ADA University

School of Information Technology and Engineering

Senior Design Project

**FINAL REPORT**

Project title: ***“Mobile Robot controlled with overhead camera in the room"***

**Authors**:

1. [IT] Gulnara Azizova

2. [CE] Mahammad Mahmudov

3. [CE] Umid Babazada

Project Advisor: **Pasha Pashazade**

Baku, 2024

Table of Contents

[**Abstract 7**](#_heading=h.v88okg6abijy)

[**1. Introduction 7**](#_heading=h.xyzwm2az4yhw)

[1.1 Purpose 7](#_heading=h.fauji8w9lduj)

[1.2 The Objectives 7](#_heading=h.q6m467qahale)

[1.3 The Significance 8](#_heading=h.y5d0nh5qig72)

[1.4 The Novelty 8](#_heading=h.by39jug1j3qg)

[1.5 Problem statement 8](#_heading=h.xtr7zf63rn61)

[**1. Literature Review 9**](#_heading=h.c63drkvdt0b6)

[**2. Design Concept 10**](#_heading=h.8vq5md9nj74r)

[3. 1 Alternative Solutions/Approaches/Technologies 10](#_heading=h.xmpt39jugt3t)

[3.2 Engineering Standards 12](#_heading=h.jpmf17qaez2q)

[3.3 Research Methodology and Technique 13](#_heading=h.285bjrsp8p0p)

[3.4 Architecture, Model, Diagram description 13](#_heading=h.8z3mb3orear3)

[3.5 Economic analysis 15](#_heading=h.jpk496e52odu)

[3.6 Social and environmental impact 15](#_heading=h.86jbwamolin2)

[**4. Implementation 15**](#_heading=h.o0fwch20byk4)

[4.1 Hardware Design 15](#_heading=h.n1na4gl3tmeo)

[4.2 Software Design 17](#_heading=h.e0lsi5qog7me)

[4.3 Essential Components of The Project 18](#_heading=h.x7bx2quo39mz)

[4.4 Testing/Verification/Validation 22](#_heading=h.7dg6pej1mc34)

[**5. Conclusion 23**](#_heading=h.ouspfm7js81k)

[**References 24**](#_heading=h.xih9w800nez)

# 

# 

List of Figures

|  |  |  |
| --- | --- | --- |
| No | Figure Caption | Page |
| 1.1 | Control of Differential Drive Mobile Robot | 13 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

List of Tables

|  |  |  |
| --- | --- | --- |
| No | Figure Caption | Page |
| 1.1 | Gantt Chart | 22 |
|  |  |  |
|  |  |  |
|  |  |  |

List of Abbreviations

|  |  |
| --- | --- |
| Abbreviation | Explanation |
| **OpenCV** | Open Source Computer Vision library |
| **HSV** | Hue, Saturation, and Value color space in color detection principle |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Abstract

This paper outlines the control design for a mobile robot with a differential drive. In this paper, various approaches and technologies, applied to design a robot control system that directs the robot to follow a path, are explained briefly.

with a predefined velocity profile. Developed control system

demonstrates stability and robustness in term of noises, errors

in the initial positions of the robot, and other disruptions.

***Keywords-component:*** *computer vision, image segmentation, ground-moving robot, image processing, control algorithms.*

# 1. Introduction

There has been a lot of interest in using vision as a sensor

and feedback for self-governing robotic systems. This is a

result of the increasing power and affordability of computers

and sensors. Robotic systems are able to perform harder tasks

through this integration. Currently, all of these are controlled

extensively. Systems for parallel parking and lane departure

warnings fall under this category. Mobile robots naturally

replacing from one location to another location. They

accomplish this without the help of outside human beings.

Compared to most of the industrial robots, which can move only

in a specific workplace,mobile robots have an ability to move

freely around within the predefined workplace to obtain the

targeted purposes.

Due to their benefit, mobile robots can be appropriate to be

used for larger applications in both structured and unstructured

environments. We applied to wheeled mobile robots, or WMRs

that are highly in demand. Their mechanical complexity and

energy consumption are relatively low, making them suitable

for common applications. A robot is a device that combines

action with vision and can function independently of humans.

Modern mobile robots are capable of autonomous thinking,

object recognition, and speech cognition[16]. When smart

control is combined with simple instructions and minimal

computational effort, intelligent design can achieve optimal

performance by modifying behavior. The mobile robot that we

are testing is based on the Differential Drive of WMRs among

the various drive types. One or two passive castor wheels are

included for stability and balance, along with two fixed-

powered wheels for independent driving on the robot platform.

A robot with a differential drive can move forward or backward

in accordance with the speed of its wheels. It’s an extremely

basic mechanical system and permits the robot to rotate around

the middle of its wheels or follow a curved route. The robot

moves after we send signals to the motors. It mechanical

components that drive the robot’s movement, and signals are

sent to them to control the speeds of both. Saying “control”

means that we are required to follow.

## *1.1 Purpose*

The project utilizes fundamental concepts in open source

computer vision, or in short, OpenCV library to process images

captured by a webcam. Image processing involves analyzing

and modifying

images through computational algorithms. The robot detects and recognizes colored circles on the paper surface, with control algorithms guiding its movement along a predefined path while

ensuring stability. The central point refers to the coordinates of detected circles, and the path indicates the robot's track. Motors drive the robot, with signals controlling their speeds, while Bluetooth enables wireless communication between the robot's control system, referred to as "embot" and the motors. Image segmentation breaks down images into individual objects, enabling the robot to identify regions of interest, such as colored circles. OpenCV is utilized for tasks like contour identification and color detection. The control algorithm generates the desired path and corrects control inputs for precise trajectory following, with Bluetooth facilitating wireless communication between the control system and the robot for remote management. Industry-standard coding practices ensure code readability, maintainability, and extensibility, with safety features like emergency stops and collision avoidance implemented to prevent collisions and reduce risks. Research papers, textbooks, and online resources inform the design process, with experiments and prototypes used to test concepts and improve the design iteratively. An economic analysis evaluates the cost-effectiveness of the mobile robot system, considering hardware costs, development time, maintenance costs, and potential benefits such as increased productivity. Vision is increasingly used as a sensor and feedback mechanism in autonomous robotic systems due to advancements in computing power and sensor affordability, with mobile robots having applications in both structured and unstructured environments. Wheeled mobile robots, like the one tested in this project, are popular for their mechanical simplicity and low energy consumption, with differential drive systems allowing robots to move autonomously and perform tasks efficiently.

## *1.2 The Objectives*

The goal of the project is to identify objects through a camera mounted on a robot. This is the creation of a robotic system that can use computational image processing methods to move along a set path. The use of the OpenCV Computer Vision Library for real-time object detection serves as the basis for the development of a machine vision-based robot control system. The following goals were also pursued: Real-time object recognition based on camera color identification using algorithm development using OpenCV and Python.   
Improved user interface (GUI) to improve user experience and ease of interaction with the robot. Implementation of an intuitive and user-friendly graphical interface, as well as an appropriate control system. Integration of mouse event functionality to allow users to draw and manipulate elements of the visual interface generated by the camera.  
Accurate Curve Drawing: Improves the precision of curve drawing on captured images. An image is proposed that ensures

the accuracy and smoothness of the traced trajectory of the robot.   
Interaction with multiple objects. This interaction allows the

system to recognize and draw several different objects

simultaneously, providing different functions or paths for each

recognized object.  
Coordination with robot movement. Developing a system in which drawn curves are directly translated into motion commands for a ground robot, allowing it to accurately follow intended paths.   
Error Handling and Reliability: Implementing reliability, error detection and recovery. Commissioning of mechanisms that ensure system reliability, minimizing errors in object recognition and coordination of movements.

## *1.3 The Significance*

The significance of the project lies in the fact that system performance is optimized by reducing time delay. There is an improvement in the response between the camera image and the interaction between the user and the robot. The importance of the project is justified by the creation of a comprehensive system that seamlessly integrates real-time object recognition, user interface design and robotic control. Using the OpenCV library and Python, as well as our own approach, involve the use of color identification, methods and k-means clustering for accurate object recognition. The graphical user interface (GUI) is developed using Tkinter, which will allow integration of mouse event functionality for user interaction. Drawing accuracy will be enhanced by algorithms that smooth and interpolate user-drawn curves using spline interpolation.

The significance of the project also lies in the fact that to process several objects simultaneously, our system implements complex tracking algorithms. A user-centered GUI design will function based on feedback to provide an intuitive and engaging experience. Coordination between the graphical user interface and robot motion will involve developing a communication protocol that converts graphical input into precise robot control commands.

Reliable error detection and recovery mechanisms based on fault tolerance principles will be implemented. Performance optimization will focus on reducing latency through parallel improvements in processing and algorithms. Python will be the main language, with OpenCV used for computer vision, Tkinter for GUI development, and a microcontroller platform such as Raspberry Pi for robot control. Our methodology emphasizes an interactive development process that includes continuous testing and user feedback to refine and improve the efficiency and usability of the system.

The implementation of the project is necessary because the development will contribute to the transportation systems industry and will help solve the problems of movement and transportation in difficult conditions.

## *1.4 The Novelty*

The novelty of the project lies in the fact that it creates a real prototype that demonstrates object recognition in real time. Interface interaction and precision robotic control will be introduced. The computer code, carefully documented in Python, will provide valuable information for future development and troubleshooting. In addition, detailed manufacturing drawings will visually represent the physical layout of the system, facilitating reproducibility. These results collectively demonstrate the successful integration of computer vision, user interface design, and robotics, offering a comprehensive solution for interactive object recognition and control.

## *1.5 Problem statement*

Recently, there has been a lot of interest in using computer vision as sensor and feedback for autonomous robotic systems. This is due to the increasing power and availability of computers and sensors. Thanks to this integration, robotic systems can perform more complex tasks. Currently, all of them are widely used. This includes parking systems and parallel lane departure warnings. Moving from one location to another is naturally called mobile robots. They perform all actions without human help. Unlike most industrial robots, which can only move within a specific job location, mobile robots can move freely around a predetermined job location to achieve goals. Due to their advantages, mobile robots can be applied to larger applications in both structured and unstructured environments.

In our project, we pose the problem of using wheeled mobile robots, or WMRs, which are in high demand. Their mechanical complexity and power consumption are relatively low, making them suitable for general applications. A robot is a device that combines action with vision and can function independently of humans. Modern mobile robots are capable of autonomous thinking, object recognition and speech recognition. When intelligent control is added to intelligent design with simple instructions and minimal computational effort, optimal performance can be achieved by modifying behavior. The mobile robot that became the basis of our project and which we are testing is based on a KMR differential drive among various drive types. One or two passive casters are included for stability and balance, along with two fixed drive wheels for independent movement on the robot platform. A differential drive robot can move forward or backward according to the speed of its wheels. This is an extremely simple mechanical system and allows the robot to rotate around the middle of its wheels or follow a curved path.

To determine the robot's direction, the program uses a webcam mounted on the robot to capture frames or images of captured objects. When colored circles are placed on the card surface, the robot device uses these frames or images to identify them. He then determines where these circles intersect. The robot's left and right motors then determine how fast they need to move forward to follow the desired path, thanks to a control algorithm. Both linear and angular velocities are transmitted via Bluetooth communication with the robot. By combining closed-loop feedback and computational image processing, our design aims to provide reliable and predictable robot motion. When we send signals to the motors, the robot starts moving. These are the

mechanical components that drive the robot and signals are sent to them to control the speeds of both. To say "governance" means that we must follow. The project uses core computer vision concepts using the OpenCV Python library to process images captured by a webcam. Image processing involves analyzing and modifying images using computational algorithms. The robot detects and recognizes colored circles on the card surface, and control algorithms guide its movement along a predetermined path, ensuring stability. The center point refers to the coordinates of the detected circles, and the path indicates the robot's track. The motors propel the robot and signals control its speed, and Bluetooth enables wireless communication between the robot's control system, called an "embot", and the motors. Image segmentation breaks images down into individual objects, allowing the robot to identify areas of interest, such as colored circles. OpenCV, a popular open source library for computer vision, is used for tasks such as edge identification and color detection. The control algorithm creates the desired path and adjusts control inputs to accurately follow the path, and Bluetooth provides wireless communication between the control system and the robot for remote control. Industry standards ensure code is readable, maintainable, and extensible, and safety features such as emergency stops and collision avoidance are implemented to prevent collisions and reduce risks. The design process is informed by scientific articles, textbooks and online resources, experiments and prototypes are used to test concepts and improve the design iteratively. An economic analysis evaluates the cost-effectiveness of a mobile robot system, taking into account hardware costs, development time, maintenance costs, and potential benefits such as increased productivity. Vision is increasingly being used as a sensor and feedback mechanism in autonomous robotic systems due to advances in computing power and sensor availability, and mobile robots have applications in both structured and unstructured environments.  
The importance of the problem is that wheeled mobile robots, such as the one being tested in this project, are popular for their mechanical simplicity and low power consumption, and differential drive systems allow robots to move autonomously and perform tasks efficiently.

# Literature Review

Color detection is a fundamental task in computer vision with numerous applications, ranging from object tracking to image segmentation. Bansal (2023) provides a beginner's reference on color detection using Python, offering insights into techniques and methodologies for identifying specific colors in images. By leveraging libraries such as OpenCV, Python enables developers to implement robust color detection algorithms efficiently. Understanding color detection is crucial for various computer vision tasks, including robotics, surveillance, and augmented reality.

Curve fitting is another essential aspect of data analysis and computer vision, particularly in trajectory planning and object recognition. Brownlee (2021) presents a comprehensive guide to curve fitting with Python, covering various curve fitting techniques and their implementation using libraries such as

NumPy and SciPy. Curve fitting plays a vital role in modeling

and interpreting data obtained from sensors, cameras, and other sources, making it a crucial component of many computer vision applications.

Multiple color detection in real-time is a challenging problem often encountered in applications like robotics, image processing, and augmented reality. GeeksforGeeks (2023) provides insights into detecting multiple colors simultaneously using Python and OpenCV. This resource offers practical examples and code snippets to help developers implement robust

color detection algorithms capable of handling real-world scenarios.

