables instead of one, especially inerealistic, noise-inducing,renvironments. Therefore these observations allow bug algorithms to be suitable for thewautonomous navigation of small robotic platforms with limitedecomputational resourses.

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

****

## 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.



Figure 2.3: Reactive tracking with camera [5]

## Robotic Motion Planning: Potential Functions

(Howie Choset, RI 16-735):

**A simple idea:**

* Supposewthe goal is aspoint g∈ℜ
* Supposewthe robot is aepointwr ∈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 therbowl and the spring analogiesrare ways of storingepotential energy

* The robot moves to a lower energy point to stabilize itself
* A potentialdfunction is a function U(q) : ℜm→ℜ
* The energy is minimized when the robot follows the negativeegradient of the potential energy function.
* A vectorsfield 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.

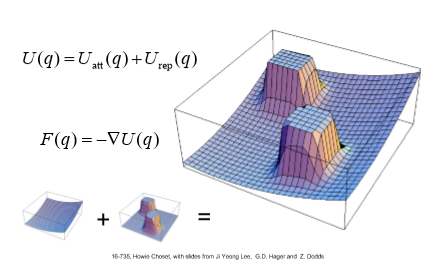


Figure 2.4: Artificial potential function [6]

## 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 graphstraversal and path search algorithm, which is often used due to its completeness,roptimality, feasibility and efficiency. The performance of the A\* path planning algorithmddepends on the estimation of the heuristic cost from the current-state to the goal-state. If the A\* path plannereunderestimates the heuristic, then it has to expandrmore nodes until it ﬁnds anroptimal path to the goal. Otherwise, if the path planner overestimates the heuristic, then the h-cost iseinadmissible and the paths found are sub-optimal.

## Design and Control for Differential Drive Mobile Robot

The differential drive robot has twodfixed powered-wheels mounted on the left and right side of the robot and it has two wheels which are independentlyrdriven. 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 curveddpath along the arcdof an instantaneousecircle. 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 PIDfcontroller based on kinematic modelling is used to control the RPM of the DC motors thus the velocity and rotation of the platform.

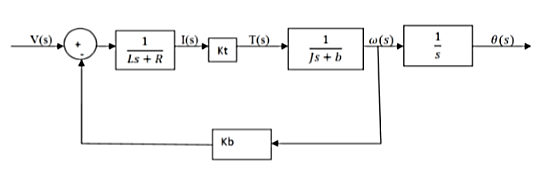


Figure 2.5: Transfer function of the DC motor [7]

## Motion Planning and Control of Differential Drive Robot

(Kaamesh Kothandaraman, 2016): This thesis studies thedpath 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 workspacescan be achieved by controlling the velocity and orientation of the robot precisely. The atan2 function was useful in driving the robot to a point, drawingeshapes and pursuingsother moving object. Systems such as Viconrcamera system, QUARC controller, XbeesqandsRedbots were used in the experiments. Also, these systems wereeconﬁgured forereal-time datadcommunication.

Many diﬀerent controllers such as PID were tested, but a simple proportionalecontroller provided optimum results compared to other complicateddcontrollers. 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 asfobstacle-avoidance, leader follower and pursuer evader.

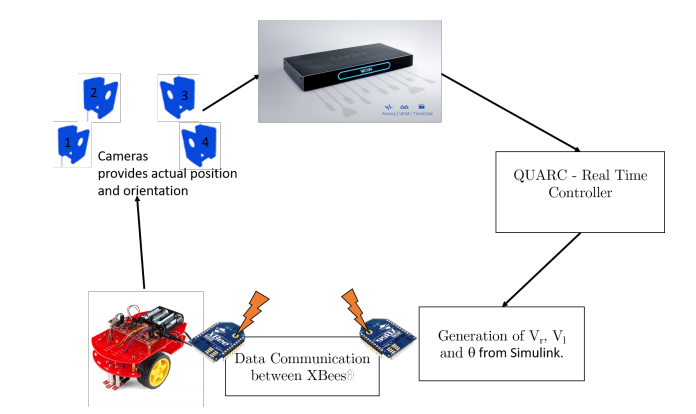


Figure 2.6: System Integration used in the paper [8]

## Summary

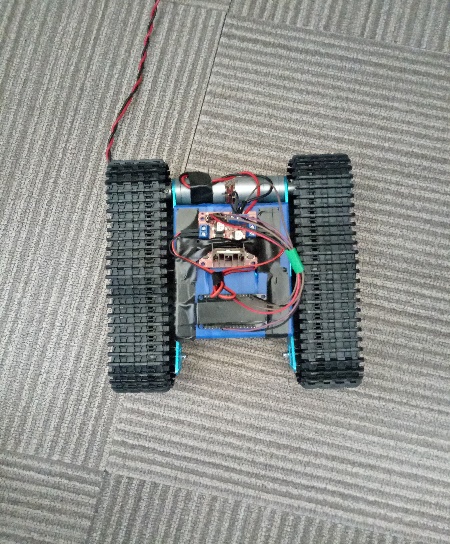
After detailed observation of the research in this topic, and study of various path planning algorithms like bug algorithms, Artificial Potential field method, Visibility Graph method, Voronoi Diagram, Cell Decomposition, RRT etc., it was decided to use the visibility graph algorithm with a search algorithm 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.

