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

List of Figures

List of Tables

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, within a predefined velocity profile. Developed control system demonstrates stability and robustness in terms 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

With progress in computation and the development of sensors, vision is increasingly being used as a sensor and feedback mechanism for autonomous robotic systems. This coordination supports robotic systems to tackle more challenging tasks. These days, people handle them to guarantee maximum effectiveness and trustwortheness. The category encompasses these robotic systems that provide lane departure alerts and parallel parking. Along with a predetermined path, mobile robots can move from one location to another with ease. It indicates that these types of robots travel from the starting point to the end point . They supervise without any human assistance. Unlike several industrial robots that are restricted to specific locations, mobile robots are able to move around with predefined areas for accomplishing certain objectives.

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. Its 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 the predefined path.

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. When looking at the frames or images captured from a web camera, image segmentation divides them into separate parts so that the robot can recognize the curious components. The tool we used (Python's library) basically defines the contours separately after detecting both two colors through the camera. After that, we manually draw the path so that the robot moves in the direction of that path. To obtain the desired errors, we establish a control algorithm that includes formulas that calculate the velocities of both left and right wheels of the robot and control them. In addition, we create a connection between Bluetooth pairing with our device to transmit signals to the motors.

In the business world, experiences we got when writing code belong to a set of rules and procedures. These are to make sure that the codes we write work correctly and they are clean. We consider security-related parameters to preserve our program. Using security features, we can ensure that our program is protected from various types of threats. The studies we conducted assisted us find articles that enhanced our design concept. We may acquire the consequences in testing and work with prototypes by upgrading our design concept. We analyze mobile robotic systems to assess their benefits. During the analysis process, the entire process is performed in detail, including all costs, development time, technical support and performance enhancement costs. People independently use sensors and feedback mechanisms in mobile robotic systems. Because sensors enact their work effectively and calculations faster. Furthermore, these types of robots are mainly divided into two groups, named structured and unstructured ones. The main advantages of the wheeled typed mobile robot used in our project incorporate an uncomplicated mechanical structure and lessens the low level of energy. Differential driven control system allows our robot to move freely and execute the given instructions at a high level.

1.2 The Objectives

The aim of the project is to recognize objects via an overhead camera in the room. By implementing image processing it is possible to move a predefined path, thus, concocting a robotic system. By utilizing OpenCV Computer Vision Library, it is possible to create a basis for the 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.

Enhances the accuracy of sketching curves on acquired photos with Accurate Curve sketching. A suggested image guarantees the precision and seamlessness of the robot's tracked path. interplay between several items. Through this interaction, the system is able to identify and depict multiple things at once,

giving each detected object a unique function or path robot motion synchronization. Drawn curves can be translated into transport commands to build a system that enables a mobile robot to follow a specified path. Error control and dependability: putting error recovery, reliability, and detection into practice. commissioning of equipment that lowers mistakes in movement coordination and object recognition, ensuring system dependability.

1.3 The Significance

The significance of the project is that by reducing the time delay the system can be optimized. The interaction between the user and the robot and the reaction between images from the camera have improved. By creating a comprehensive system that seamlessly integrates real-time object recognition, developing a user interface and robotic control, one can conclude the importance of the project. Using our customized approach, the OpenCV and Python libraries introduce the use of color identification, methods, and k-means clustering for accurate object recognition. A graphical user interface (GUI) was developed using Tkinter, allowing the integration of mouse event functionality for user interaction. Due to algorithms that smooth and interpolate user-drawn curves using spline interpolation, drawing accuracy will be increased. The significance is also through complex tracking algorithms for simultaneous processing of multiple objects in our system. By providing an intuitive and attractive interface, user-centered GUI design will function based on feedback. A communication protocol that converts graphical input into precise robot control commands will be needed for coordination between the graphical user interface and robot movement. Based on the principles of fault tolerance, reliable error detection and recovery mechanisms will be implemented. By focusing on reducing latency through parallel improvements in algorithm processing, optimization will be created. The main language will be Python, the graphical user interface (GUI) will be developed using Tkinter, computer vision will be provided by OpenCV, and control will be carried out using a microcontroller platform such as Raspberry Pi. User feedback, and continuous testing to continually improve the efficiency and usability of the system, are included in the interactive development process in our method. The project is subject to the necessary pedaling as its promotion will contribute to the growth of the transport systems sector and provide solutions to complex mobility and transport problems.

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.By providing a visual representation of the physical configuration of a system, detailed manufacturing drawings promote repeatability. Taken together, these findings demonstrate how computer vision, user interface design, and robotics can be successfully integrated to create a complete interactive object detection and control system.

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. Nowadays everything is widely used, parallel lane departure warnings, moving from one place to another are called mobile robots. everything happens without human intervention. Compared to industrial robots, which only move within the range of a specific work location, mobile robots can freely move around a predetermined work location to complete a goal.