Efficiently defining lower and upper color ranges is crucial for accurate color detection. The OpenCV community discusses strategies for defining color ranges effectively in the OpenCV Q&A Forum (n.d.). This resource provides insights into various factors affecting color range definition and offers practical tips for optimizing color detection algorithms [4].

Thresholding is a fundamental technique used in image processing and computer vision for segmenting images based on pixel intensity. OpenCV provides extensive documentation on image thresholding techniques (n.d.), offering developers a comprehensive understanding of thresholding methods and their applications in color detection, object segmentation, and image enhancement [5].

The application of object detection using OpenCV and Python is a prevalent topic in computer vision research and development. Researchers have explored various methodologies and techniques for object detection, including cascade classifiers (OpenCV, n.d.) and deep learning-based approaches. IEEE conference publications[6] have documented advancements in object detection algorithms and their practical applications in robotics, surveillance, and autonomous systems.

The OpenCV Cascade Classifier[7] is a fundamental component in object detection methodologies, widely utilized in computer vision applications. This classifier operates based on a series of trained cascade stages, enabling the detection of objects within

images or video streams. Understanding the intricacies of cascade classifiers is crucial for developing robust object detection systems in various domains.

OpenCV Documentation[8] serves as a comprehensive resource for understanding the functionalities and capabilities of the OpenCV library. With detailed documentation and examples, developers can effectively utilize OpenCV's extensive features for tasks ranging from image processing to machine learning. This resource remains invaluable for both beginners and experienced practitioners in the field of computer vision.

Drawing functions in OpenCV[9] play a crucial role in visualizing data, annotations, and results in image processing and computer vision applications. The OpenCV documentation provides detailed explanations and examples of drawing functions, enabling developers to create custom visualizations, annotations, and overlays in their projects.

Performance measurement and optimization techniques[10] are essential for enhancing the efficiency and speed of computer vision algorithms. OpenCV offers a range of optimization techniques and best practices for improving the performance of image processing and computer vision applications. By leveraging these techniques, developers can optimize their algorithms for real-time processing, resource-constrained environments, and high-throughput applications.

Mouse events play a crucial role in interactive applications and user interface design, including drawing and annotation tools. OpenCV provides functionalities for handling mouse events, enabling developers to create interactive applications where users can interact with images and graphical elements using mouse input[11]. This resource offers insights into how to draw curves using mouse events in OpenCV and Python, facilitating the development of interactive drawing and annotation tools.

The planning and control of mobile robots in image space from overhead cameras is an emerging area of research within robotics[12]. Leveraging overhead cameras for navigation represents a departure from conventional sensor-based methods, offering potential advantages in adaptability and flexibility across diverse environments.

Real-time object recognition using OpenCV and NumPy in Python[13] presents a significant advancement in computer vision capabilities. By harnessing the computational power of these libraries, researchers can achieve efficient processing of visual data, enabling timely and accurate object recognition essential for robotics applications.

Curve fitting in Python, as elucidated by Samal, offers a comprehensive guide to practitioners seeking to analyze and model data trends. By leveraging Python's rich ecosystem of libraries, including NumPy and SciPy, researchers can apply

sophisticated curve fitting algorithms to extract meaningful

insights from experimental data[14].

The project on color detection using Pandas and OpenCV in Python[15] provides hands-on experience in applying computer vision techniques to identify and analyze colors within digital Python-based solutions in solving real-world problems in computer vision images. This practical application demonstrates

the versatility of Python-based solutions in solving real-world

problems in computer vision.

Mobile robot control is an interdisciplinary field that encompasses aspects of robotics, control theory, and artificial intelligence. Tzafestas (2013) provides an introduction to mobile robot control, covering topics such as motion planning, trajectory generation, and feedback control algorithms. This resource serves as a comprehensive reference for understanding the principles and techniques of mobile robot control, offering insights into both theoretical foundations and practical implementations.

Color-based object detection is a fundamental task in computer vision with applications in object tracking, recognition, and localization. Yacine (n.d.) discusses techniques for color-based object detection using OpenCV and Python, providing practical examples and code snippets for implementing color-based object detection algorithms. By leveraging color information,

developers can robustly detect and track objects in images and videos, enabling various computer vision applications.

# 

# 

# 

# 

# Design Concept

## *3. 1 Alternative Solutions/Approaches/Technologies*

The first approach we applied in our project was linked to

color detection principle. Basically, three primary colors from

which all other colors are made are red (R), green (G), and

blue (B). A number between 0 and 255 symbolizes each color

on a computer. Once we created a function for detecting

specific two color ranges and their coordinates, the first step

we completed in this process was to convert the frame captured

by the camera from the standard BGR color space into hue-

saturation-value color space, or HSV in short [1]. By moving

to the HSV color space, we can more accurately detect specific

colors, such as red and blue colors, and extract beneficial

information so that we can convey extra analysis. This

convention is implemented by OpenCV’s cv2.cvtColor( )

function [4]. By establishing boundaries on the HSV frame,

this function generated the first red color mask. We established

the lower and upper bounds to represent red hues. Then, we

created the second mask for red color by modifying a different

red color range to match the hue range of the HSV. Eventually,

red region detection was enhanced by cv2.bitwise\_or( ), which

combined both red masks into a single red mask [3].

Furthermore, in a frame, our goal was to identify and label

color regions. Initially, here we computed moments, including

the region denoted by m00, using cv2.moments( ) on the

binary red mask subsequently. Next, we wrote a bunch of code

we determined the centroid of each color region in the mask

that has been identified. We gathered statistical data regarding

the arrangement of color regions through examining them. We

could ascertain whether the mask contains actual colors by

verifying that M[“m00”] is not equal to 0. If so, the algorithm

makes use of the frame’s color distribution to calculate the

middle points, or cX and cY. On both horizontal and vertical

lines, these points indicate the locations of the majority of the

colors. After that, they are saved in the coordinates list for

future reference. Otherwise, we discovered the absence of an

important central point if the program could not detect valid

color regions.

Next, the main principle we covered in the project tracks

closed-loop control algorithm. To draw curves on a frame

with the mouse, we created a function, named

draw\_curve\_with\_mouse\_events function. This allows for

the smooth creation of curves by responding to mouse events.

The code snippet in our program identifies whether this is the

first point, st == 0 and includes the coordinates to the list,

indicating the beginning of the curve, when we press the left

button of the mouse. In the same process, we should stop

drawing, meaning stop == 0. Meanwhile, we make sure that

there is enough distance between the new and the previous

data points.

After that, with the code modifications we made, the control

behavior should be adjusted according to the difference

between the current and desired positions. The significance

of Pi\_error diminishes with increasing robot distance from

the target. As the robot gets closer to its destination, the

primary objective is to give preference to reaching the

desired position (x\_desired, y\_desired). This was accomplished

by dynamically adjusting K\_phi in accordance with the

positional distinction. When the robot is farther away, K\_phi

drops, minimizing the effect of Phi\_erro on control. By

concentrating more on efficiently reaching the target, the

robot becomes less susceptible to small directional errors.

In contrast, K\_phi rises as the robot approaches the intended

position and location error falls. With this modification,

Phi\_error’s authority is increased and accurate alignment

with the desired direction, phi\_desired is promised. As K\_phi

arrives closer to the target , the robot can respond to the

direction changes more successfully and navigate the path

with greater accuracy. A line, Ferr = phi\_desired - phi, is also

contained in the program to compute Phi\_error and make sure

that it remains within the range of -pi to pi for precise control.

Following this, next activity to implement was calculating

velocities of both left and right wheels for differential drive

mobile robot. So, after creating a new function, named the

calc\_wheels\_velocities, we defined the velocities both left

and right wheels which were required. Wheelbase distance,a,

angular velocity, omega, and linear velocity, v, are the

parameters that it takes. Simply, this function calculates each

wheel’s velocity using provided inputs. Moreover, there are

two primary steps in the velocity calculation for each wheel:

To find the linear velocity component, v/p2d, first we divided

this velocity by p2d. This made certain that our linear velocity

corresponds to the physical dimensions of the robot and the

units used in the control algorithm. The second phase involves

multiplying the angular velocity, omega, by a constant factor

and adding it the linear velocity component (fo the left wheel)

or subtracting it from the right wheel. In order to safely

perform angular motions while preserving trajectory stability

and accuracy, this makes up for the robot’s differential motion

characteristics.

In essence, we added another functionality as

an alternative method to calculate wheel velocities, which

involving (v\_left = v - (a \* omega) and v\_right = v + (a \*

omega)). With this method, we used the wheelbase and

angular velocity of the robot to directly alter the linear

velocity parameter. Nevertheless, simplicity and clarity

are given priority in the widely accepted implementation

of this function.

Then, we had to generate another function to compute angle,

midpoint, and distance between two coordinates so that we

named it calc\_distance\_angle\_and\_midpoint function. The

function takes two coordinates tuples, called coord1 and

coord2, serving as points on a Cartesian plane, are fed. After

that, we determines the angle, distance, and midpoints between

them. Using the Euclidean distance formula, this function

finds the distance that exists between two data points. This

entails calculating the square root of the total squared

differences between their coordinates in x and y. The linear

distance that results from the points’ distance is measured.

This function also calculates the angle formed by the positive

x-axis and the line that joins the two points. In the light of

the variations in the x and y coordinates, this applies the

arctangent function. For flexibility, we gave the angle in

degrees as well as radians. In addition, using the x and y

values as averages, we determined the midpoint between

the provided coordinates. The line that integrates these two

input coordinates has its midpoint at this location.

For the project, we applied spline interpolation methods to

derive smooth paths from user-specified waypoints. The first

thing we did was take the given data points and use splprep

function to compute the B-spline representation of the curve.

In this case, a list ([xd, yd]) we created containing the

coordinates of the waypoints on the x and y axes are passed

as an argument. In addition, since we set the s parameter to

0, no smoothing is needed, meaning the spline must precisely

intersect through each input point. Two primary elements make

up the output of the splprep function. One of them is called u,

which indicates the parameter values corresponding to

input points. Another parameter, named tck is a tuple

containing the knot points, coefficients, and degree of the

B-spline curve. Next, we used the splev function, which

assesses how well the spline is represented at particular curve

points. np.linspace(0, 1, 10 \* len(lines\_arr)) produces values

ten times the number of waypoints, evenly spaced between 0

and 1. These numbers represent sample points on the tck-

representable B-spline curve. The system is able to create a

constant trajectory by seamlessly.The x and y coordinates

of the interpolated points are stored in the resulting xi and yi

arrays navigating between waypoints with the help of these

points.

We guided a mobile robot to create a continuous and smooth

trajectory. First, we created a time vector, named t that defines

the trajectory’s duration and spans from 0 to 2 seconds at

intervals of 0.01 seconds. A collection of data points that

defines this path, usually obtained through interactively or

through other methods, such as image processing. We created

two arrays, named xd and yd, representing the x and y

coordinates and saved them respectively. Next, we used the

splprep function from SciPy library to perform spline

interpolation on the given data points. Mainly, this technique

avoid overfitting and ensures a smooth curve that either passes

through or closely matches the provided data points. The (tck,

u) tuple represents the complete spline function. In this

notation, u indicates the spline parameterization whereas tck

features the knots, coefficients, and degree of the spline. Then,

we evaluated spline at uniformly spaced intervals along its

parameterization using the splev function. In this process, we

generated new coordinate arrays xi and yi which comprise the

desired path data points obtained from the interpolated spline.

Consequently, the path indicated with xi and yi provides

constant and seamless path for the robot to travel.

Picking the desired path and angular path for directing a

robot's movement depends critically on an interpolated spline.

It starts by using spline interpolation to compute its trajectory.

Allocating coordinates (x\_d and y\_d) established the intended

path for the robot. Next, we identified Phi\_desired to find the

direction of the robot along the given path. After this, we tried

to calculate the angle between successive points in the desired

path using np.arctan2 function. Basically, we regulated the

basic parameters required to specify the motion profile so that

the robot should follow, allowing the accurate control and

navigation along the desired path.

Furthermore, we covered an important facet of arranging and

handling the robot’s trajectory. To control how precisely the

trajectory is planned, we created included another parameter,

named Ts to demonstrate the time interval between consecutive

updates of the desired path. Then, based on the rates of change

of the robot’s desired x and y positions, we determined the

desired linear velocity of the robot. Our program calculated the

velocity elements along both axes by examining these changes

over time. Calculating the total speed of the robot in its desired

path refers to splitting these changes by the time interval to

get the velocity elements per unit time. In addition, we

calculated the desired angular velocity by measuring the rate

at which the desired direction angle changes. This type of

velocity just handles the capability of the robot to rotate or

direction while staying in the desired trajectory. So, this part

is considered necessary for defining the robot’s movement

and providing accurate control over its movements because

it computes both linear and angular velocities from the

derivatives of the path.

The next section outlines interaction with external devices

and video input capture. Using OpenCV, initially, we created

a video capture object. For future use, we configured this

object to utilize index 0 to access the default camera. After

initializing the camera, we established Bluetooth

communication within the device. To do this, we must specify

the port, which in this case is designated as “COM4”. Our

serial module has a 115200 baud rate and this is used for

Bluetooth connection. Furthermore, the method, named

flushInput() we created makes sure that the communication

is straightforward by clearing the input buffer of the Bluetooth

object. As soon as the communication interface is established,

we sent to the Bluetooth device its first message. Two digits,

separated by forward slashes, make up this message, which is

encoded in bytes. This is likely an initialization signal or

control command for the attached hardware. These two

numbers in this message, indicated by 0s are placeholders,

that may represent particular commands that the person

receiving the device has been recognized.

Our code snippet related to serial communication displays

how to apply image processing to control a robot’s path

via Bluetooth connectivity. The program takes images from

a camera in an ongoing process, identifies certain colors in

the frames, and computes relevant information like the angle,

distance, and midpoint between detected data points. Then,

we triggered trajectory control algorithm by these parameters,

and uses them to calculate the necessary velocity and angular

velocity for the robot’s movement. Next, we transformed these

speeds and modified them to suit the kinematics of the robot.

Ultimately, we release the camera and stop the robot. For the

sake of resource management and stability, it closes all

OpenCV windows.