# 

# 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.



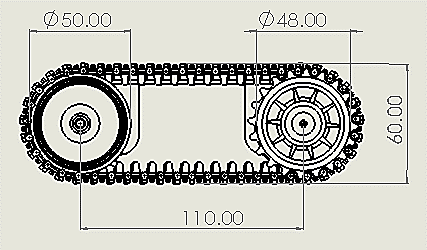
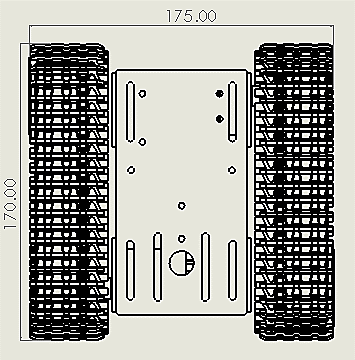
|  |  |
| --- | --- |
|  |  |

Figure 3.1: Assembled tank drive

## 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.



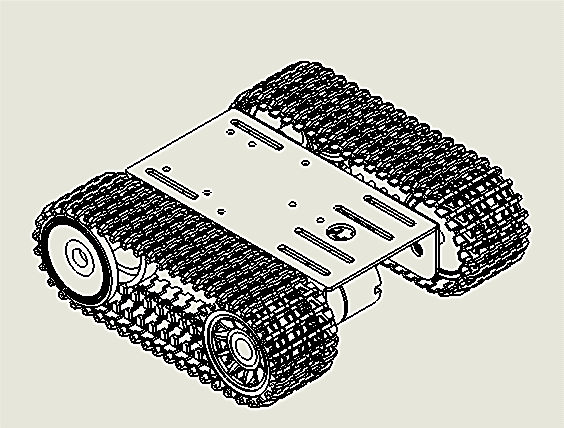


Figure 3.2: Overall dimensions of the design

## Assembly

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

### 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

### 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 × I²S interfaces  2 × I²C 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 | Amazon.in: Buy Logitech C310 HD Webcam, HD 720p/30fps, Widescreen ... | 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 | https://cdn.shopify.com/s/files/1/0014/4313/5560/products/1_bc581b8e-00ce-4f4a-b625-04273597fe3d_1800x1800.jpg?v=1577794110 | **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 | C:\Users\HP\Desktop\major proje ct\vis_graph\IMG20200320132139.jpg | **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

### 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.

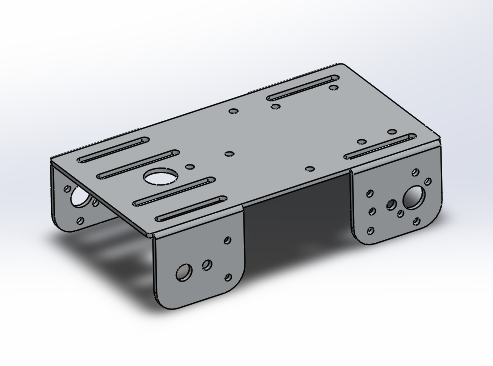
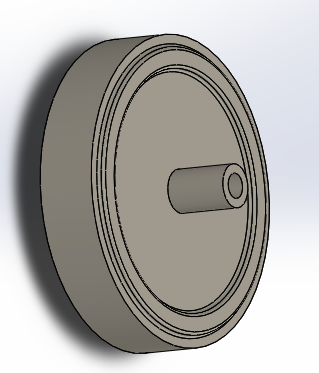
 

Figure 3.3: Chassis Figure 3.4: Free wheel

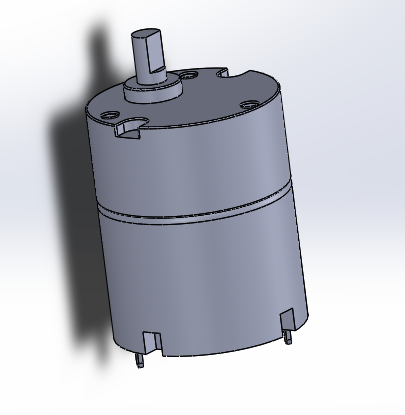
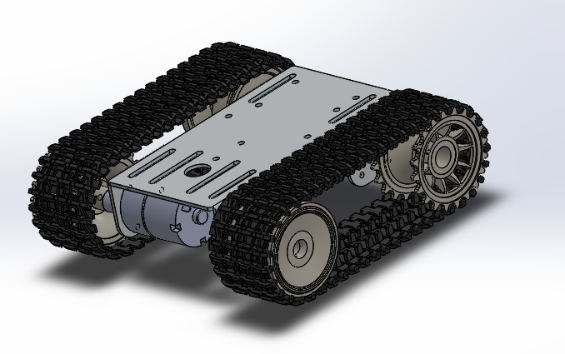
 

Figure 3.5: DC motor Figure 3.6: Complete Assembly

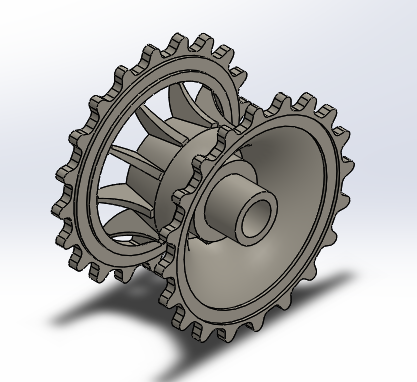
 

Figure 3.7: Track Figure 3.8: Driver wheel

# 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.

## Microcontroller

A microcontroller is a small chip having onboard processor, RAM, IO pins, Memory etc. The microcontrollerrcontrols the system bygexecuting thehcommands written in the code and issuing them to the motorjdriver. 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.

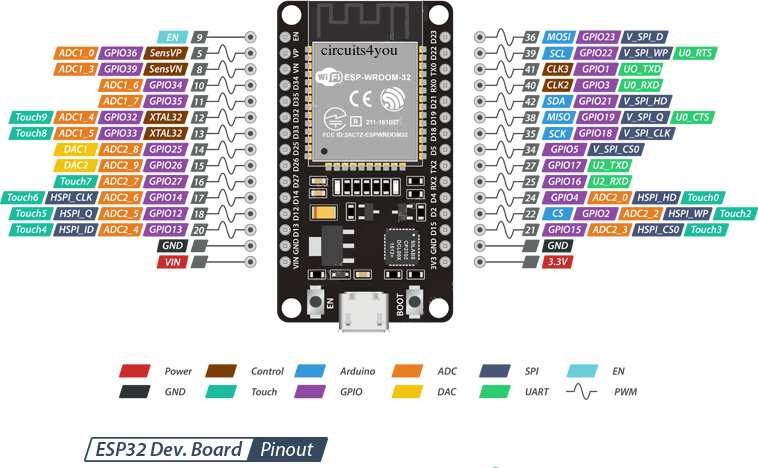


Figure 4.1: Pinout of ESP32 [9]

## 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 moreginterpolations between the zero and maximum RPM range of the motor, thus providing superiorgcontrol 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.