In both structural and non-structural environments, mobile robots can be used for larger applications due to their advantages. We set the use of wheeled mobile robots, or WMRs, which are in great demand, as a goal in our project. Their mechanical simplicity and energy consumption make them suitable for general use. 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. We find the central coordinates of both colored circles, determine the direction of the robot’s movement along the predefined path with sending signals that control the robot's speed. Then, Bluetooth enables wireless communication between the motor and the robot's control system, called an "embot". By breaking down an image into individual elements, the robot can recognize areas of interest such as colors, etc.

OpenCV is a well-known open source toolkit used for various tasks such as color and edge recognition. The control algorithm develops the desired path and configures the controller to follow it exactly; Bluetooth provides wireless communication between the control system and the robot for remote control. Industry standards ensure code is readable, maintainable and scalable, and safety features such as emergency stop and collision avoidance are implemented to prevent collisions and reduce risk. The design process is based on scientific papers, textbooks, and online resources, and experiments and prototypes are used to test concepts. Prototypes are used to test concepts and iteratively improve designs. From an economic analysis perspective, the effectiveness of mobile robotic systems is assessed by taking into account hardware cost, development time, and potential benefits such as increased productivity and maintenance. Mobile robots have sensors and evaluation systems as long as the high performance of calculations and the benefits of sensors.

2. Literature Review

Color detection, which involves image segmentation starting with object detection using a webcam, is commonly used in many computer vision applications. Bansal (2023) offers a beginner's guide to color recognition using Python, explaining methods for recognizing specific colors from given images. Python uses libraries such as OpenCV and allows developers to implement the necessary color detection algorithms with the greatest effect. Color recognition is becoming central to a variety of computer vision tasks, including robotics, surveillance, and augmented reality.

Curve fitting is also an important factor in data analysis and computer vision, especially in path planning and object recognition. Brownlee (2021) provides insight into various curve fitting techniques using Python and their implementation using the NumPy and SciPy libraries. Curve fitting is a fundamental factor in modeling and interpreting data that comes from sensors, cameras, and other sources. Curve fitting is becoming a core component of many computer vision applications. Recognizing not just one, but several colors in real time is a fairly complex problem that is often encountered in robotics, image processing and augmented reality.

GeeksforGeeks (2023) provides information on multi-color detection using Python and OpenCV. This resource provides specific examples and code snippets that are aimed at helping you find the right color detection algorithms and handle real-life scenarios. Detection of the lower and upper color ranges becomes crucial for accurate color determination. The development of effective color range recognition is discussed by the OpenCV community in special OpenCV Q&A forums (undated). Thanks to this resource, an idea is formed about a number of factors that influence the determination of the color range. He also presents real-life tips for optimizing color detection algorithms [4].

Thresholding is a fundamental, important method that is used in image processing. It is also used in computer vision to segment images based on pixel intensity. OpenCV has extensive documentation on image thresholding (undated), ensuring developers understand thresholding techniques. It also offers potential applications in color detection, object segmentation, and image enhancement [5].

Note that the development of an application related to object detection using OpenCV and Python is often addressed in various studies in the field of computer vision. A wide variety of capabilities and methods for object recognition and detection are being developed, cascade classifiers are being studied (OpenCV, date not specified), etc. We emphasize that IEEE conference publications[6] demonstrate successful developments in the field of object detection algorithms, the practical value of which lies in their use in robotics and various autonomous systems.

If we look specifically at the OpenCV cascade classifier [7], then this is the basic element in the object detection process. It is quite often used in computer vision applications. A cascade classifier operates on the basis of several cascaded stages that help recognize objects within certain images or video streams. Cascade classifiers are a very complex device. Therefore, awareness of this complexity plays an important role in the development of a high-quality object detection system in a particular area.

It is very important that the OpenCV library has very extensive resources. Using a variety of OpenCV[8] documentation, researchers are developing

OpenCV in a variety of areas, including machine learning. This is a very important resource in the field of computer vision, which provides assistance to both young professionals and experienced researchers.

Drawing,image functions in OpenCV[9] play an important role in,visualizing the acquired data. This fact makes it possible to process the received frames in a computer vision application. Examples of the drawing function can be found in the OpenCV documentation. This allows researchers to contribute their own visualizations, annotations, and overlays. We also emphasize that performance optimization [10] should be implemented in order to improve the efficiency of the computer and increase the speed of work in the field of vision algorithms. OpenCV introduces various techniques to improve application performance. These techniques are designed to improve the efficiency of image processing and computer vision. Also, these methods will help improve real-time processing algorithms, improve the capabilities of resources in applications with high throughput.

It should be noted that OpenCV offers capabilities for handling mouse events. Because mouse events are gaining importance in interactive applications and user interface design. This includes, naturally, drawing and annotation tools. Using OpenCV, you can develop interactive applications where using the mouse [11] opens up the possibility of using images and graphic elements. This resource explains the capabilities of drawing curves using mouse events in OpenCV and Python. This fact helps in the development of interactive drawing and annotations.