## *3.2 Engineering Standards*

The mobile robot system has been diligently developed and

refined to conform to the principles of engineering standards

for robotics, software, and image processing. The system’s

extensibility, maintainability, and comprehensibility are

guaranteed by strict adherence, which also make it durable

and flexible enough to meet changing requirements. The

system has cut-edge features such as emergency stops and

collision avoidance mechanism to prioritize safety and reduce

potential risks. In order to avoid collisions and guarantee the

safety of the robot and its surroundings, these preventive

measures are considered essential. Furthermore, a serial

communication adheres to engineering standards.

Communication protocols between system elements are vital

for efficient and trustworthy data exchange. Our program

adheres to Bluetooth standards by implementing these

protocols for both serial and wireless communication. Through

a channel, data is sent bit by bit during serial communication.

To communicate serially, we employed COM4 port and

a baud rate of 115200. Data speed and consistency across

devices are influenced by this rate. In accordance with serial

communication standards, our program clears input buffers and

encodes data into bytes before transmitting it. Short-range

wireless communication is feasible with Bluetooth. When a

Bluetooth is paired with our device, wireless data transmission

is initiated. The frequency bands, modulation strategies,

encoding formats, and connection protocols are all covered in

Bluetooth. It makes sure that Bluetooth works with it by

adhering to observe these requirements.

In conclusion, to guarantee compatibility and interoperability

between systems and devices, the program complies with

accepted communication protocols. For serial communication,

it sets the baud rate to 115200. By enabling smooth data

transfer between various elements, these procedures enhance

the communication system’s dependability and efficacy.

## *3.3 Research Methodology and Technique*

Examining existing works, conducting experiments, and

building prototypes are all steps in the design process. To stay

up to date with the newest developments in image processing,

control algorithm, and mobile robotics, we consult research

papers, textbooks, and online resources. Next, we create

prototypes and test our concepts to see well how the system

performs in various scenarios. That way, we can gradually

improve the design.

Two primary tasks are involved in this part: taking pictures

with overhead camera and documenting the robot’s motion

data. Certain system functions, such as calculating distance,

angle, and midpoints related and detecting colors and

get their coordinates related functions are crucial to this

procedure. These functions carefully examine each frame

captured by the camera. The first indicating function provides

beneficial information regarding the robot’s direction and

positioning. In addition, the second one aids in defining the

location of the robot and other relevant distance data. These

mechanisms provide the groundwork for further examination

and assessment of the robot's navigation capabilities.

Through trajectory comparison, error calculation, and

visualization, this phase assess the precision and effectiveness

of the navigation system. These are made with the help of the

control algorithm function, which computes control commands

based on the actual and desired states. To direct the robot along

a predefined path and modify its movements in response to

real-time feedback, these instructions are delivered in the main

loop. Matplotlib and other visualization tools plot intended and

actual trajectories, offering insights into system behavior for

further analysis and improvement.

## *3.4 Architecture, Model, Diagram description*

System architecture entails hardware devices and software

modules. The hardware of the system consists of a camera and

a robot. In the meantime, the camera records video input for

the software, and the robot physically carries out commands

from it. With the camera providing visual data for analysis and

decision making, the robot acts according to calculated paths

and velocities. When these parts work together, the system can

effectively interact with its environment, adjust to changes, and

deliver a variety of tasks. The software modules include

functions related to color detection, control algorithm, and

curve details. For instance, color detection can only identify

red and blue colors in camera-captured frames. Both velocities

are estimated by the control algorithm using the detected colors

and the desired path. Basically, we used this control to

accurately follow a predefined trajectory by adapting linear

and angular velocities. Plus, with curve details, users can

interactively draw curved paths on images and save the drawn

curve coordinates to be used as the desired path input for the

control algorithm. By enabling efficient color identification,

velocity computation, and user-specified path planning, these

software elements reinforce system functionality and

versatility in a variety of solutions.

Interaction flow demonstrates that the camera records

environmental frames to serve as visual input for operations

that come after as a part of the interaction process.

Subsequently, the color detection module examines these

frames to identify specific colors - red and blue ones. The

control algorithm calculates the velocities required for the

robot’s movement using this color information and the desired

trajectory. Then, the robot can smoothly execute the motion

instructions due to the calculated velocities sent to its motors.

In meanwhile, the curve details module allows users to draw

interactively desired curve trajectories. The robot can exactly

adhere user-defined paths with the help of determined

coordinates of these drawn curves as input for the control

algorithm stored.

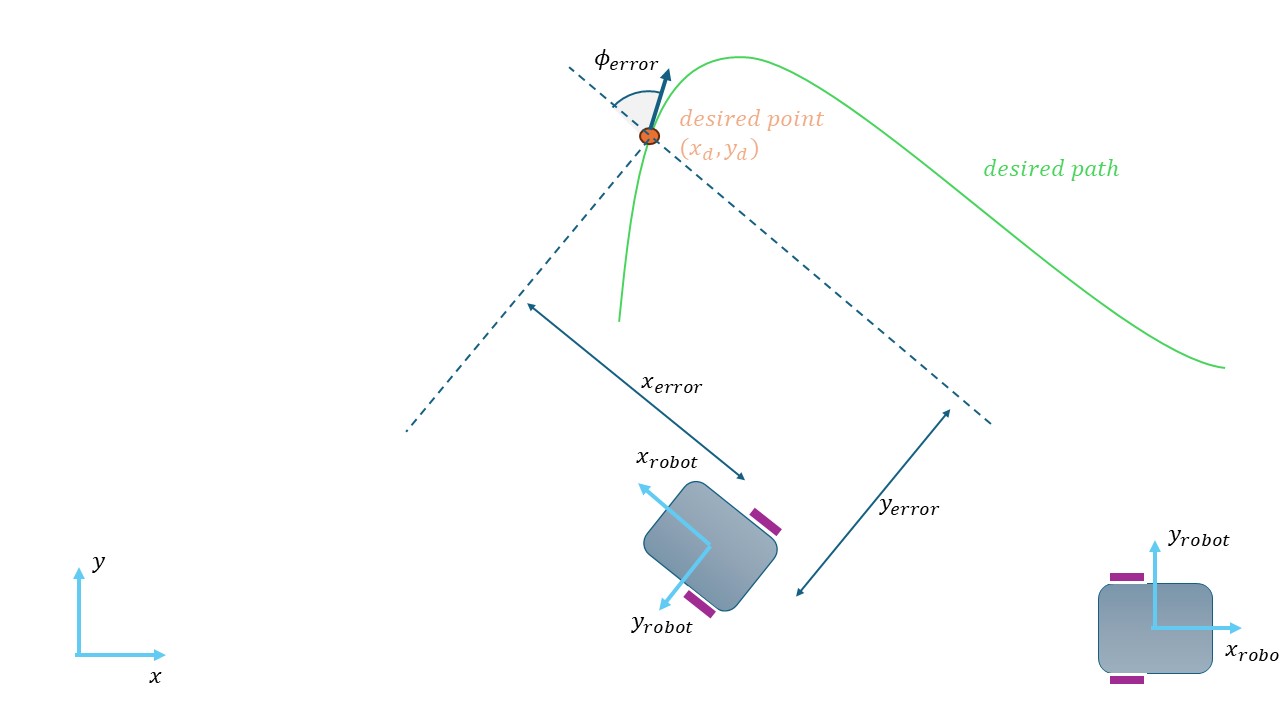


Figure 1.1 Control of Differential Drive Mobile Robot

Figure 1.1 demonstrates the process of how to control mobile

robot with differential drive. The robot’s speed increases or

decreases based on the value of Kx parameter, which implies

that the sensitivity also changes. But the angular velocity is

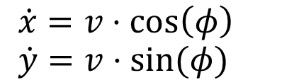
influenced by the Ky value. According to the Lyapunov

stabilizing method [16], the dynamic model, one of the stages

in control procedure, and the other, called kinematic model

manages the robot motion using the following formulas

(Equation 1):

 (Equation 1)

A kinetic model of the system is delivered by another

formula depicted in the following equation (Equation 2):

 (Equation 2)

The kinematic non-holonomic constraint, x ̇⋅sin⁡(ϕ)=y ̇⋅cos⁡(ϕ)

holds and indicates that vas a vector (x ̇,y ̇)always has the

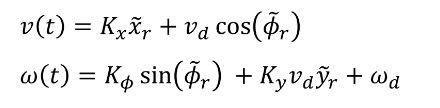
angle ϕ with the positive x axis.

Assuming that the DDMR system and the intended trajectory

meet all of the system’s equations and constraints,

the following control algorithm for y(x) and y(y) is stable

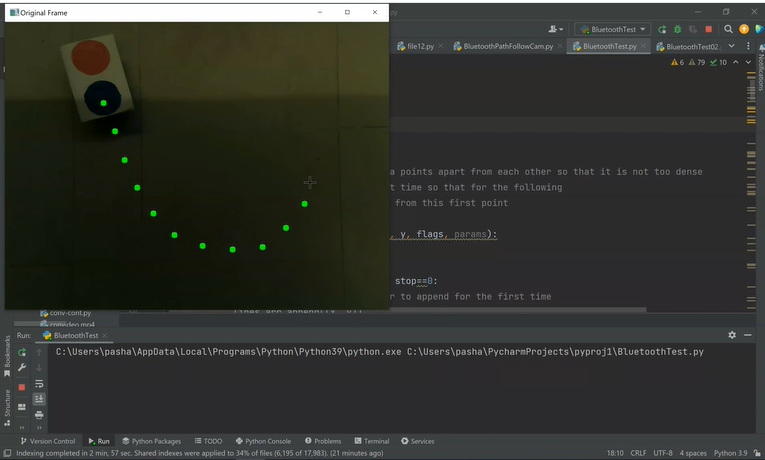
over time (the error approaches zero rapidly):

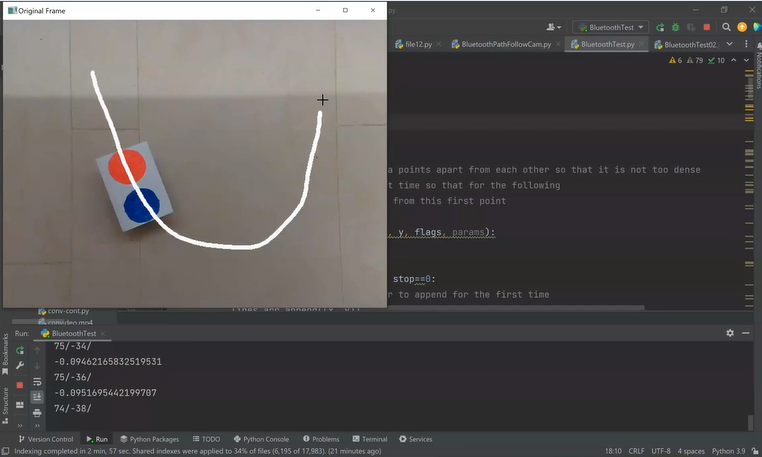


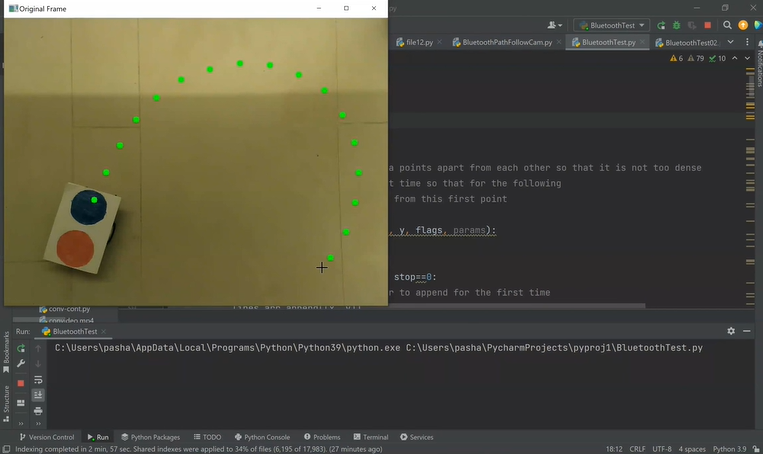


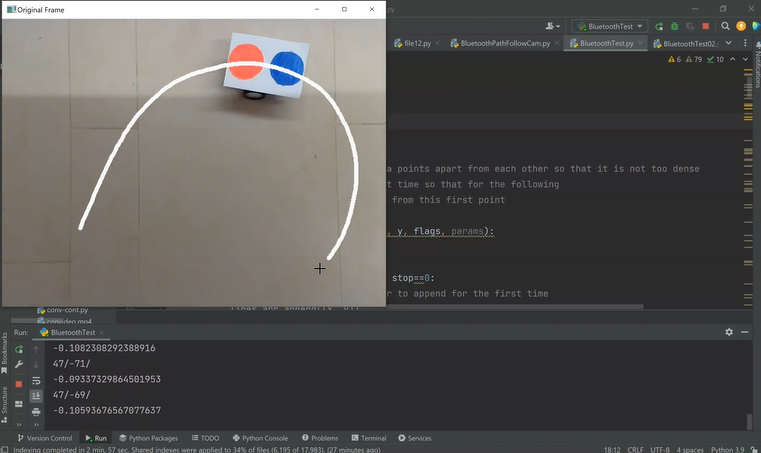
If we make it clear in writing:

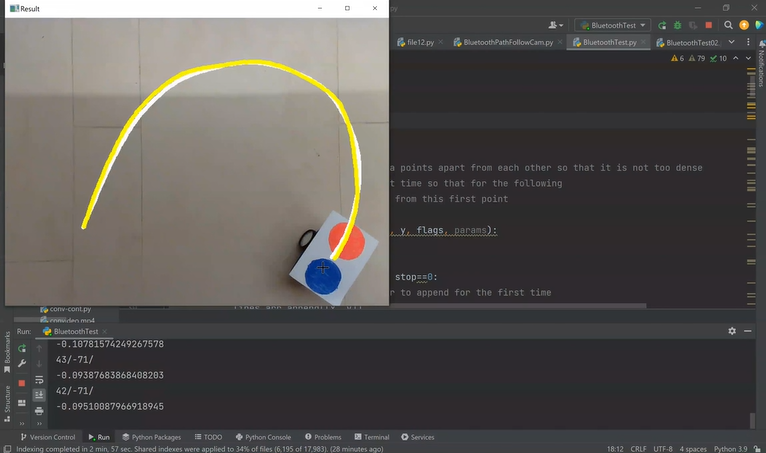


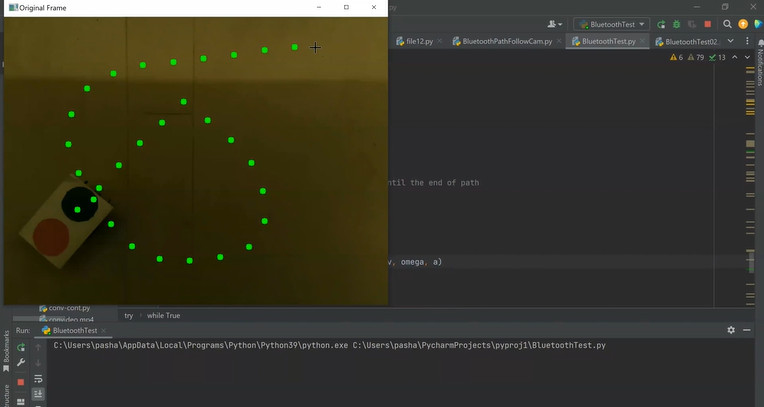


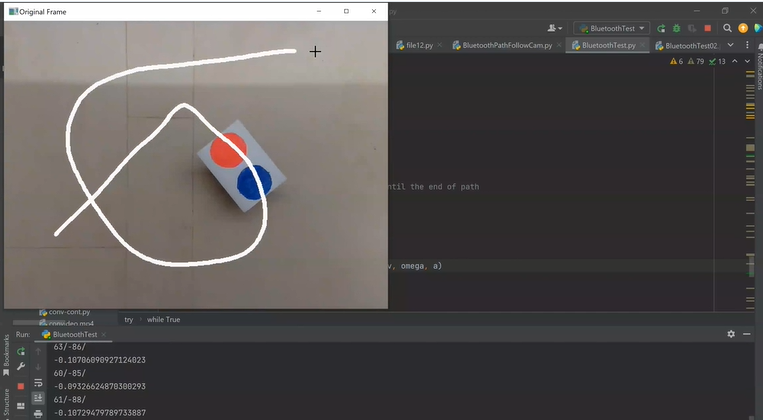


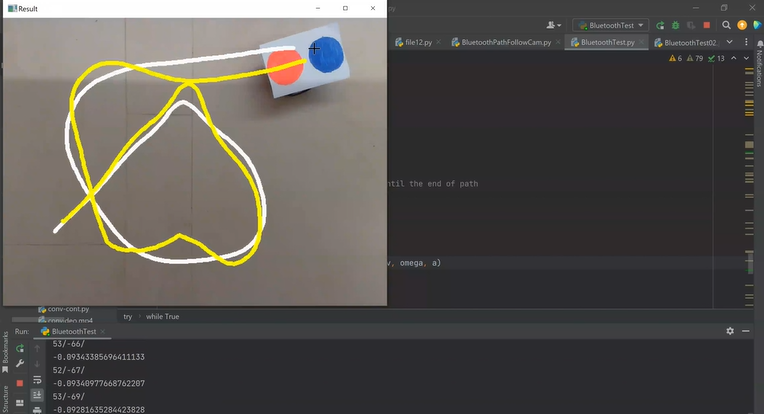




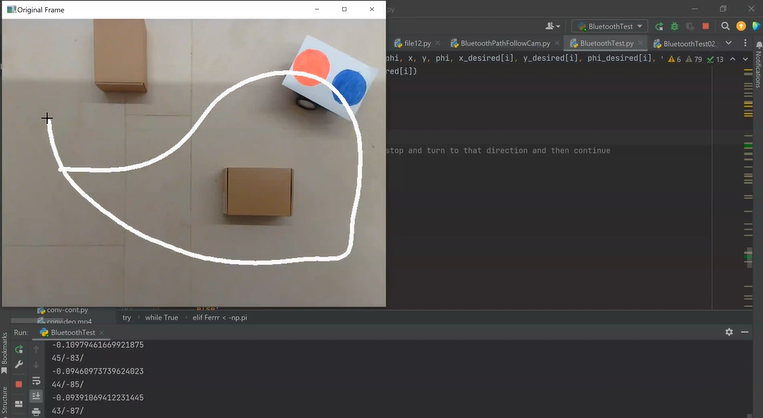
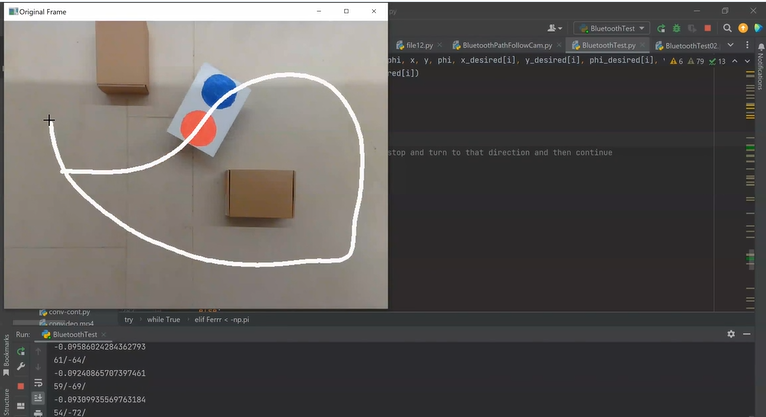


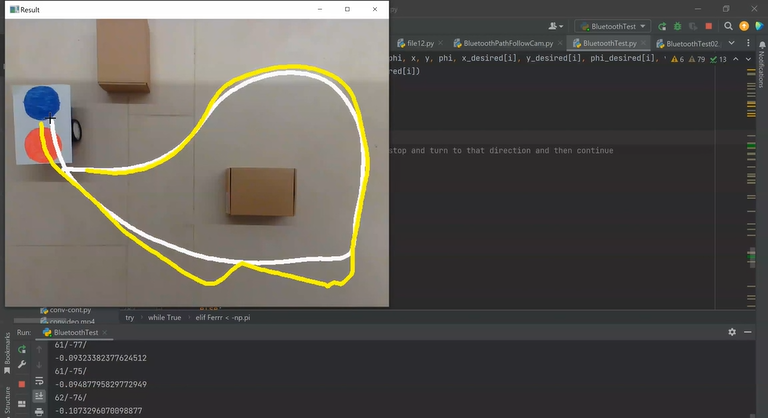
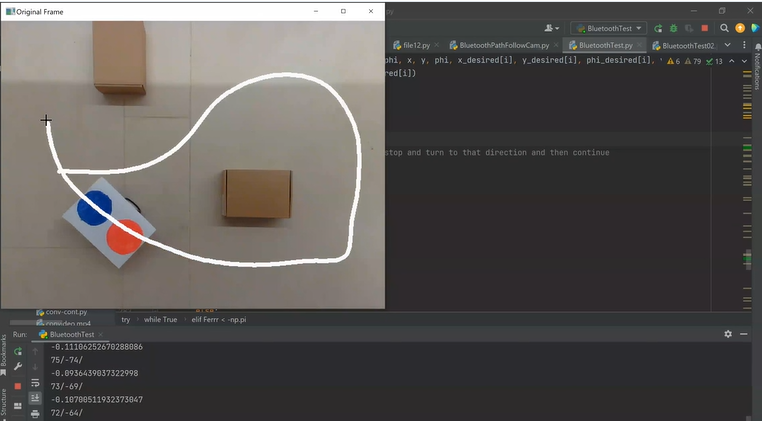


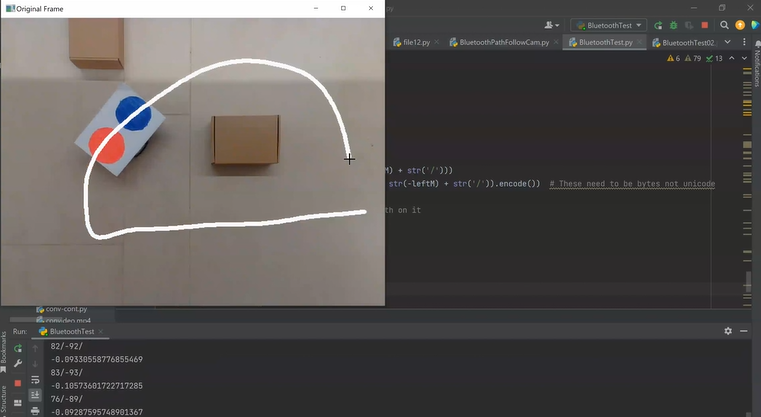
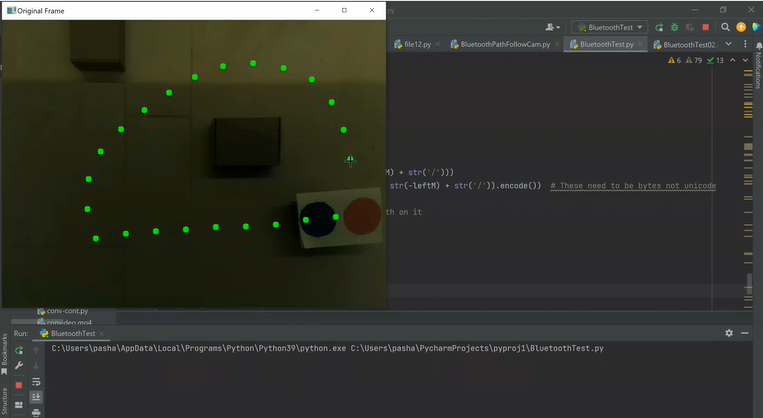


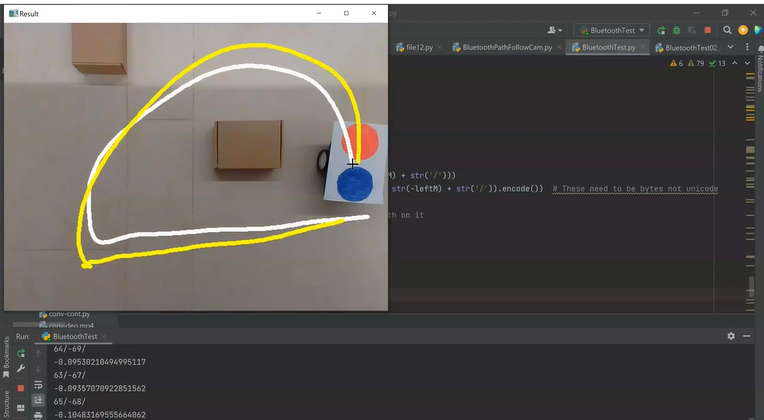












## *3.5 Economic analysis*

Conducting an economic analysis entails a comprehensive

evaluation to determine the mobile robot system’s

cost-effectiveness. Many vital aspects are take into account

in this assessment, such as the cost of hardware components,

the time needed for development, the cost of constant repair,

and any potential benefits the system may offer (like increased

productivity or automation). Initially, the study examines the

expenses associated with gathering all the hardware

components required for the mobile robot system. These cover

costs for other necessary hardware, such as communication

modules, processors, sensors, and other modules. Furthermore,

through evaluation of the development time is conducted.

The hours spent designing, prototyping, implementing, and

testing the mobile robot system by engineers, developers, and

designers are included in this assesment. The length of each

stage of development is further taken into account, as are the

related labor expenses. Along with related labor expenses,

the length of each stage of development is taken into account.

Thirdly, the lifetime maintenance costs of the system are

considered. Estimating costs for route maintenance, repairs,

software upgrades, and hardware replacements is a part of this.

By doing this, the system’s performance is optimized and

its constant operation is guaranteed. Finally, the mobile robot

system’s possible advantages are evaluated.

## *3.6 Social and environmental impact*

To start, the system records live video data, which is essential

for a real-time environment analysis. Following data

collection, the program processes data to find navigational

paths and pinpoint potential obstacles. Analyzing visual cues

in the surroundings to identify safe routes and steer clear of

hazards is most likely part of this step. This is considered

vital for autonomous vehicles or robots to operate. Then, it

goes on to explain how the environmental data is transformed

into movement instructions that can be followed. Functions

related to calculating velocities and control algorithm help

with this conversion. Based on these functions, it appears that

there is a feedback control system in place, which updates

movement based on input from the environment in real time

to provide safe and accurate navigation. With this

configuration, the autonomous system can dynamically adjust

its path to accommodate changes in its surroundings.

There are significant socioeconomic consequences to the use

of autonomous navigation systems, particularly in the areas

of facility management, delivery services, and transportation.

As these systems become more widely used, there is an

opportunity that jobs that are normally held by human drivers

and navigators will be significantly disrupted. These

modifications might have an effect on job prospects in these

sectors, leading to a change in job requirements and possibly

retraining employees to operate new technologies that improve

and make use of the capabilities of these systems.

The robotic system can communicate with other human

operators via COM ports and Bluetooth due to the hardwares,

such as camera and Bluetooth communication devices.

Employees may handle multiple autonomous systems at once

rather than just one through manual operation as a result of

this connectivity, which could replace manual labor with

more technical maintenance and monitoring duties. The

autonomous systems navigate real-time data efficiently. The

system can detect navigation cues and calculate the distances

between data points. Because of this move toward real-time

automated decision-making systems, jobs roles may become

more strategic oversight-focused on human decision-making

becomes less necessary in operational settings. Furthermore,

as work in an automated environment changes, there may be

greater need for qualified experts for the growing automation.

Next, we emphasized the importance of error control and

flexibility in autonomous systems, especially with regard to

control parameters such as Kx, Ky, and K\_phi. These

parameters suggested us the existence of an adaptable control

system that can alter its path in response to environmental

feedback. Due to their increased adaptability, autonomous

robots are able to navigate complex environments with

greater efficiency and dependability. In the past, jobs in

environments like factories or cities could have been difficult

without human sense and flexibility.

In human-centric environments, safety and efficiency are

prioritized through the integration of real-time image

processing and feedback mechanism in autonomous systems.