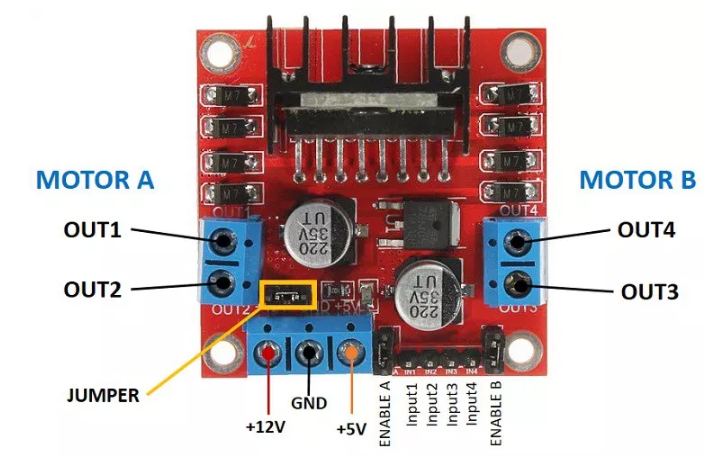


Figure 4.2: L298N pinout [10]

## 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**dLOW 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 bigqjump in current to move, so the motorslshould 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 switchwbetween the battery holder and the motorqdriver is optional, but it is very handy to cut and apply power. This way you don’t need to constantly connect and thenfdisconnect the wiring to save power. Also a 0.1uF ceramicdcapacitor is soldered to the positive and negativeeterminals of the DC motor, as shown in the diagramlto help smoothmout any voltagekspikes.

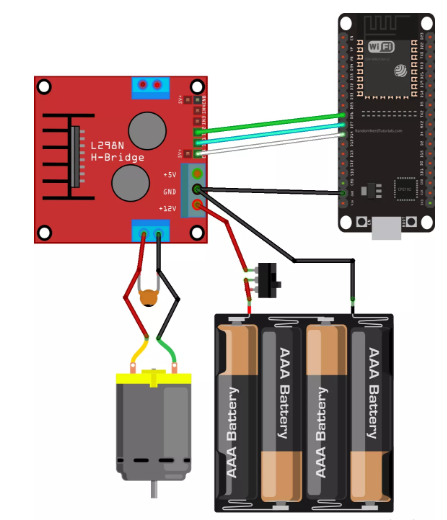


Figure 4.3: Schematic Connection of hardware

## Socket Server-Client Connection Model

Now, to transmit the velocity values and the angles to the robot we need some sort of connection between 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 theeserver 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 communicationsechannel 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.

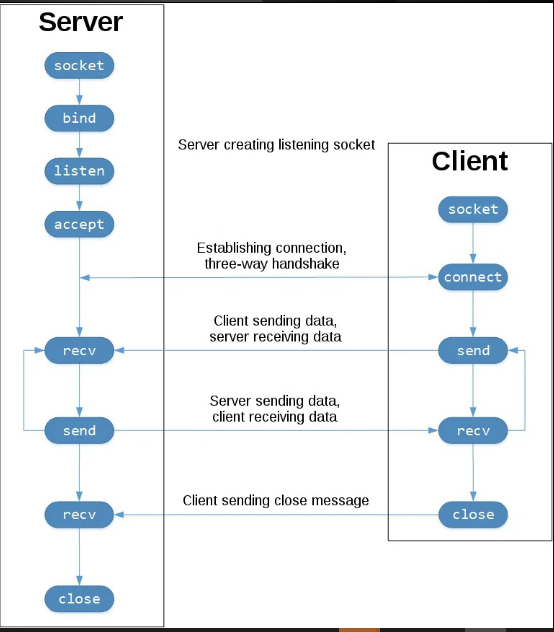


Figure 4.4: Server Client Interaction

# 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.

## Programming

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

### 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.

1. **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.

1. **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++.

1. **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.

## 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 learningtalgorithm 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) Crossqvalidation 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 giveswnegative. It is similar to howyhumans 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 trainingais 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 learningwit 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 trainingrexample, 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.

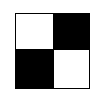
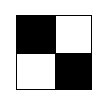
1. Edge feature

1. Line feature

1. 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.

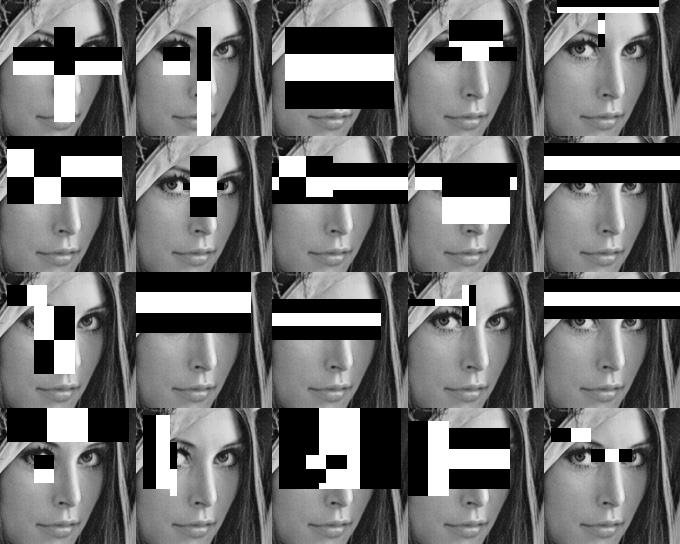


Figure 5.1: Haar-cascade for face detection [2]

## 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]**

## 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.

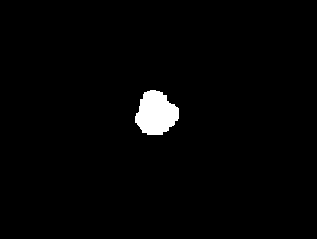


Figure 5.4: Masked image for blob detection

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.