As is known, mobile robots, their various functions, robot control, and the use of images from the top of the camera are a completely new area of development in robotics [12]. Conventional sensory methods are outdated. Adaptability and flexibility in various industries is guaranteed by innovation - the use of overhead cameras for navigation. The use of OpenCV and NumPy in Python[13] for object recognition at a specific time is a breakthrough in the field of computer technology and computer vision.

Thanks to these resources, developers will be able to clearly process visual data that will enable timely detection of the necessary objects in relevant robotics applications. For researchers and developers who analyze and model basic data, curve fitting in Python provides the essential foundation and guidance. It is this Python base, which includes NumPy and SciPy, that allows developers to implement complex curve fitting algorithms. Such an implementation creates conditions for recording conclusions from experimental data [14]. Research on color detection using Pandas and OpenCV in Python [15] provides the opportunity for the practical use of computer vision methods for color analysis and identification. This development demonstrates the implementation of digital solutions based on Python in solving specific problems in the field of computer vision.

Mobile robots and their control are an interdisciplinary field that includes robotics, control theories, and artificial intelligence.

An introduction to robot control is Tsafestas (2013). It includes the following topics: motion planning, trajectory generation, and feedback control algorithms. This resource presents both theoretical and practical aspects and is an excellent reference for learning how to control mobile robots. Three stages: webcam object detection, recognition and localization are included in image segmentation, which is included in the development of color detection. Yasin (n.d.) demonstrates certain techniques for identifying two objects with colored circles via a webcam in a programming language. This programming language is based on computer vision. Also presents several fragments of our program for the purpose of identifying and implementing color-based object detection algorithms.

With color information, researchers can reliably detect and track objects in videos and images. This fact creates conditions for the use of various computer vision applications.

PM Receiving frames or images from the top of the camera of mobile robots and controlling them is an emerging area of research in the field of robotics[12]. The use of overhead cameras for navigation represents a departure from conventional sensor methods, offering potential advantages in adaptability and flexibility in different environments. Real-time object recognition using OpenCV and NumPy in Python[13] represents a significant advancement in the field of computer technology. vision capabilities. Using the computing power of these libraries, researchers can achieve efficient processing of visual data, enabling timely and accurate object recognition needed for robotics applications. Curve fitting in Python, Samal explained, offers comprehensive guidance for practitioners looking to analyze and model data trends. Using 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 Color Recognition Project Using Pandas and OpenCV in Python[15] provides hands-on experience in applying computer vision techniques to identify and analyze colors in Python-based digital solutions while solving real-world problems in computer vision images. This hands-on application demonstrates the versatility of Python-based solutions for solving real-world computer vision problems. Mobile robot control is an interdisciplinary field covering aspects of robotics, control theory, and artificial intelligence. Tsafestas (2013) provides an introduction to the control of mobile robots, covering topics such as motion planning, path generation, and feedback control algorithms. This resource serves as a comprehensive reference for understanding the principles and techniques of controlling mobile robots, offering insight into both theoretical foundations and practical implementations. Color detection involves image segmentation, starting with webcam-based object detection, recognition, and localization. Yasin (n.d.) discusses methods for identifying two objects with colored circles via a webcam in a computer vision-based programming language, providing practical examples as well as several snippets of our program for implementing color-based object detection algorithms. Using color information, developers can reliably detect and track objects in images and videos, enabling a variety of computer vision applications.

Color detection, which involves image segmentation starting with object detection using a webcam, is commonly used in many computer vision applications. Bansal (2023) provides a beginner's guide to color recognition using Python, offering insight into techniques and methodologies for identifying specific colors in images. Using libraries such as OpenCV, Python allows developers to efficiently implement robust color detection algorithms. Understanding color recognition is critical to a variety of computer vision tasks, including robotics, surveillance, and augmented reality. Curve fitting is another important aspect of data analysis and computer vision, especially in path planning and object recognition. Brownlee (2021) provides a detailed tutorial on curve fitting with Python, covering various curve fitting methods and their implementation using libraries such as NumPy and SciPy. Curve fitting plays a vital role in modeling. and interpreting data from sensors, cameras, and other sources, making it a critical component of many computer vision applications. Detecting multiple colors in real time is a challenging problem often encountered in applications such as robotics, image processing, and augmented reality. GeeksforGeeks (2023) provides information on simultaneous detection of multiple colors using Python and OpenCV. This resource offers practical examples and code snippets to help developers implement robust color detection algorithms that can handle real

3. 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 though 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 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 according to 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 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 determine 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 the splprep function to compute the B-spline representation of the curve. In this case, a list ([xd, yd]) we created containing the positions of the waypoints drawn through 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 seamlessly. Both locations of the interpolated points can be stored in 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 avoids 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 the 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 a constant and seamless path for the robot to travel. Picking the desired path and angular path for directing the 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