By integrating technology better, productivity is increased and

accidents are prevented while responsiveness in changing

conditions have improved. But it could also result in changes

to the roles that employees hold, with an emphasis on

monitoring, preserving safety, and correct operation of

autonomous systems. Consequently, employees might need

to keep or pick up new skills to adapt.

# 4. Implementation

## *4.1 Hardware Design*

For our project, we used an educational robot called mBot.

mBot is an award-winning programming robot developed

by Makeblock. The chassis of the MakeBlock mBot is

constructed from high-quality ABS plastic, providing both

durability and lightweight characteristics. Its dimensions

measure precisely 17 cm in length, 13 cm in width, and 9

cm in height, providing ample space for component

mounting and maneuverability. The chassis features

multiple mounting holes and slots, allowing for easy

attachment of additional components and accessories. For

our project, specifically, we attached a thin A4 paper that

has two identical sized circles: one colored red and one

colored blue. The high-quality ABS plastic chassis of the

MakeBlock mBot serves as the foundation for integrating

all other components. Using the multiple mounting holes

and slots on the chassis, the motors, wheels,

microcontroller board, and other accessories are securely

attached. The thin A4 paper with two identical-sized

circles, one colored red and one colored blue, is affixed to

the top surface of the chassis using adhesive or clips. This

paper serves as the visual reference for the robot’s image

processing system, allowing it to detect and track the

colored circles for navigation. Robot’s sleek and

streamlined design minimizes air resistance and improves

overall aesthetics.

The MakeBlock mBot is equipped with two powerful

geared DC motors, each rated at 6V and capable of

delivering a maximum torque of 2.5kg.cm. These motors

feature built-in encoders for precise speed and position

control, enabling accurate navigation and motion planning.

The motors are paired with high-traction rubber wheels,

measuring 6.5cm in diameter and 2.2cm in width,

providing excellent grip on various surfaces. Each wheel

is mounted on a metal hub with a ball bearing for smooth

and efficient rotation, minimizing friction and wear. Each

of the two powerful geared DC motors is mounted onto

the chassis using motor brackets and screws. The motors

are aligned and positioned such that their shafts

protrude through the mounting holes in the chassis,

allowing for direct connection to the wheels. The

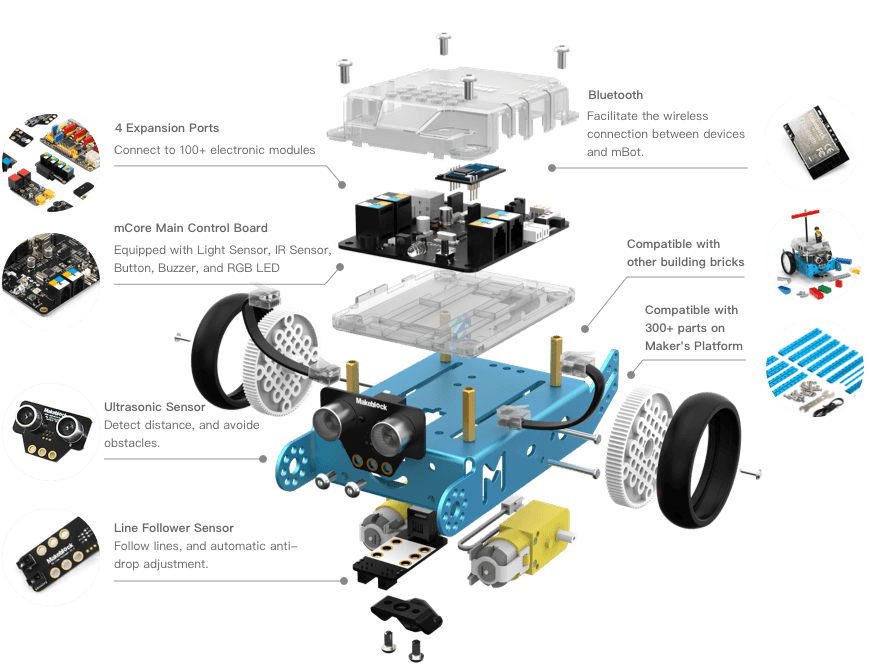
high-traction rubber wheels are then attached to the motor

shafts using wheel hubs or couplers. The wheels are

securely fastened to the motors to ensure smooth and

efficient rotation, minimizing any wobbling or

misalignment during operation.



The MakeBlock mBot is powered by an

Arduino-compatible microcontroller board, specifically

the MakeBlock Orion board (shortly mCore), based on

the ATmega328P microcontroller. The board features a

dual H-bridge motor driver (TB6612FNG), capable of

controlling the speed and direction of two DC motors

independently. Integrated electronics include an

ultrasonic sensor for obstacle detection, an infrared

receiver for remote control, and a line-following sensor

for autonomous navigation. The microcontroller board

also includes a Bluetooth module (HC-06) for wireless

communication with external devices, such as

smartphones or tablets. For our project, we utilized

mBot’s this capability to an extended degree. A laptop,

on which our Python script is running, connects to the

robot through Bluetooth, namely COM port. The

MakeBlock Orion board (mCore), serving as the main

microcontroller, is mounted onto the chassis using

screws. The microcontroller board is positioned

centrally within the chassis to optimize weight

distribution and balance. Integrated electronics,

including the dual H-bridge motor driver (TB6612FNG),

ultrasonic sensor, infrared receiver, and line-following

sensor, are connected to the microcontroller board via

appropriate headers, connectors, or jumper wires. The

Bluetooth module (HC-06) is also connected to the

microcontroller board, allowing for wireless

communication with external devices. The

microcontroller board may also feature mounting points

for additional accessories, such as the optional camera

module, although it is not utilized in this specific project.

The MakeBlock mBot can be equipped with the

MakeBlock mBot Add-on Pack - Interactive Light &

Sound, which includes a 480p camera module. The

camera module features a CMOS image sensor with a

resolution of 640x480 pixels and a frame rate of 30

frames per second. It is mounted on a tiltable bracket,

allowing for adjustable viewing angles and perspectives,

with a range of motion of approximately 180 degrees

horizontally and 90 degrees vertically. The camera

module communicates with the microcontroller board

via a dedicated serial interface, providing real-time video

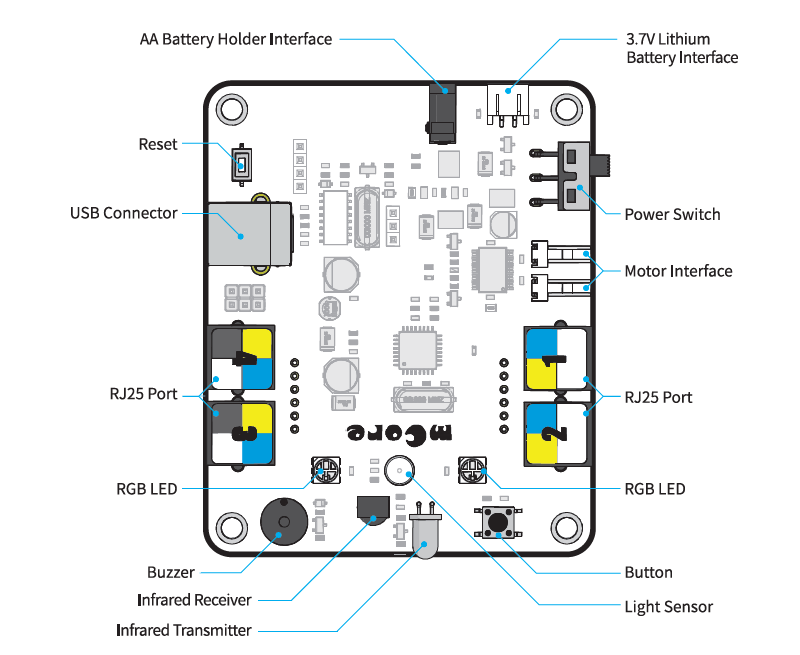
streaming and image capture capabilities. For our project,

we did not utilize this camera, instead, robot’s activity is

monitored through an overhead camera. In Software

design part, the connection between an external camera

and the robot will further be discussed.



The MakeBlock mBot is powered by a high-capacity

rechargeable lithium-ion battery pack, specifically a 3.7V

1800mAh LiPo battery. The battery pack provides

extended runtime of up to 8 hours on a single charge,

depending on usage conditions and operating parameters.

Charging is facilitated through a micro USB port located

on the microcontroller board, with a charging time of

approximately 2 hours. Additionally, the robot can also be

powered by four AA batteries (not included) for convenient

operation in situations where recharging is not feasible. The

high-capacity rechargeable lithium-ion battery pack is

securely housed within the chassis. The battery pack is

positioned in a designated compartment or slot, ensuring

stability and minimizing movement during operation.

Charging of the battery pack is facilitated through the

micro USB port located on the microcontroller board,

allowing for convenient recharging. By assembling these

components together in a systematic manner, the

MakeBlock mBot is transformed into a fully functional

robotic system capable of executing the desired tasks and

functionalities required for a wide range of projects.

In addition to its core components, the MakeBlock mBot

comes with a variety of additional features and accessories

to enhance its functionality and versatility. These include

an RGB LED matrix for visual feedback and customization,

a buzzer for audible alerts and notifications, and an

expansion port for connecting external sensors and modules.

Optional accessories, such as gripper arms, distance sensors,

and temperature sensors, can be easily integrated into the

robot’s design, expanding its capabilities for specific

applications. The MakeBlock mBot is compatible with

MakeBlock’s proprietary programming software, mBlock,

which provides a user-friendly interface for coding and

customization using graphical programming blocks or

Arduino-compatible C/C++ code. In our project, we

mainly coded on more advanced computers using Python

programming language and did not use this integrated

software. MBot is pre-programmed using mCore, which is

identical to Arduino UNO but has a built-in motor driver.

After establishing Bluetooth connection, strings are

constantly sent to this microcontroller in order to maintain

speed of two motors throughout the path. This connection

will be discussed in great detail further into the

Implementation part.

## *4.2 Software Design*

The software design of our robotic system encompasses

various components and functionalities to achieve the

desired behavior of following a predefined path based on

visual cues. The implementation is based on Python

programming language, utilizing libraries such as OpenCV,

NumPy, and Matplotlib for image processing, numerical

computations, and visualization.

The software architecture follows a modular design,

comprising several key components; Camera Interface is

responsible for capturing video frames from the overhead

camera mounted in the room, utilizing OpenCV’s

`VideoCapture` class to access the camera feed. Then, the

serial connection with the robot is initialized using the

specified COM port (COM4) and baud rate (115200).

bluetooth.flushInput() flushed the input buffer and send

initial control signals to the robot. Path Drawing

implements a mouse event callback function to allow the

user to draw a desired path on the video feed. This

component facilitates user interaction and input for

defining the trajectory. Using HSV ColorSpace, used

define color ranges for detecting red and blue markers,

and initialize variables for tracking robot position and

orientation. Trajectory Interpolation performs spline

interpolation on the drawn path to generate a smooth

trajectory. Utilizes the `scipy.interpolate.splprep` function

for spline fitting. This function evaluates the spline fits for

1000 evenly spaced distance values and assigns the

interpolated coordinates to x\_d and y\_d. The program also

implements the closed-loop control algorithm to calculate

the desired velocity and angular velocity of the robot

based on its current position and orientation relative to the

desired path. The control algorithm adjusts the robot’s

motion to minimize the deviation from the path, computing

the linear and angular velocities based on control gains Kx,

Ky, and K\_phi, current position x, y, and phi, desired

position values, as well as desired velocities. Wheel

velocity is calculated momentarily and converts the

desired linear and angular velocities into wheel velocities

for the left and right wheels of the robot. This component

ensures that the robot moves with the desired speed and

direction. Color Detection detects specific colors (red and

blue) in the video feed to locate visual markers placed on

the robot’s path. This component enables the robot to

recognize reference points for navigation and is a crucial

part of our project. Our program also establishes serial

communication with the robot via Bluetooth to send

control commands, utilizing the `serial` library to

communicate with the robot’s microcontroller.

Python is the primary programming language used for

implementing the software components due to its ease of

use, extensive libraries, and versatility in scientific

computing and robotics applications. OpenCV provides

functionalities for image and video processing, including

camera access, color detection, and drawing overlays on

video frames. NumPy offers support for numerical

computations and array manipulation, essential for

handling image data and performing mathematical

operations in the control algorithm. Matplotlib is used for

data visualization, particularly in plotting the trajectory

and visualizing the robot’s motion on the video feed.

The heart of the software design lies in the closed-loop

control algorithm, which determines the robot’s motion

based on its deviation from the desired path. The

algorithm calculates the desired linear and angular

velocities by comparing the robot’s current position and

orientation with the interpolated trajectory. Proportional

and derivative control parameters are adjusted dynamically

based on the distance from the desired path, allowing the

robot to navigate smoothly even in challenging conditions.

In detail, closed loop control means that we continuously

measure the output (x, y, phi) by using camera and change

our inputs v and omega in accordance with them, contrary

to open-loop control, where the control action is

determined solely by the input command without

considering the system’s output or response. Here the

algorithm for processing measurements and producing

proper control inputs is critical and is called Closed Loop

Controller. The Closed Loop Controller involves the

following steps:

- Calculation of the heading error (difference in

orientation) between the robot’s current orientation and

the desired orientation at each point along the trajectory.

- Adjustment of the robot’s linear and angular

velocities based on the heading error and other parameters

such as proportional gains (`Kx`, `Ky`, `K\_phi`), desired

velocities (`v\_desired`, `omega\_desired`), and physical

parameters of the robot (`a`).

- Transformation of the desired linear and angular

velocities into velocities for the left and right wheels of

the robot using kinematic equations.

- Sending the wheel velocities to the robot's actuators

(motors) to execute the desired motion.

The software performs image processing algorithm to

detect specific color markers (red and blue) in the camera

feed. It converts the captured frame from the camera to the

HSV color space for better color segmentation and

thresholding. A range of HSV values is defined for both

red and blue colors to create masks that isolate these colors

in the image. The software then calculates the centroid

coordinates of the detected color markers using the

moments of the color mask. These centroid coordinates are

used to determine the position of the color markers in the

image, which in turn guides the robot along the desired

trajectory.