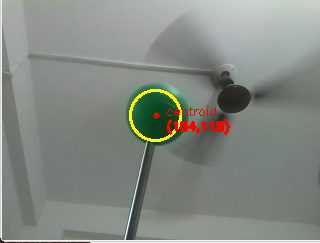


Figure 5.5: Blob detection

## 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 shapezanalysis 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().**

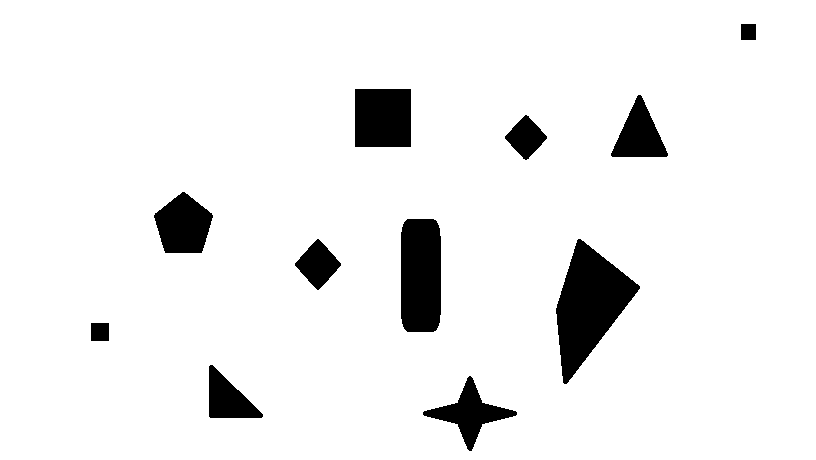


Figure 5.6: Obstacle Course

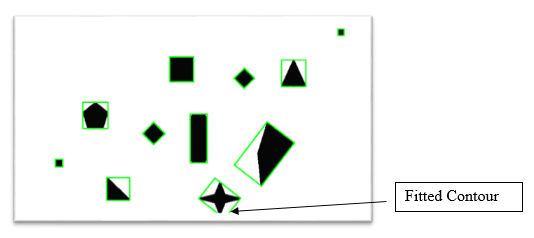


Figure 5.7: Obstacle course with contours

## 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 thewcontrast between them can be properlydrecognized 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 simpleeimages that are still readeproperly by a camera.

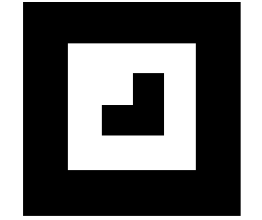


Figure 5.8: A simple AR marker

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.

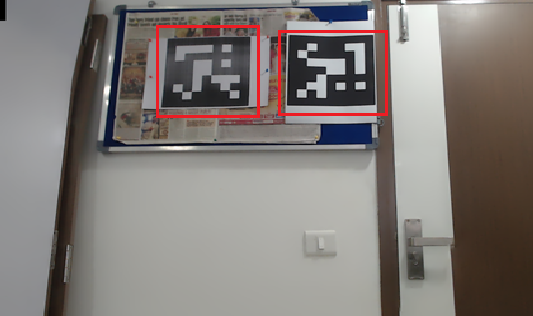


Figure 5.9: Detection of the AR marker

# Path planning of the robot

## Path Planning

A rudimentary need inerobotics is to have algorithms that convert high-levelespeciﬁcations of tasks fromehumans into low-leveledescriptions 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?

## 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 spacegis a handyhmathematical and conceptualftool which was developedjto help us think about some kinds of problems in a unifiedhframework. 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

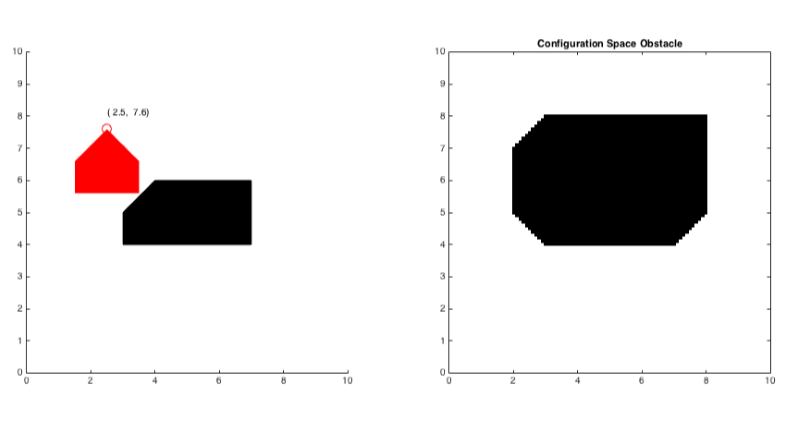


Figure 6.1: Robot in a 2D space and a configuration space obstacle [11]

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.

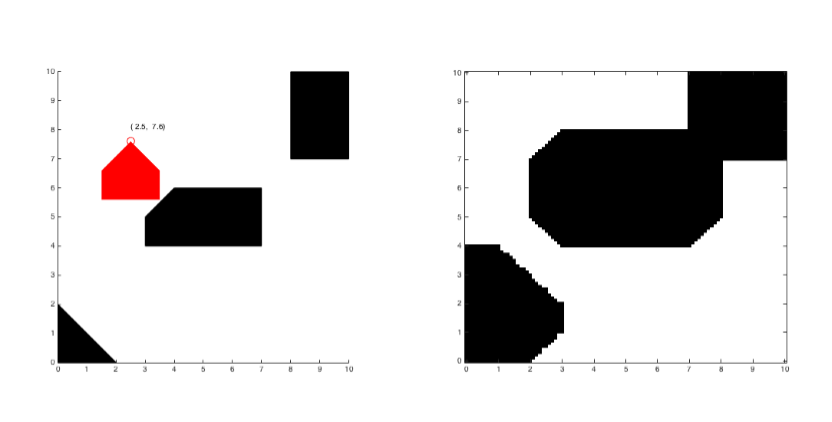


Figure 6.2: Robot in a 2D space and configuration space obstacles [11]