This mobile robotic system has been carefully designed according to the principles of engineering standards in the fields of robotics, software and image processing. The system is scalable, maintainable, easy to understand guarantee strict compliance and are durable. In addition, they can flexibly adapt to change requirements. The systems are equipped with advanced features such as emergency stop and collision avoidance mechanisms. Safety is a priority and potential risks can be reduced. Potential risks can be reduced by collision avoidance and such precautions are considered necessary to ensure the safety of the robot and its environment. In addition, serial communication complies with technical standards. Communication protocols between system elements are designed for efficient and reliable data exchange necessary for effective and reliable data exchange. Our program complies with the Bluetooth standard, protocols are implemented for serial and wireless communication. Through a channel, data is sent bit by bit during serial communication. To communicate seriously, we employed a 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 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 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

The concept of design entails the study of activities performed, analysis by experienced individuals and the development of prototypes corresponding to them. 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 an overhead camera and documenting the robot's motion data.

System functions such as calculating distances, corners and midpoints, etc., related to detecting colors and obtaining their coordinates. Obtaining coordinates and related functions are very important for this procedure. These features are used to ensure that every frame captured by the camera is carefully checked. The first function of the indicator is to provide useful information about the orientation and position of the robot. Providing useful positioning information, it also helps determine the robot's position and other important distance data. These mechanisms are the robot's navigation capabilities. They serve as the basis for assessing the robot's navigation capabilities. By comparing trajectories, calculating errors and visualizing, at this stage the accuracy and efficiency of the robot is assessed. Using visualization at this stage, the accuracy and efficiency of the navigation system is assessed.

The navigation system is performed by the function of the control algorithm. They are performed by the control algorithm function. Moving the robot forward 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, and 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.

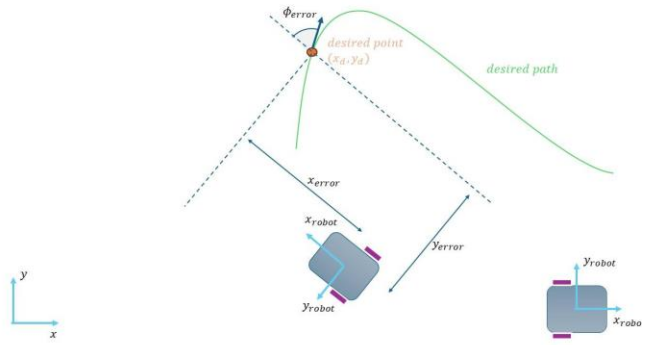


Figure 1 Control of Differential Drive Mobile Robot

Figure 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):

$$\dot{x} = v \cdot \cos(\phi)$$
$$\dot{y} = v \cdot \sin(\phi) \qquad \text{(Equation 1)}$$

A kinetic model of the system is delivered by another formula depicted in the following equation (Equation 2):

$$\dot{\phi} = \omega \qquad \text{(Equation 2)}$$

The kinematic non-holonomic constraint, $\dot{x} \cdot \sin(\phi) = \dot{y} \cdot \cos(\phi)$ holds and indicates that v as a vector $(\dot{x}, \dot{y})$ 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

$$v(t) = K_x \tilde{x}_r + v_d \cos(\tilde{\phi}_r)$$
$$\omega(t) = K_\phi \sin(\tilde{\phi}_r) + K_y v_d \tilde{y}_r + \omega_d$$

over time (the error approaches zero rapidly):

$$\tilde{\phi}_r = \tilde{\phi}$$
$$\tilde{x} = x_d - x$$
$$\tilde{y} = y_d - y$$
$$\tilde{\phi} = \phi_d - \phi$$

If we make it clear in writing:

$$v(t) = K_x\big(\cos(\phi)\,(x_d - x) + \sin(\phi)\,(y_d - y)\big) + v_d \cos(\phi_d - \phi)$$
$$\omega(t) = K_\phi \sin(\phi_d - \phi) + K_y v_d\big(-\sin(\phi)\,(x_d - x) + \cos(\phi)\,(y_d - y)\big) + \omega_d$$

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 assessment. The weight of each development phase is calculated as extra work requirements. as are the related labor expenses. When calculating the duration of the phase of the next work, the weight of this phase is considered. 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

First, the mobile robot system interprets the data obtained through live video capturing in a real environment. The created system then procedures the analyzed data by identifying predetermined trajectories and the obstacles they encounter. The most vital part of this process is closely related to visual signs, which are used to drive safe routes and avoid risky situations. Besides, the step of independently operating robots or vehicles are considered a part of this process. The next step is that data estimated in the real environment are switched into the instructions given to the robot, which it executes. This transformation is performed by calculating the velocities of both left and right wheels of the robot and control algorithms assistance. The control algorithm that updates the robot's movement based on the information available within the real-world operating environment creates a precise navigation system.