Mouse event-based curve drawing algorithm allows the

user to define a curved trajectory using mouse events on

the graphical interface. When the user clicks and drags the

mouse, the software records the mouse cursor’s

coordinates to define the trajectory. The trajectory is

stored as a series of points, which are later interpolated to

create a smooth curve. This functionality enables the user

to specify custom trajectories for the robot to follow.

The software establishes communication with a Bluetooth

module to send control commands to the robot. It opens a

serial connection to the Bluetooth port specified by the

user. The control commands, consisting of left and right

wheel velocities, are sent to the robot via the Bluetooth

serial connection. This communication enables real-time

control of the robot's motion based on the calculated

velocities.

Overall, the software integrates control algorithms, image

processing techniques, user interaction, and

communication protocols to facilitate the robot’s

autonomous navigation along a predefined curved

trajectory. It provides a flexible and interactive platform

for controlling and guiding the robot's movement in

real-world environments.

The software includes robust exception handling

mechanisms to handle unexpected errors or interruptions.

For example, the program can be terminated cleanly using

keyboard interrupts (Ctrl+C), ensuring proper shutdown

procedures and resource release. The try block

encapsulates the main logic of the program, where it

continuously processes frames from the camera and

controls the robot’s movement. The except block

specifically handles the KeyboardInterrupt exception,

which occurs when the user interrupts the program, in

the terminal. When this exception is raised, the program

prints “Program is interrupted!” before exiting. This

ensures that even if the user interrupts the program, it will

still clean up resources and provide feedback to the user.

The software provides a user-friendly interface for

interacting with the robot, allowing users to draw custom

paths using a mouse interface. This intuitive approach

simplifies the process of defining complex trajectories and

enables real-time adjustments during operation.

Overall, the software design emphasizes modularity,

flexibility, and user-friendliness, enabling seamless

integration of image processing, control algorithms, and

user interaction to achieve precise and reliable robotic

navigation based on visual feedback.

## *4.3 Essential Components of The Project*

Image Acquisition section is responsible for capturing

video frames from the camera using the OpenCV library.

It initializes the camera using `cv2.VideoCapture(2)`,

where `2` denotes the camera index. The captured frames

serve as the visual input for subsequent image processing

and control. Without image acquisition, the software lacks

visual input essential for trajectory planning and

navigation. The robot would be unable to perceive its

surroundings, rendering it incapable of autonomous

navigation or interaction with the environment.

Mouse event handling enables user interaction with the

graphical interface to define the desired trajectory for the

robot. The `draw\_curve\_with\_mouse\_events` function

responds to mouse events (such as button clicks and

movements) within the designated window. It records the

trajectory points as the user clicks and drags the mouse,

ensuring that points are sufficiently spaced apart to avoid

overcrowding. Absence of mouse event handling would

prevent user interaction and trajectory definition. Users

would be unable to specify the desired path for the robot,

limiting the software's flexibility and usability.

After capturing the trajectory points from user input, the

software utilizes spline interpolation techniques to create

a smooth curve passing through these points. The

`scipy.interpolate.splprep` function fits a spline curve to

the trajectory points, ensuring smoothness while

preserving the overall shape of the curve. The interpolated

curve represents the desired trajectory that the robot

should follow, providing a continuous path for navigation.

Without trajectory definition and interpolation, the

software cannot generate a smooth and continuous path

for the robot to follow. The robot’s motion would lack

coherence and could result in erratic behavior, making

navigation challenging and imprecise.

The control system governs the robot's motion along the

defined trajectory. It calculates the robot’s velocity and

angular velocity based on the difference between its

current pose and the desired trajectory. Proportional

control mechanisms with adjustable gains (`Kx`, `Ky`,

`K\_phi`) fine-tune the robot's response to trajectory

deviations, ensuring smooth and accurate motion. Lack of

a control system would hinder the software’s ability to

regulate the robot’s motion and maintain its position along

the trajectory. The robot’s movement would be

uncontrolled, leading to deviations from the desired path

and potentially causing collisions or navigation errors.

Image processing techniques are employed to detect

specific color markers (red and blue) in the camera feed,

aiding in robot navigation. The software converts captured

frames to the HSV color space for better segmentation, as

it separates hue, saturation, and value components. Color

segmentation using thresholding isolates regions of interest

corresponding to the red and blue markers. Centroid

coordinates of the detected color markers are determined to

provide guidance for the robot’s motion. In the absence of

image processing, the software cannot detect visual cues or

landmarks for navigation. The robot would be unable to

identify obstacles, landmarks, or reference points, limiting

its ability to navigate autonomously and adapt to changing

environments.



Bluetooth communication facilitates the transmission of

control commands from the software to the robot. A serial

connection is established with a Bluetooth module,

specifying the COM port and baud rate (`115200`).

Velocity commands (left and right wheel velocities) are

encoded as bytes and transmitted via the Bluetooth serial

connection to drive the robot's motion. Without Bluetooth

communication, the software cannot transmit control

commands to the robot. The robot would be

non-responsive to navigation instructions, rendering the

software unable to influence its motion or behavior

remotely.

Exception handling mechanisms are incorporated to

manage unexpected errors or interruptions during program

execution. The software handles KeyboardInterrupt,

ensuring a graceful exit when the user interrupts the

program execution, preventing potential issues with

resource cleanup or data corruption. Absence of exception

handling mechanisms leaves the software vulnerable to

runtime errors or interruptions. Unexpected errors or user

interruptions could cause program crashes, data

corruption, or resource leaks, compromising the

software’s reliability and robustness.

The graphical interface displays the camera feed,

trajectory, and robot’s motion in real-time for user

feedback and monitoring. Trajectory, detected color

markers, and robot’s path are visualized on the interface

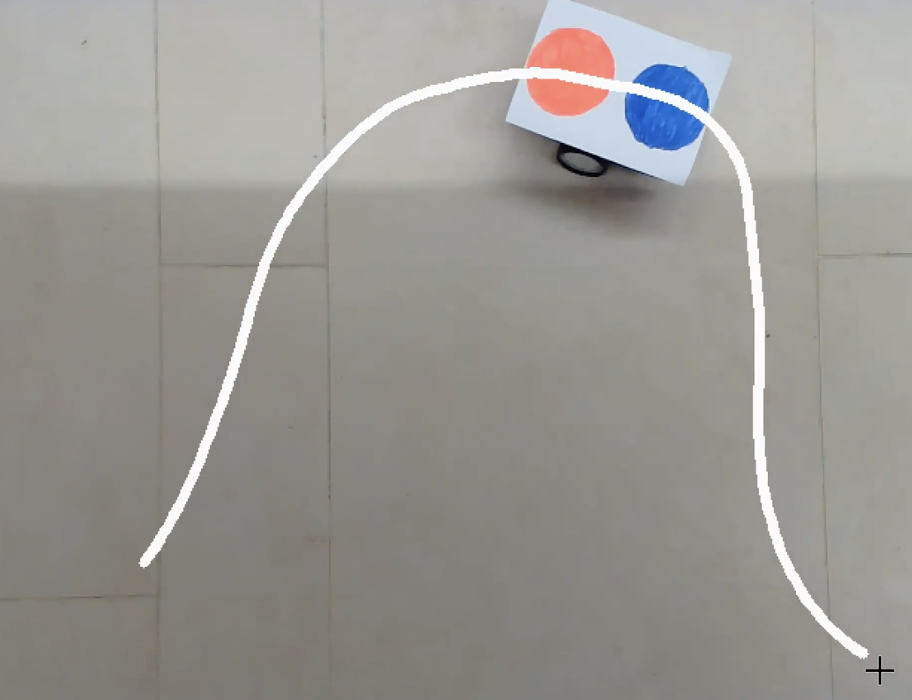
to aid in debugging and validation. Without a graphical

interface, users lack real-time feedback and monitoring

capabilities. Debugging, validation, and user interaction

become challenging, impeding the software’s usability and

hindering development and testing processes.



Robot kinematics calculations determine the velocities of

the robot’s left and right wheels based on desired linear

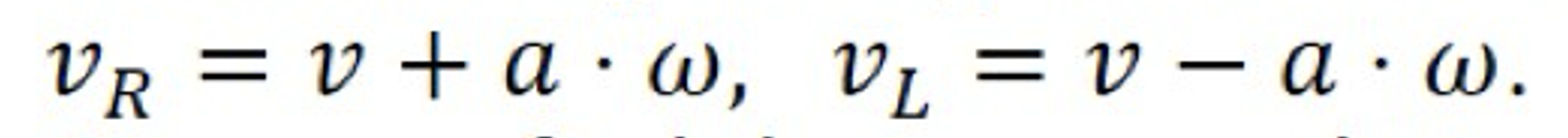
and angular velocities. Physical parameters of the robot,

such as the wheelbase (`a`), are considered to ensure

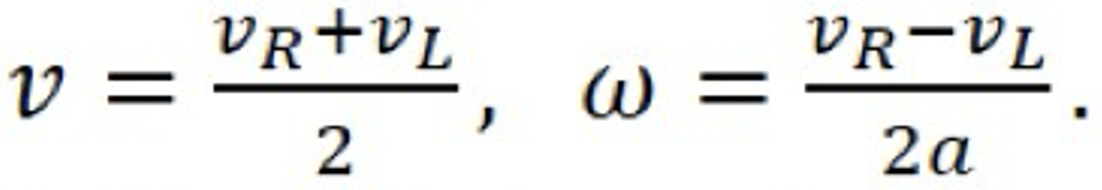
accurate motion control and trajectory tracking. If we write

velocity of right wheel and velocity of left wheel, in terms

of ,

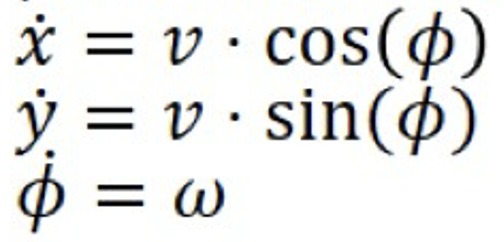


Or we can find the inverse relation:



Differential equations representing the system (Kinematic

model):



Now that we have a direct relation between wheel

velocities , we can assume that our inputs to the

system are which are directly related to R and L.

Without accurate robot kinematics calculations, the

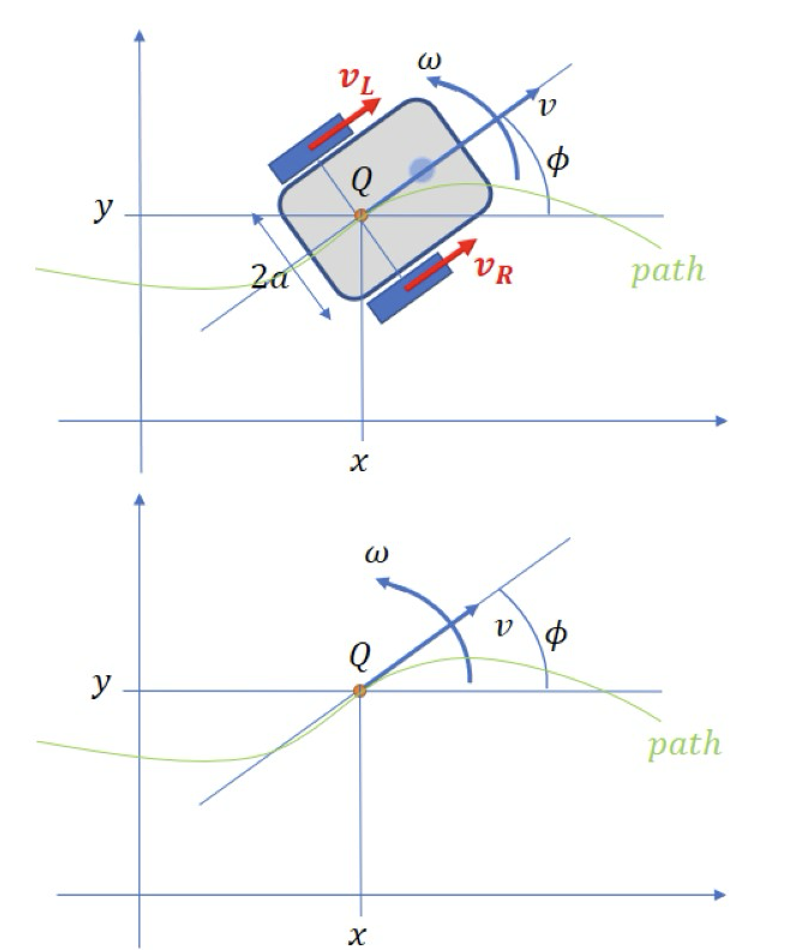
software cannot translate desired velocities into motor

commands effectively. The robot’s motion control would

be inaccurate, leading to poor trajectory tracking,

inefficient energy usage, and potential mechanical stress

on the robot’s components.



Each section plays a crucial role in the overall

functionality of the software, enabling the robot to

autonomously navigate along user-defined trajectories

while responding to environmental cues detected through

image processing. The integration of these components

ensures robustness, flexibility, and efficiency in the robot’s

navigation system.

*Timeline*

Week 1-3

- Modified code to handle mouse events for drawing a

curve.

- Implemented `draw\_curve\_with\_mouse\_events()`

function to store coordinates in `circles\_arr`.

- Optimized line drawing algorithm to visualize continuous

curve.

- Utilized Matplotlib to visualize the curve with NumPy

array conversion.

- Conducted research on advanced image processing

concepts like Distance Transformation and Voronoi

Diagram.

- Refined path generation code.

- Integrated Matplotlib for improved data representation.

- Explored advanced image processing techniques like

Distance Transformation and Voronoi Diagrams.

- Focused on computerized path generation while

collaborating with team members.

- Worked on getting image input and displaying it to get

start and ending points.

- Implemented marking of start and end points on the

image using mouse click events.

- Applied masking and bitwise transformations on the

image to enhance visibility.

- Utilized `cv2.distanceTransform()` to dilute the image

and make obstacles more visible.

- Imported and used the Voronoi library from Scipy for

path generation avoiding obstacles.

Week 4

- Imported the `curve\_fit()` function from the

`scipy.optimize` module to perform curve fitting.

- Declared a function `fit\_curve(x, a, b, c)` to represent

the quadratic equation, returning `a \* ((x + b) \*\* 2) + c`.