## Graph Generation Techniques

### Visibility Graph

A visibilitywgraph is a graphwofeintervisible locations, typically for a setwof points and obstacles in thewEuclidean plane. Eachenode in the graph represents aepoint location, and eacheedge represents a visibleeconnection between them. That is, if the linefsegment connecting two locations does not pass through any obstacle, an edgefis drawn between them in the graph. When thefset of locationsglies 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 thershortest path from start to goal that avoidsethe 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 polygonalhobstacles 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, whose edges do not intersect (total of N)

- Joinfthe givenfvertex to all other vertices (which has a total of N-1).

- Furthermore, check if the line segment intersectsgthe given N edges.

- Keep repeating the proceduregfor every vertex (total of N).

- This algorithm has a time complexity of *O()*.

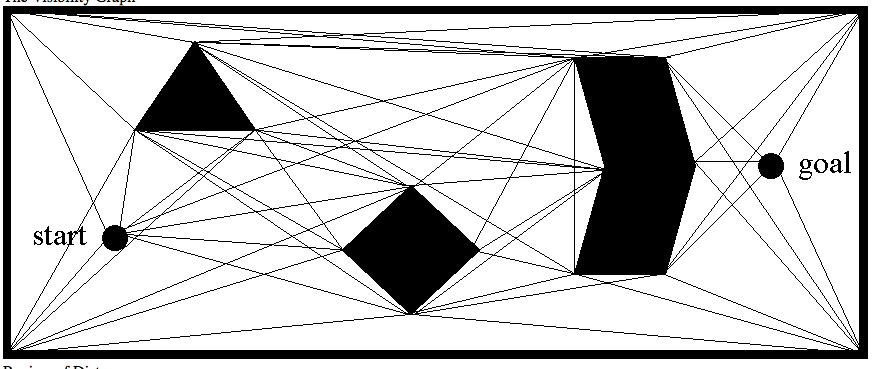


Figure 6.3: Visibility Graph Generation

### 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 minimalwdistance to each site.

To construct a voronoi diagram, follow the below steps:

* First draw straightelines 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.

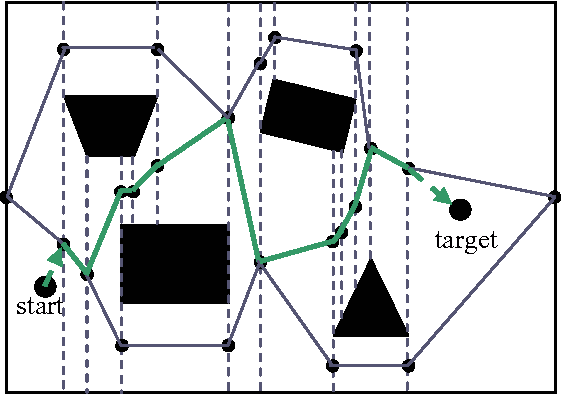


Figure 6.4: Voronoi Diagram

### 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.

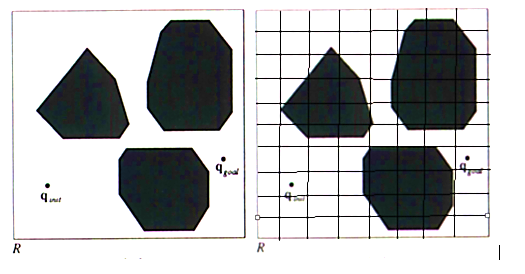


Figure 6.5: Exact Cell Decomposition

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 followingrscenarios 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 fulfilsaone 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.

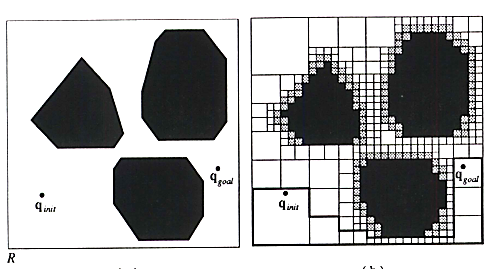


Figure 6.6: Approximate cell decomposition

## Graph Search for the Shortest Path

### Grassfire Algorithm

This algorithm works like fire spreads. Itwstarts 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 unmarkedenodes adjacent to the markedqnodes 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:

(6.1)

Where is the number of nodes in the graph.

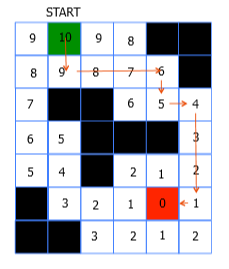


Figure 6.7: Grassfire algorithm [CMP]

### Dijkstra’s Algorithm

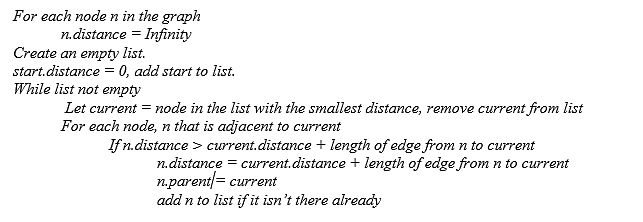
Dijkstra’s algorithm is quite an efficient algorithm to find the shortest path in the graph. It is different from minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph. This algorithm uses a greedy approach, in the sense that we find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

Here, we need to storeethe 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 mapweach vertex to the vertex that lasteupdated 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 , but the computational complexity with priority queue data structure is reduced to :

(6.2)