Thus, the mobile robot can dynamically conform its path due to the modifications it made in the environment. Transportation systems, including the transportation of vehicles in certain areas and the service provided, have significant socio-economic impacts. Since these systems have been spreading more quickly, the work available to human drivers and the role of navigation systems will be reduced. These modifications may affect various job departments. At the same time, occurring changes may have had an impact on the corrections in the intended requirements. Retraining employees to use new technologies that improve and the capabilities of the system. Employees can be retrained to take advantage of new job systems and advanced robotics that are expected to be introduced and can be scanned. By managing multiple systems, connections can be established for more technical and monitoring tasks.The robot system has the ability to handle the analyzed data more productively in the real work environment. Initially, by computing the distance between two data points of the trajectory, this system discovers predetermined paths. In a real-world environment, they adapt to the strategic decisions of people of several functions, transitioning to automated decision-making systems. After that, this became another essential point to emphasize how the three different values we created within the control algorithm, named Kx, Ky, and K_phi alter the x and y coordinates of the mobile robot. In fact, these values measure whether the robot is moving faster along a predefined path and if it encounters any corner, this turns at an angle, and moves along the x-coordinate.

Therefore, they can demonstrate higher reliability and efficiency in less complex environments.

Maintaining safety and efficiency are important factors. In autonomous systems, the live image processing has high level of priority. 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.

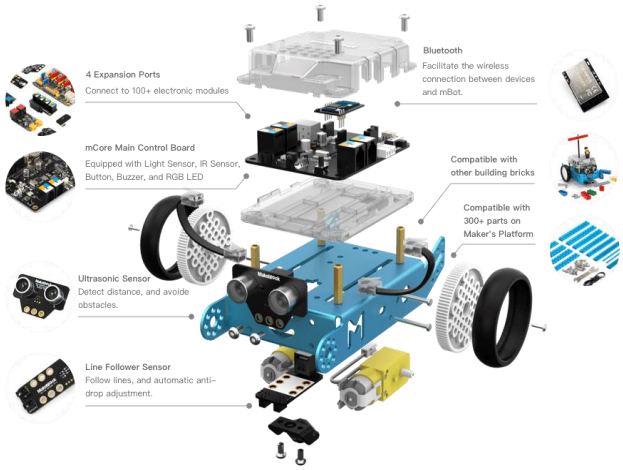


Figure 2. The components of the mBot mobile robot

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 are 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.

MakeBlock mBot is based on Arduino-compatible microcontroller boards, in particular MakeBlock Orion boards (abbreviated as mCore) based on the ATmega328P microcontroller. The board is equipped with H-bridge motor driver control (TB6612FNG), which allows you to independently control the speed and direction of two DC motors. The board integrates electronics such as an ultrasonic detection sensor, an infrared receiver for remote control, and lamp tracking sensors for autonomous navigation. In addition, the system includes a microcontroller board and a Bluetooth module (HC-06) for wireless communication with devices such as smartphones and tablets. For our project, we utilized mBot's 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, includes dual H-bridge motor driver (TB6612FNG),
Includes ultrasonic sensor, infrared receiver and line tracker sensor. The sensors are connected to the microcontroller board through the appropriate connectors, sockets and jumpers. They are connected to the microcontroller board through the appropriate connectors, sockets and jumpers. A Bluetooth module (HC-06) is also connected to the microcontroller board.
For wireless communication with external devices. Communication is guaranteed. There is also a mounting point on the microcontroller board. There is also an additional camera module on the microcontroller board. There are also mounting points for additional accessories such as modules.

MakeBlock mBot can be equipped with the following additional devices:
- Additional package
-MakeBlock mBot
- interactive light and sound.
-Includes a 480p camera module.
-The camera module contains a CMOS image sensor with a resolution of 640x480 pixels.
-Resolution 640x480 pixels, frame rate 30 Frames/sec. It is mounted on a swivel bracket and has a movement range of approximately 180° horizontally and 90° vertically, allowing you to adjust the viewing angle and point of view. The camera module communicates with the microcontroller board through a dedicated serial interface, providing real-time video streaming

and image capture capabilities. In our project, this camera is not used; instead, the upper camera monitors the robot's movements. The software development part describes the communication between the external camera and the robot.

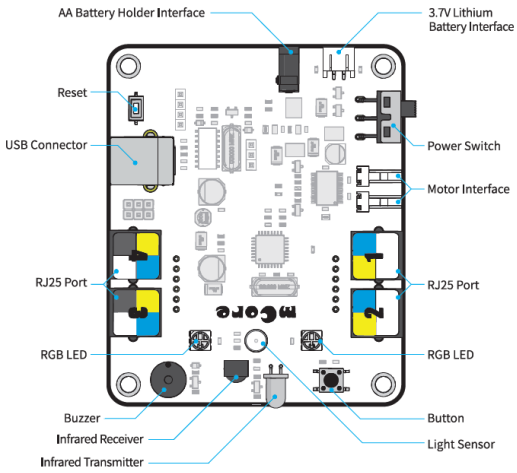


Figure 3. The mBot's software design

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 are 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 program records the movement of the mouse cursor.coordinates to determine the flight path. The trajectory is saved as a series of points and later interpolated to produce a smooth curve. This feature allows the user to set individual routes for the robot to follow. The software communicates with the Bluetooth module to send control commands to the robot. Opens a serial connection to a user-specified Bluetooth port (). Control commands, consisting of left and right wheel speed data, are transmitted to the robot via Bluetooth.