- Identified x- and y-coordinates of data points

(`lines\_arr\_np`) using NumPy arrays derived from the list

`lines\_arr`.

- Extracted x-coordinates from the first column of the

NumPy array using `x\_data\_points = lines\_arr\_np[:, 0]`.

- Extracted y-coordinates from the second column of the

NumPy array using `y\_data\_points = lines\_arr\_np[:, 1]`.

- Plotted the x and y data points as blue circles on a graph

using `plt.plot(x\_data\_points, y\_data\_points, 'bo')`.

- Demonstrated the shape of the generated `lines\_arr\_np`

array using `print(lines\_arr\_np.shape)`.

- Created a scatter plot of the data points using

`plt.scatter(x\_data\_points, y\_data\_points)`.

- Displayed the scatter plot using `plt.show()`.

- Utilized the `curve\_fit()` function to fit the data points

by providing the `fit\_curve()` function, x, and y data

points.

- Obtained two essential values, `popt` and `pcov`, where

`popt` contains the optimized parameter values (a, b, c)

estimated by the fitting process, and `pcov` displays the

covariance matrix or errors of the optimized parameters.

- Printed the optimized parameter values (`popt`) to

assess the accuracy of the fitting.

- Calculated the y-coordinates of the curve fitting function

using `fit\_curve(x\_data\_points, \*popt)`.

- Plotted the curve fitting function against the x-data

points using

`plt.plot(x\_data\_points, fit\_curve(x\_data\_points, \*popt))`.

Week 5

- Defined essential parameters such as `radius\_of\_robots`,

`radius\_of\_motors`, `duration\_of\_time\_steps`, `robot\_pose`, `motor\_velocities`, `numb\_of\_ticks\_per\_rev`,

and `motor\_encoder\_ticks` to facilitate kinematics

computations.

- `radius\_of\_robots` and `radius\_of\_motors` were defined

to represent the wheel radius and the distance between the

robot’s center and wheels, respectively.

- `duration\_of\_time\_steps` specifies the duration for each

time step during robot movement.

- `robot\_pose` initialized the robot's initial position and

orientation.

- `motor\_velocities` set initial velocities for both left and

right motors.

- Developed `calculate\_robot\_kinematics()` function to

update the robot’s position and direction based on linear

and angular velocities.

- Utilized `calc\_motors\_encoders()` function to calculate

encoder ticks for both left and right motors, considering

linear and angular velocities.

- Conducted in-depth research to understand the

theoretical concepts and practical implications of

differential drive systems.

- Investigated time-stepping in robot movement to control

motion through discrete time intervals.

- Applied control calculations to estimate the robot’s pose,

updating its position and orientation accurately.

- Integrated kinematics calculations into the existing

codebase, ensuring compatibility and functionality.

- Translated kinematics calculations into executable

Python code, leveraging NumPy for numerical

computations.

Week 6

- Created a function called `closed\_loop\_control()` to

compute control signals for the robot based on current and

desired coordinates, directions, and velocities.

- Modified the function to adjust velocity and angular

velocity calculations based on ratio control terms and

desired values.

- Developed the `calc\_wheels\_velocities()` function to

calculate velocities for both left and right wheels based on

linear and angular velocities and wheel radius.

- Consolidated color detection functionality into a single,

efficient function to simplify color identification in video

frames.

- Optimized the mouse callback function for drawing paths

on the video feed to improve user experience and path

creation.

- Streamlined the codebase by focusing on blue color

detection and mouse-based path drawing, eliminating

redundant variables and functions.

- Prepared for integrating robot kinematics calculations by

setting placeholders for future implementation of velocity

calculations based on color coordinates.

- Conducted a general cleanup of the code, improving

variable consistency and removing unnecessary comments

to enhance overall readability.

Week 7

- Modified the value of the variable 'a' based on robot

kinematics, ensuring accurate calculations of angular

velocity.

- Utilized curve smoothing to extract desired coordinates

for trajectory planning, including desired velocity, angular

velocity, and angle.

- Computed left and right wheel velocities (v\_left and

v\_right) based on pre-computed linear and angular

velocities, ensuring smooth and aligned motion with the

predefined trajectory.

- Planned to use Bluetooth for transmitting calculated

wheel velocities to an Mbot, ensuring continuous

adjustment of wheel motion until the specified endpoint is

reached.

- Collaborated to address recurring errors, refine the control

algorithm, and integrate code into the main program.

Week 8

- Addressed semantic issues in formulas-related lines of

code, resolving problems encountered with invalid values in

sqrt functions.

- Replaced multiplication operators with exponentiation

operators in relevant code lines.

- Utilized the interpolation method from the Scipy library to

smooth data points in the curve.

- Applied cubic spline interpolation to drawn curve’s x and y

coordinates using `scipy.interpolate.splprep()` function.

- Analyzed spline fits at evenly distributed distance values

along the spline curve using `scipy.interpolate.splev()`

function.

- Initialized a global variable, `lines\_arr`, to track mouse

movements and draw corresponding lines on the video feed

in real time.

- Updated the main loop of the script to ensure continuous

drawing based on user input.

- Refined the `draw\_curve\_with\_mouse\_events` function for

more responsive event handling, capturing mouse events

effectively for seamless drawing over a live video feed.

- Integrated real-time updates into the display output,

providing instant visual feedback to users.

- Addressed challenges such as minimizing latency

between cursor movement and drawing through code

optimization.

- Conducted a comprehensive analysis of the code to

discern disparities from previous iterations and identify

areas for refinement.

- Commenced work on the final report, distributing tasks

among team members and laying the groundwork for

crafting a comprehensive document.

- Aimed to encapsulate the project journey, methodologies,

findings, and conclusions in the final report.

Week 9

- The control algorithm computes inputs, such as linear

and angular velocities, to direct the robot along the

intended path. A newer iteration of the control algorithm

was implemented into the main source code.

- Newer version has a feedback mechanism to make

adjustments based on the disparity between the robot’s

present pose and the intended trajectory.

- The response time of the robot to path deviations is

determined by parameters such as Kx, Ky, and K\_phi,

representing improvements in the control system.

- Significant variables such as time step, robot dimensions,

and curve coefficients are specified by the camera.

Variables were toggled for better recognition.

- The new algorithm determines the required velocity and

angular velocity by comparing the robot's position with the

desired path. It then utilizes the intended motion profile to

calculate each wheel's velocity, ensuring proper movement

and guidance along the desired path.

- Real-time analysis was applied for obstacle detection,

allowing the program to modify the trajectory as needed to

navigate around obstacles.

- The updated program aims to ensure safe navigation

through complex and cluttered environments, adjusting to

changing circumstances in real time.

Week 10

- Our team, guided by our professor, focused on refining

and perfecting our project’s codebase.

- The main goal was to ensure that the final code was

flawless, error-free, and met the highest standards of

quality.

- Additionally, we conducted research to explore

innovative solutions and best practices in software

development.

- In recent weeks, we’ve been working on refining our

project, focusing on improving the interaction between

color detection, path tracking, and robot kinematics for a

Differential Drive Mobile Robot.

- The groundwork for integrating robot kinematics

calculations was laid, setting placeholders for future

implementation.

- Concurrently, work continues on the final report,

ensuring it encapsulates our project journey,

methodologies, findings, and conclusions.

Week 11

- Set up communication with a Bluetooth device using

serial communication.

- Defined a port variable and set its value to "COM4",

corresponding to the serial port used for the Bluetooth

connection.

- Established the Bluetooth connection with a baud rate of

115200 using the serial.Serial() function.

- Ensured a clear beginning to communication by

removing any data collected in the input buffer of the

Bluetooth device with bluetooth.flushInput().

- Transmitted the initial command "0/0/" to the Bluetooth

device using bluetooth.write() to indicate a particular

action or procedure.

- Declared a variable named st and set it to 0 to keep data

points separated from each other in the lines\_arr collection.

- Modified the mouse events function to determine

whether it’s the first click of the left mouse button and add

coordinates to the lines\_arr list.

- Ensured that new data points are not too close to the old

ones to prevent an excessive number of points from being

collected.

- Completed the drawing process when the mouse is

released by assigning stop to 1.

- Introduced the Ts variable representing the sampling

time, indicating how much time passed between each task

implemented by the robot.

- Understanding Ts is crucial for determining the speed at

which the robot completes each task and ensuring efficient

operation.

Week 12

- Testing the robot in the lab room with our mentor.

- Making modifications to the color ranges for red and

blue to accurately detect the robot’s paths.

- Adjusting lower and upper boundaries for both blue and

red color ranges in the HSV color space.

- Facilitating color detection by converting the color

format from BGR to HSV.

- Refining the code snippet to identify red areas in the

video and observe the robot's movement.

- Enhancing algorithm performance and efficiency

through code evaluations and optimizations.

- Decreasing resource consumption and improving

execution speed.

- Successfully integrating our device with the robot

through Bluetooth connectivity for data transfer and

commanding, laying the foundation for remote control

and automation capabilities.

Week 13

- Conducted testing with instructor in lab room to

evaluate robot performance

- Made slight adjustments to K values and experimented

with different robot paths

- Began drafting project’s final report

- Modified code to prioritize freezing and arriving at

desired position when robot is far from target

- Dynamically adjusted parameter K\_phi based on

difference between current and desired positions

- Decreased K\_phi when robot is far from target to

minimize directional errors' influence on control input

- Increased K\_phi as robot approached target point to

improve responsiveness to direction corrections

- Implemented calculation of error between desired and

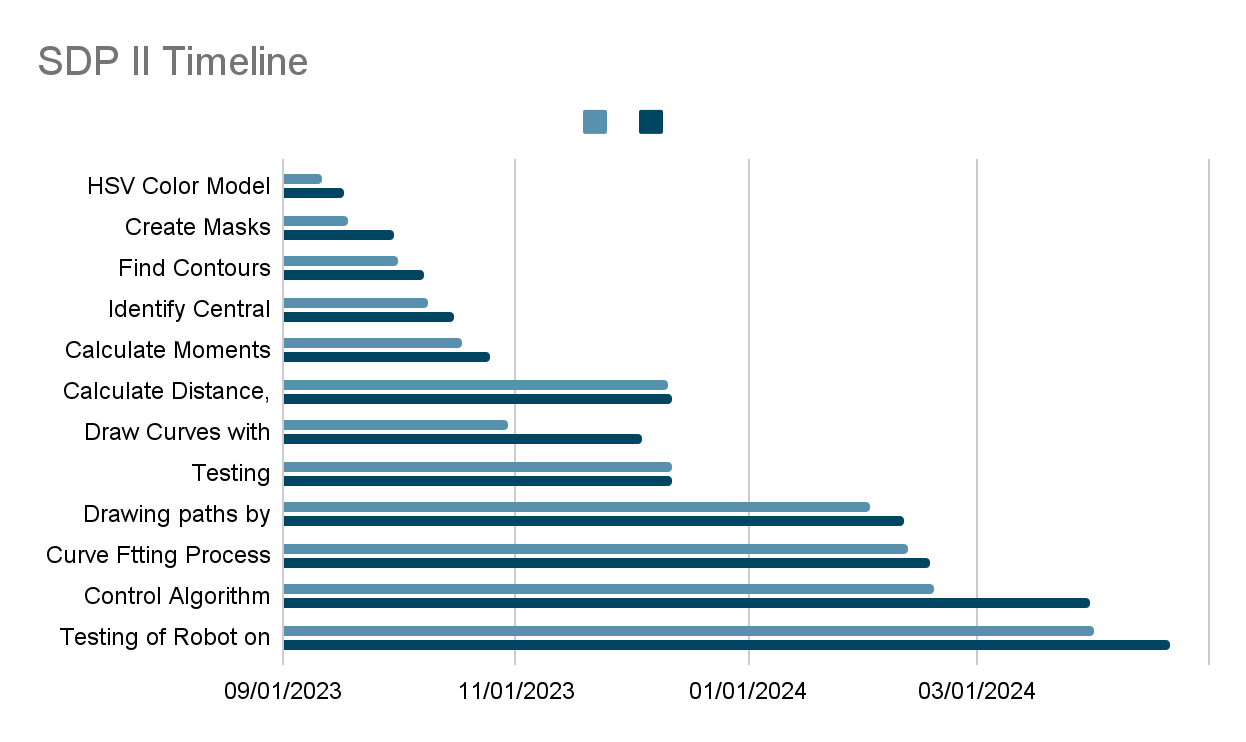
current direction (Ferr = phi\_desired - phi)

- Ensured error fell within range of -π to π for

consistency in directional adjustments

- Aimed to enhance robot's navigation capabilities and

improve ability to traverse desired trajectory efficiently

Table 1.1 Gantt Chart

## *4.4 Testing/Verification/Validation*

At the start of our project, even before we had access to

the robot, our main task was to complete path generation.

In the course of 4 weeks, we did many experiments in this

part of our project. In the older iterations of the source

code, we tried line computing, linear regression, Voronoi

diagrams, quadratic dot multiplication, etc. Detailing the

testing procedures used to verify the functionality and

performance of our robotic system involved conducting

various tests both in the lab room and household

environments. We tested different control algorithms and

parameters, implemented real-world scenarios, and

utilized manual control inputs and automated scripts.

Additionally, we employed Bluetooth communication to

remotely command and monitor the robot’s actions.

After tackling initial problems, our first test run went

extremely well. The user interface we made recognized

user input and generated a virtual path for the robot to

follow. The camera did not have any problems

recognizing the colors on the mBot’s chassis and sent

secondly data to the computer concerning its position.

The robot did not have any issues while communicating

with the camera or the computer. However, the project

was far from gone, as there were still few obstacles.

Firstly, the room in which the experiment was conducted

was ideally bright, therefore we had no idea how the

image detection would perform under dimmer light.

Secondly, the would the robot be able to complete the

path if it was longer or had more acute turns. What

about obstacles with different colors?

To validate the results against the project objectives and

requirements, we compared the observed performance

with the predefined criteria. We verified that the robot

successfully navigated along desired paths, avoided

obstacles, and responded appropriately to external

commands. We also assessed the accuracy and reliability

of trajectory following, color detection, and path tracking

functionalities.