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

***Pseudo Code for Dijkstra’s Algorithm:***



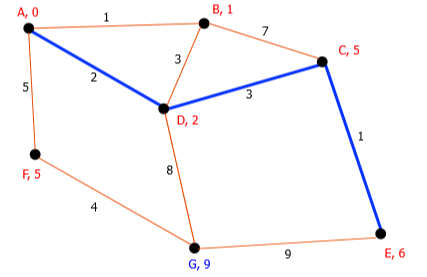


Figure 6.8: Dijkstra’s Algorithm [CMP]

### 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.

**F = G + H** (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.

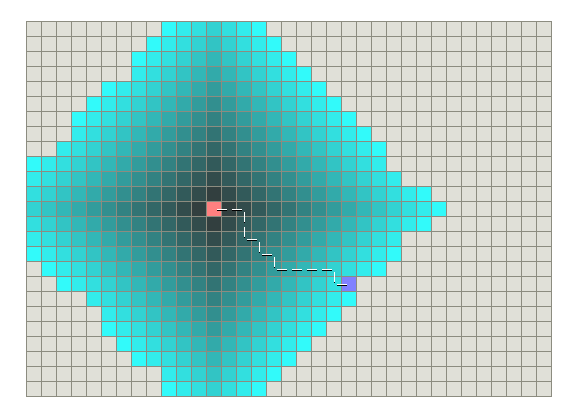
 

Figure 6.9: Dijkstra’s Algorithm on a grid Figure 6.10: A\* Algorithm on the same grid

## Artificial Potential Field

The basic thinking behind is to construct a smoothefunction over the extent of the configuration space which has higheravalues 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 attractiverpotential function, (x), can be constructedrby considering the distance between the current position of the robot, x = , and the desired goalelocation, = as follows:

(6.4)

Here is simply a constant scaling parameter.

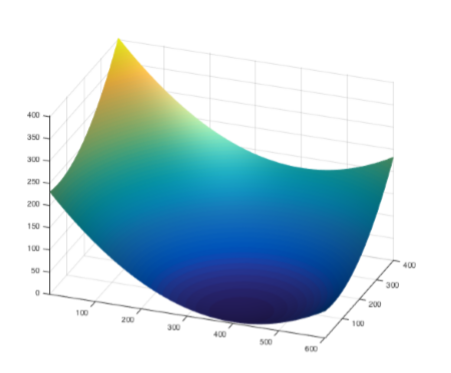


Figure 6.11: Attractive Potential field [6]

A repulsivedpotential functioniin the plane, , 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.

(6.5)

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

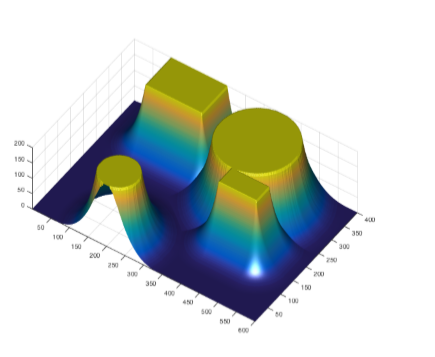


Figure 6.12: Repulsive potential field [6]

Combined potential function with gradient based control strategy:

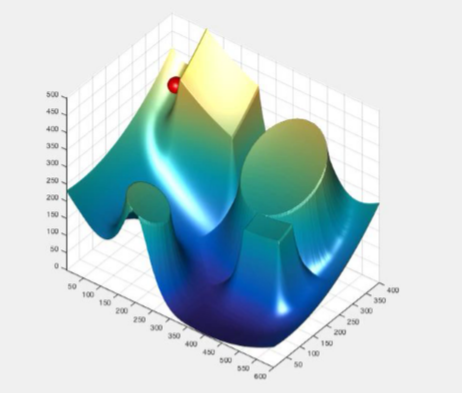


Figure 6.13: Combined Potential field [6]

## 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.

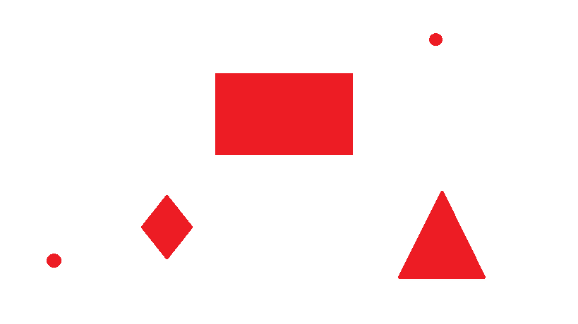
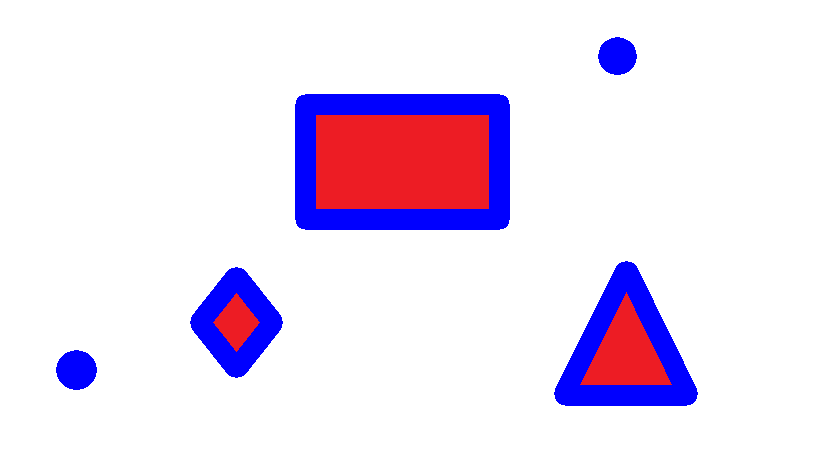
 