This sequential connection process allows the robot's movement to be controlled in real time based on the estimated speed. In general, the software integrates control algorithms, image processing techniques, user interaction and communication protocols to facilitate the robot's autonomous navigation along given path. It provides a flexible and interactive platform for controlling and directing the robot's movement in real-world environments.

The software includes a robust exception handling mechanism to handle unexpected errors or interruptions. For example, a program can be terminated without problems using keyboard interrupts (Ctrl+C), which ensures that the procedure terminates correctly and resources are freed. The trial block encapsulates the main logic of the program, which continuously processes camera footage and controls the robot's movement. The exception block specifically handles the KeyboardInterrupt exception that occurs when the user interrupts the program in the terminal. When this exception occurs, the program displays the message "The program has been interrupted!" before leaving. This ensures that even if the user interrupts the program, it will still release 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 is a complex process of determining complex trajectories and Allows you to make adjustments in real time while you work. The overall design of the software ensures that the software is modular, flexibility and ease of use ensure efficient operation. Image processing and control algorithms integrate user experience and make accurate and reliable navigation.

4.3 Main components of the project

Image acquisition section

Capture video frames from a camera using the OpenCV library. In this section, the idea is enabled using the `cv2.VideoCapture(2)` command, where `2` represents the camera index. Captured footage. The captured frames are used as visual input for image processing, and control and introduction of visual input for control. Without captured images the software will not work. Visual input is required for path planning and navigation. It is necessary for navigation. The robot cannot recognize its environment. Unable to move autonomously or interact with the environment. The robot cannot navigate or interact with its environment. Mouse Event Handling Allows the user to interact with the GUI. The user can interact with the graphical interface and determine the desired flight path of the robot. The robot's flight path can be determined. function draw_curve_with_mouse_events.

The draw_curve_with_mouse_events function responds to mouse events in the designated window (such as button application or movement movement). Wherein. records the scientists' trajectory and drags the user's mouse, Points to be reached at some interval to avoid difficulties. Ensure sufficient distance between points to avoid confusion. 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 a geometric position point. The function fits a spline curve to the locus points. It ensures smooth movement while maintaining the overall shape of the curve. The interpolated curve represents the desired path that the robot should follow, and represents the desired path that the robot should follow and provides a continuous path for navigation.

Without path definition and interpolation the software will not be

able to create a smooth and continuous trajectory cannot be created. 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.

Figure 4. The view of robot with colored circles

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.

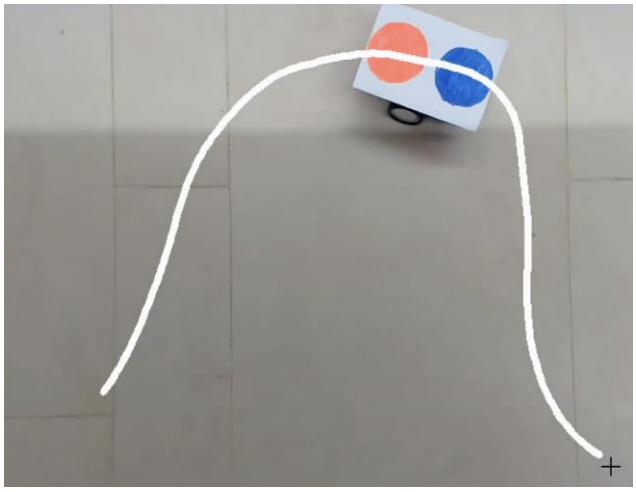


Figure 5. The robot moves along the predefined path

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 □ □□□ □,

$$v_R = v + a \cdot \omega, \quad v_L = v - a \cdot \omega.$$

Or we can find the inverse relation:

$$v = \frac{v_R + v_L}{2}, \quad \omega = \frac{v_R - v_L}{2a}.$$

Differential equations representing the system (Kinematic model):

$$\dot{x} = v \cdot \cos(\phi)$$
$$\dot{y} = v \cdot \sin(\phi)$$
$$\dot{\phi} = \omega$$

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 $\square_R$ and ω_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.