During the testing phase, we set up experimental

configurations and collected data on the robot’s position,

velocity, and sensor readings. We used overhead camera

to capture video footage of the robot's movements and

employed sensors to gather additional information about

the environment. Performance metrics such as path

deviation, execution time, and error rates were measured to

quantify the system’s behavior and effectiveness.

According to recorded data, our most recent experiments

indicates 2-2.5 cm path deviation, 340 msec execution

time, and no errors.

Throughout testing, we encountered challenges related to

Bluetooth connectivity, sensor calibration, and algorithm

tuning. These issues were addressed by ensuring proper

pairing and configuration of Bluetooth devices, adjusting

sensor calibration parameters, and fine-tuning control

algorithms based on observed performance and feedback.

The mBot had irregular Bluetooth protocols that made

the testing period harder for us. So, if the mBot is

connected to a certain device, it cannot be connected to

another device through wireless or cable connection

even if the primarily connected device is out-of-range.

For some computers, the mBot did not even appear in

discoverable device list despite being in pairing mode.

Most of these problems were solved when we installed

mBot’s official software and carried out the pairing on

its application. To tackle the lighting problem, we

implemented camera filters, color mask, and variables

to modify contrast level easily from inside the program.

The ideal brightness setting was no longer a requirement

for our robot to function properly. The last significant

issue was the irregular movement and path deviation at

several points along the way. Through trial and error, we

tried to find the most optimal K\_phi value for our program

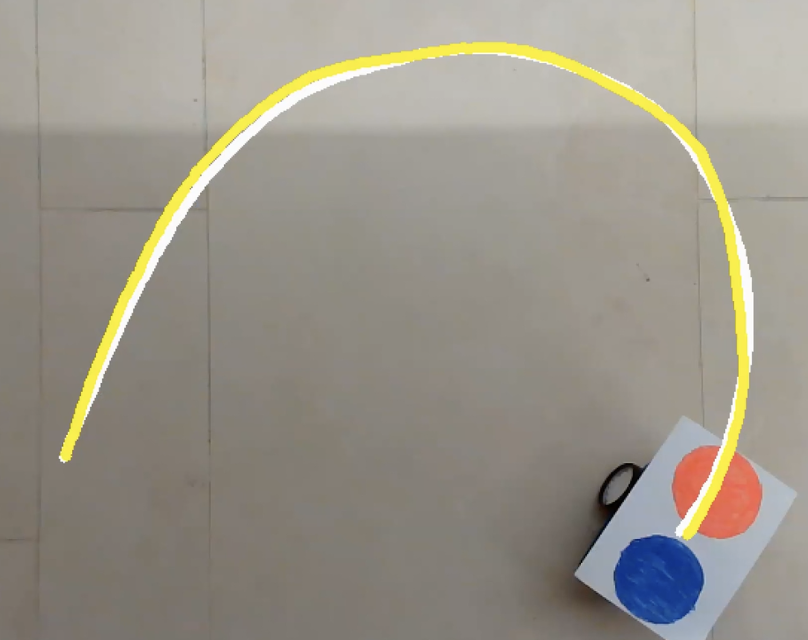
using which would give us the most optimal results with

less path deviation. Overall, these efforts contributed to

refining the robotic system and ensuring its functionality

met project requirements.





# 5. Conclusion

Firstly, we observed that the control algorithms and

parameters significantly influenced the robot’s navigation

and trajectory following capabilities. Fine-tuning these

parameters allowed us to optimize the robot’s performance

and ensure smooth movement along desired paths.

Secondly, we found that the integration of color detection

and path tracking functionalities provided valuable insights

into the robot’s environment and enabled it to adapt to

changing conditions. By accurately identifying colors and

tracking paths, the robot could effectively navigate through

complex environments and avoid obstacles. Additionally,

we discovered that the use of Bluetooth communication

facilitated seamless interaction between the robot and

external devices. This enabled remote control and

monitoring of the robot’s actions, enhancing its versatility

and usability in various applications. Furthermore, our

testing revealed the importance of robustness and

reliability in the robotic system. By conducting extensive

testing and validation procedures, we ensured that the

system performed consistently under different conditions

and scenarios, meeting the project’s objectives and

requirements. Overall, the implementation and testing of our

robotic system highlighted the importance of algorithm

optimization, sensor integration, and communication

protocols in achieving successful navigation and control.

These findings provide valuable insights for future

development and refinement of robotic systems for

real-world applications.

The performance of the system in achieving its objectives

and meeting the project requirements can be analyzed

across several key aspects. One of the primary objectives

of the project was to develop a robotic system capable of

accurately navigating predefined paths. Through extensive

testing and validation, we assessed the system's ability to

follow designated trajectories. By comparing the robot’s

actual path with the desired path, we evaluated its

navigation accuracy. The system’s performance in this

regard was crucial for ensuring that the robot could

traverse complex environments and reach specific

destinations reliably. Another important aspect of the

project was to implement obstacle detection and avoidance

mechanisms to enhance the robot's autonomy and safety.

We evaluated the system’s ability to detect obstacles in its

path and make necessary adjustments to avoid collisions.

By simulating various obstacle scenarios and monitoring

the robot’s responses, we assessed the effectiveness of the

obstacle avoidance algorithms. The system’s performance

in this aspect was critical for ensuring safe navigation in

dynamic environments. The project requirements

emphasized the importance of real-time responsiveness in

the robotic system. We analyzed the system’s latency and

response times during different tasks and operations. By

measuring the time taken for the robot to react to external

stimuli or user commands, we evaluated its real-time

performance. The system's ability to promptly respond to

changes in its environment or inputs was essential for

achieving smooth and efficient operation. Given the

integration of Bluetooth communication for remote control

and data transfer, we assessed the reliability of

communication between the robot and external devices.

We evaluated the system's ability to establish and maintain

a stable connection, transmit commands and data accurately,

and handle communication errors or interruptions. The

reliability of communication was crucial for ensuring

seamless interaction between the robot and its operators or

other connected devices. Overall, the performance of the

system was evaluated based on its ability to fulfill the

project objectives and requirements in terms of navigation

accuracy, obstacle avoidance, real-time responsiveness,

and communication reliability. By conducting thorough

testing and analysis across these key aspects, we gained

insights into the system’s capabilities and identified areas

for improvement to enhance its overall performance.

Throughout the project, we encountered both notable

successes and shortcomings, which provided valuable

insights and opportunities for improvement. One of the

notable successes of the project was achieving high

precision in navigation. Through tuning of control

algorithms and trajectory planning, we were able to guide

the robot along predefined paths with remarkable

accuracy. This precision was essential for tasks requiring

the robot to traverse specific routes or reach precise

locations within an environment. Implementing effective

obstacle avoidance mechanisms was another success of

the project. By integrating sensors and algorithms capable

of detecting obstacles in the robot's path, we successfully

enabled the robot to autonomously navigate around

obstacles while maintaining its trajectory. This capability

enhanced the robot's safety and autonomy, allowing it to

operate in dynamic environments with minimal human

intervention. Establishing reliable communication

between the robot and external devices via Bluetooth was

a significant success. We ensured stable and consistent

communication channels for transmitting commands,

receiving sensor data, and interacting with the robot

remotely. This communication reliability was crucial for

enabling seamless interaction between the robot and its

operators, facilitating remote control and data exchange.

One of the key shortcomings encountered during the

project was related to real-time responsiveness. Despite

efforts to optimize the system's response times, we

observed occasional delays in the robot's reactions to

external stimuli or user commands. These delays, although

minor, affected the system's overall performance in

dynamic environments and required further optimization

to improve real-time responsiveness. Another challenge

was the limited coverage of camera, particularly in

detecting obstacles from certain angles or distances. This

limitation occasionally resulted in blind spots where the

robot’s obstacle avoidance mechanisms were less effective.

Addressing this issue required exploring alternative sensor

configurations or implementing additional sensor fusion

techniques to enhance the robot's perception capabilities.

Developing and fine-tuning complex control algorithms

posed a challenge, particularly in balancing the trade-offs

between navigation accuracy, speed, and stability.

Achieving optimal performance required extensive

experimentation and tuning, which consumed significant

time and resources. Simplifying control algorithms

without compromising performance could be explored to

mitigate this challenge in future iterations. Overall, while

the project achieved notable successes in navigation

precision, obstacle avoidance, and communication

reliability, addressing shortcomings related to real-time

responsiveness, sensor coverage, and algorithm

complexity remains essential for further enhancing the

system’s performance and robustness.

Based on the insights gained from our project, several

potential areas for further research or development have

emerged, which could significantly advance the field of

robotics and related domains. Future iterations of the

robotic system could focus on further optimizing control

algorithms to enhance navigation accuracy, real-time

responsiveness, and stability. This optimization may

involve refining existing algorithms, exploring new control

strategies, or implementing machine learning techniques to

adaptively adjust control parameters based on

environmental conditions. Improving the sensor

capabilities of the robotic system could enable more

comprehensive perception and situational awareness. This

may involve integrating additional sensors, such as LiDAR,

radar, or depth cameras, to augment existing sensor suites

and address blind spots or limitations in obstacle detection

and localization. Future iterations could aim to enhance the

autonomous decision-making capabilities of the robotic

system, enabling it to perform more complex tasks and

navigate dynamic environments with minimal human

intervention. This may involve developing advanced

planning and decision-making algorithms, such as path

planning in cluttered environments or adaptive navigation

strategies in unpredictable conditions. Exploring the

integration of advanced learning techniques, such as

reinforcement learning or imitation learning, could enable

the robotic system to acquire new skills and adapt its

behavior through experience. This could facilitate

autonomous learning and adaptation to changing

environmental conditions, leading to more robust and

versatile robotic systems. Investigating methods for

enhancing human-robot interaction could improve the

usability and effectiveness of the robotic system in

collaborative settings. This may involve developing

intuitive interfaces, natural language processing

capabilities, or haptic feedback mechanisms to enable

seamless communication and collaboration between

humans and robots. As robotics technology continues to

advance, it is essential to consider the ethical and societal

implications of widespread adoption. Future research

could explore frameworks for ethical robotics design,

guidelines for responsible deployment of robotic systems,

and strategies for mitigating potential societal impacts,

such as job displacement or privacy concerns. Overall,

pursuing these future directions could lead to significant

advancements in robotics technology, enabling the

development of more capable, intelligent, and socially

responsible robotic systems that can address a wide range

of applications and challenges in diverse domains.

# References

[1] Bansal, I. (2023, March 23). Color Detection using Python -

Beginner's Reference - AskPython. *AskPython*.

<https://www.askpython.com/python/examples/color-detection>

[2] Brownlee, J. (2021, November 14). *Curve fitting with*

*Python*. MachineLearningMastery.com.

<https://machinelearningmastery.com/curve-fitting-with-python/>

[3] GeeksforGeeks. (2023, March 22). *Multiple Color Detection*

*in Real-Time using Python-OpenCV*. GeeksforGeeks.

[https://www.geeksforgeeks.org/multiple-color-detection-in-real-t](https://www.geeksforgeeks.org/multiple-color-detection-in-real-time-using-python-opencv/)

[ime-using-python-opencv/](https://www.geeksforgeeks.org/multiple-color-detection-in-real-time-using-python-opencv/)

[4] *How to define “lower” and “upper” range of two (or more)*

*different color? – OpenCV Q&A Forum.* (n.d.).

<https://answers.opencv.org/question/181705/how-to-define-lower-and-upper-range-of-two-or-more-different-color/>

[5] *Image Thresholding — OpenCV-Python Tutorials beta documentation*. (n.d.). <https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html#thresholding>

[6] *Object Detection using OpenCV and Python*. (2021, December 17). IEEE Conference Publication | IEEE Xplore.<https://ieeexplore.ieee.org/document/9725638>

[7] *OpenCV: Cascade Classifier*. (n.d.).  <https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html>

[8] *OpenCV Documentation.* (2009, May 7). (n.d.).<https://cvhci.anthropomatik.kit.edu/download/visionhci09/opencv.pdf>

[9] *OpenCV: Drawing functions in OpenCV*. (n.d.). <https://docs.opencv.org/3.4/dc/da5/tutorial_py_drawin_function>

[s.html](https://docs.opencv.org/3.4/dc/da5/tutorial_py_drawing_functions.html)

[10] *OpenCV: Performance Measurement and Improvement techniques.* (n.d.)<https://docs.opencv.org/3.4/dc/d71/tutorial_py_optimization.html>

[11] *OpenCV Python – How to draw curves using Mouse*

*Events?* (n.d.)

[https://www.tutorialspoint.com/opencv-python-how-to-draw-cur](https://www.tutorialspoint.com/opencv-python-how-to-draw-curves-using-mouse-events)

[ves-using-mouse-events](https://www.tutorialspoint.com/opencv-python-how-to-draw-curves-using-mouse-events)

[12] *Planning and control of mobile robots in image space from overhead cameras* | IEEE conference publication | IEEE Xplore. (n.d.). <https://ieeexplore.ieee.org/document/1570437>

[13] *Real time object recognition using OpenCV and NumPy in*

*Python. (*2023, March 14). IEEE Conference Publication | IEEE

Xplore.<https://ieeexplore.ieee.org/document/10099584>

[14] Samal, P. (2022, October 30). Curve fitting in Python: A

Complete Guide - AskPython. *AskPython*.

[https://www.askpython.com/python/examples/curve-fitting-in-py](https://www.askpython.com/python/examples/curve-fitting-in-python)

[thon](https://www.askpython.com/python/examples/curve-fitting-in-python)

[15] Team, D., Team, D., & Team, D. (2021, March 14). *Project in Python – Colour Detection using Pandas & OpenCV*. DataFlair. <https://data-flair.training/blogs/project-in-python-colour-detection/>

[16] Tzafestas, S. G. (2013). *Introduction to mobile robot control* (1st ed.). Elsevier Science.

<https://ebookcentral-proquest-com.ada.idm.oclc.org/lib/adaaz/detail.action?docID=1457948>

[17] Yacine, R. (n.d.). *Color-Based Object Detection with OpenCV and Python* | *Don’t Repeat Yourself.*<https://dontrepeatyourself.org/post/color-based-object-detection-with-opencv-and-python/>