Figure 6.14: Obstacle Course Figure 6.15: Configuration space obstacles

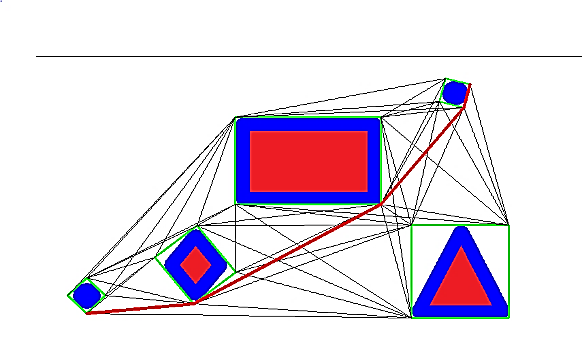
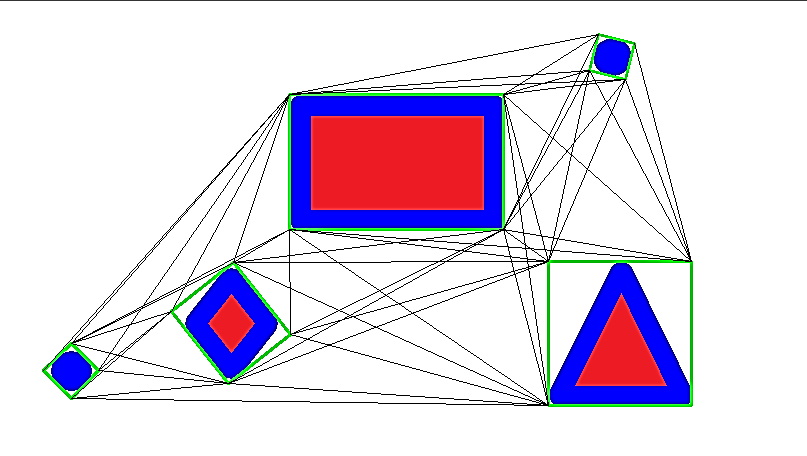
 

Figure 6.17: Obstacle Course with shortest path Figure 6.16: Graph generation

**Results on a more complex course:**

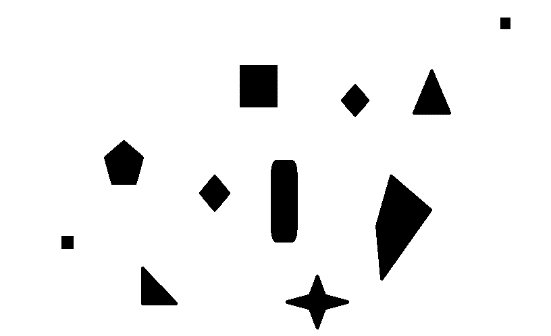
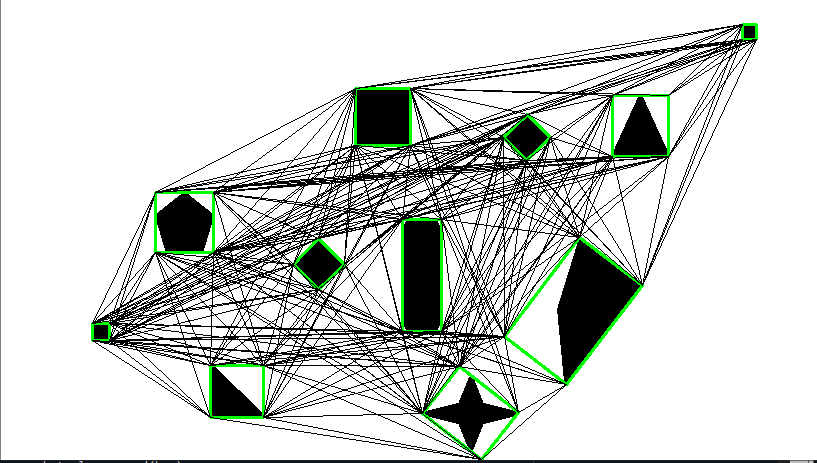
 

Figure 6.18: Obstacle Course Figure 6.19: Graph generation

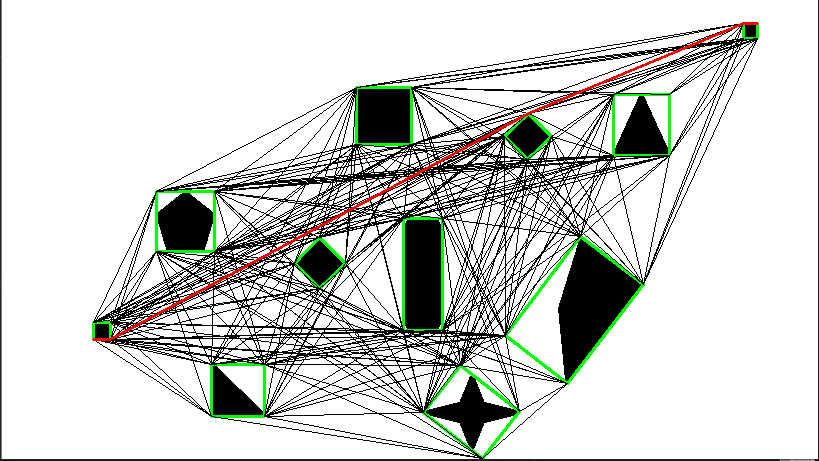


Figure 6.20: Obstacle Course with shortest path

# Motion Planning and Control Simulation

## 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 diﬀerentialsdrive robot has two controllable wheels mounted on its chassis as shown in ﬁgure 7.1. To control and maneuver any diﬀerential drivedrobot in a plane, the robot should be given a linear velocity V and a heading θ. Also, the path of thearobot can be planned by controlling the velocity and orientation of the robot.

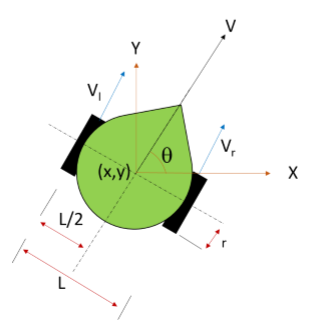


Figure 7.1: Robot parameters [8]

### 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 and that in y-direction is and the angular velocity are given by:

(7.1)

Now,

(7.2)

(7.3)

= (7.4)

Substituting the linear velocity V into and we get:

(7.5)

(7.6)

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

**= V +** (7.7)

**= V -** (7.8)

The outputs, and , can now be used to generate the output , , . 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.