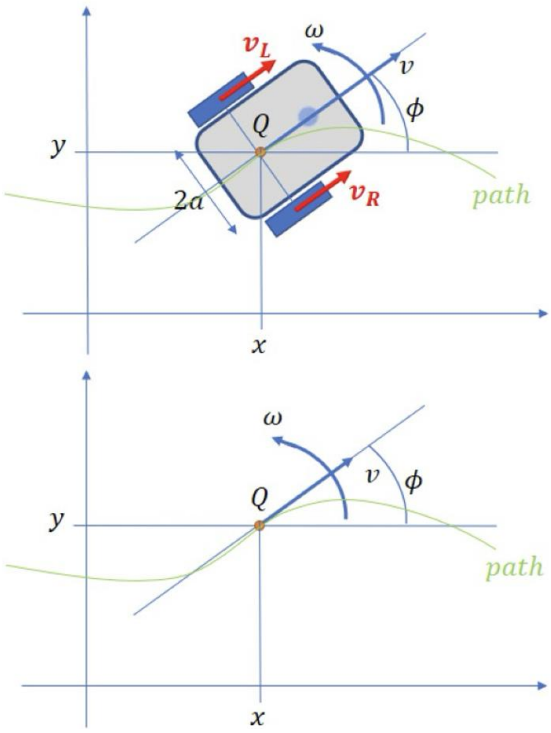


Figure 6. The robot kinematics

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.

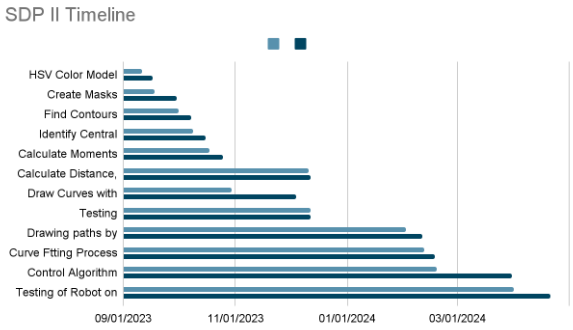
- Declared a variable named st and set it to 0 to keep data points separated from each other in the lines_arr collection.
- Modified the mouse events function to determine whether it's the first click of the left mouse button and add

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 Gantt Chart



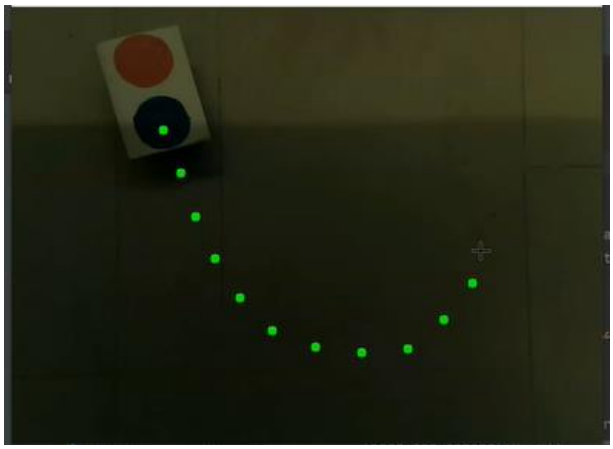
4.4 Testing/Verification/Validation

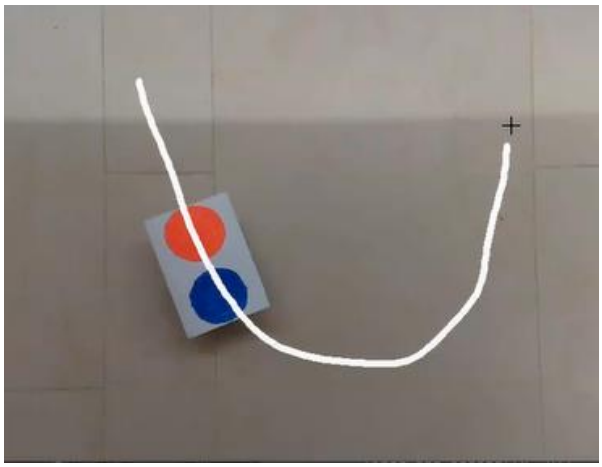
At the start of our project, even before we had access to the robot, our main task was to complete path generation. In the course of 4 weeks, we did many experiments in this part of our project. In the older iterations of the source code, we tried line computing, linear regression, Voronoi diagrams, 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, would the robot be able to complete the path if it was longer or had more acute turns. What to do with obstacles of different colors?
Checking results for compliance with project goals and requirements The observed indicators were compared with pre-established criteria, compared with the specified criteria. In order to make sure that the robot correctly followed the given route and overcame obstacles and responded adequately to external commands. Accuracy and reliability were also assessed. The accuracy and reliability of the following functions were also evaluated. Trajectory tracking, color detection and route tracking functions. These features were also rated. At the testing stage, an experimental configuration was created and data on the robot's position was collected. Data on the position of the robot was established, data on the position of the robot, speed and sensor readings were collected. A gimbal camera was used to videotape the robot's movements sensors to collect additional information about the environment. Additional environmental information was collected.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 indicate 2-2.5 cm path deviation, 340 msec execution time, and no errors.