## Control System

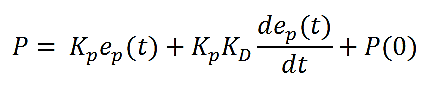
### Proportional Integral Derivative (PID) Control

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 Kp, , Kd 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.

Kp: 0.38

: 0.54

Kd: 0.76

 (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 measuredrprocess variable has been from the set point during the span of the time. Integration is a process of continualdsumming. Integration ofwerror 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.

## 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 **SimplejDirectMedia 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.

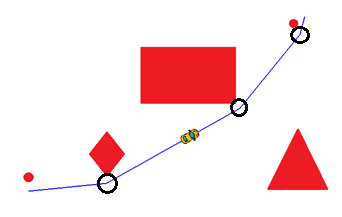


Figure 7.2: Simple Course Simulation without Interpolation

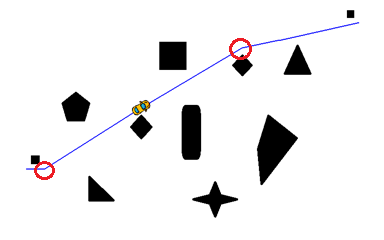


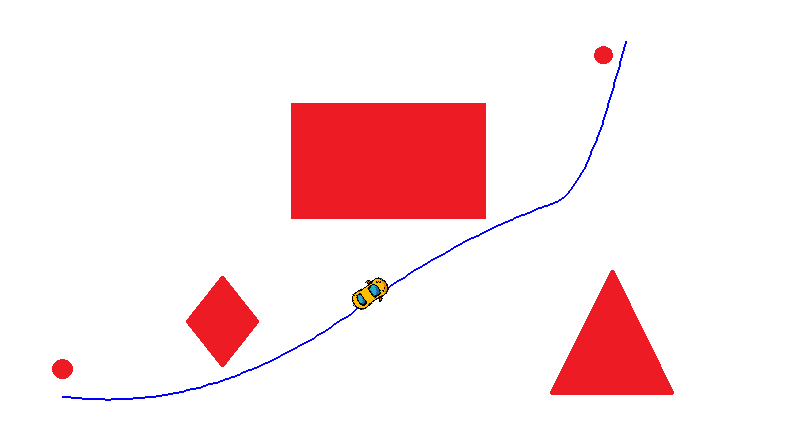
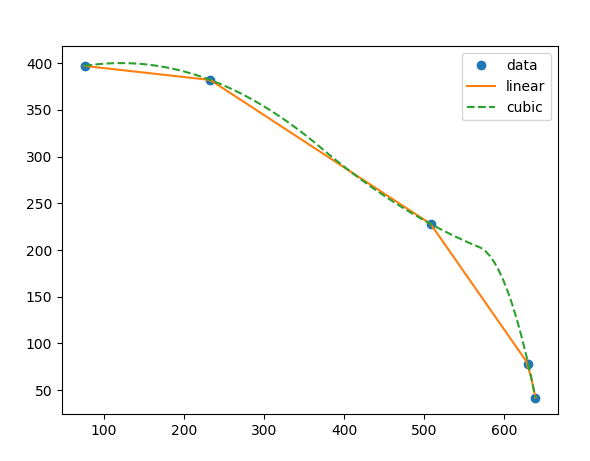
Figure 7.3: Complex Course Simulation without Interpolation

### 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

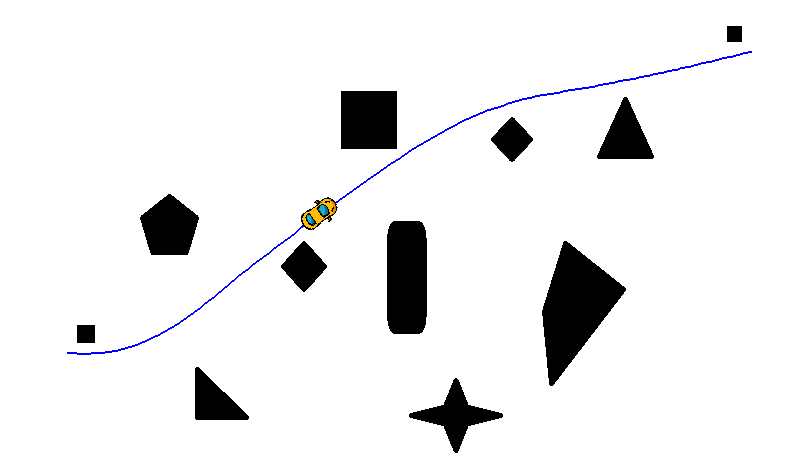
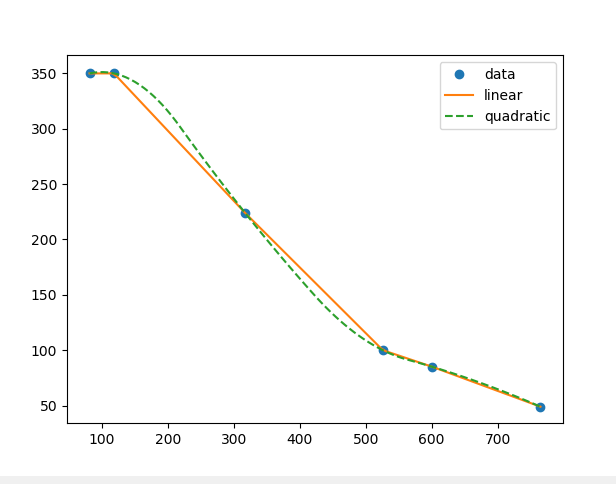


Figure 7.5: Complex Course Simulation with Interpolation

# 

# Conclusion and Future Scope

## 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.

## 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
   1. *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()