During testing, problems arose with Bluetooth connection, sensor calibration, and algorithm settings. These issues were resolved by properly pairing and configuring the Bluetooth device, setting sensor calibration parameters, and fine-tuning the control algorithm based on observed performance and feedback.
The mBot had irregular Bluetooth protocols that made the testing period is 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

the 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. No longer did we need to set the ideal brightness for our robots to work correctly. The last serious
problem was inaccurate movements and deviations from the trajectory. Through repeated trial and error, we tried to find the optimal K_phi value for the program that would minimize deviations from the trajectory and give the best results. Overall, these efforts helped improve the robotic system and ensure it operated as required by the project.





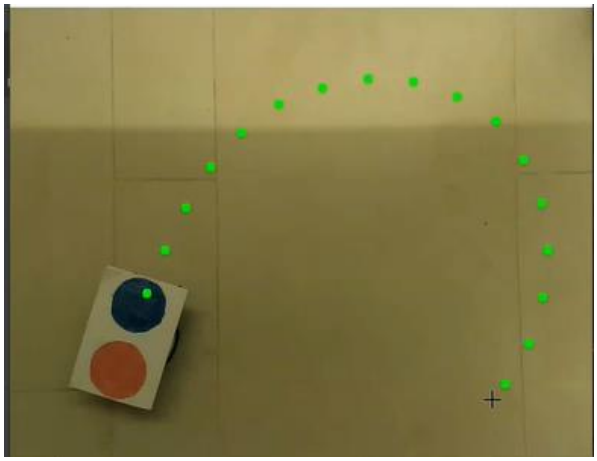


Figure 7.1 The robot receives the user input

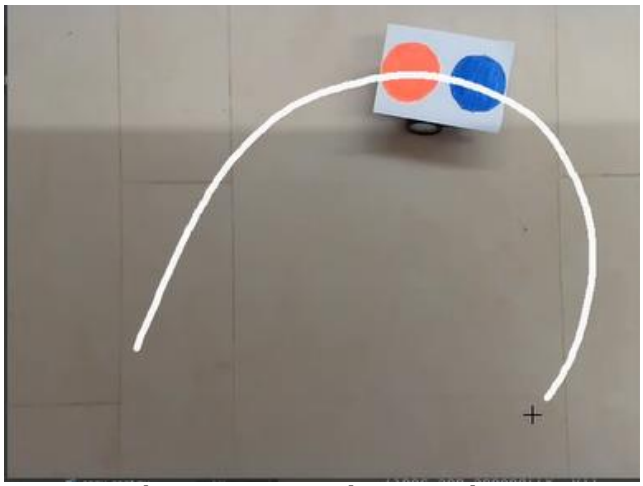


Figure 7.2 Then, it generates spline interpolation

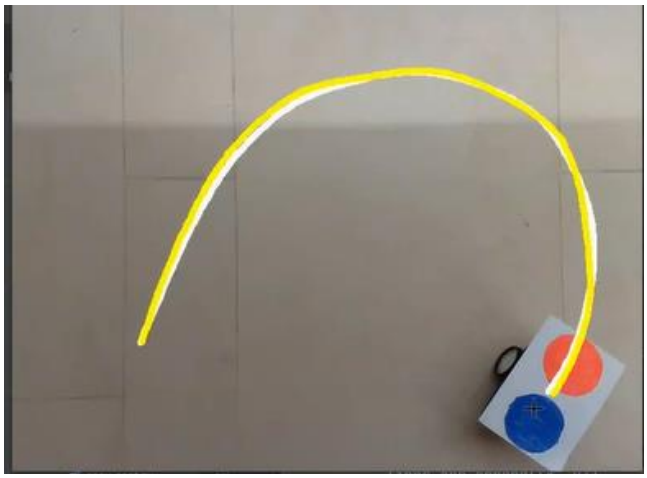
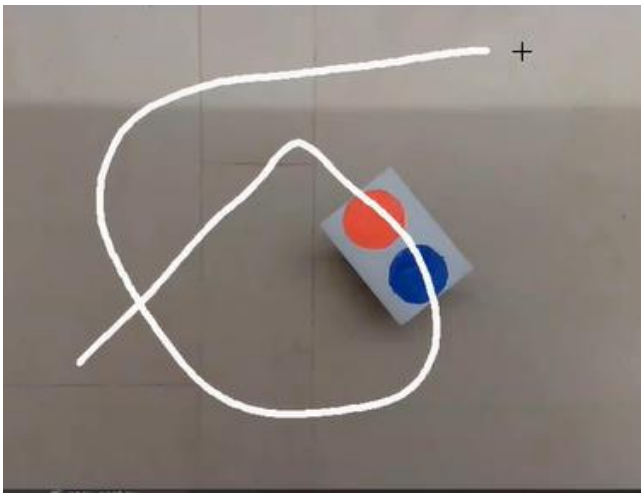
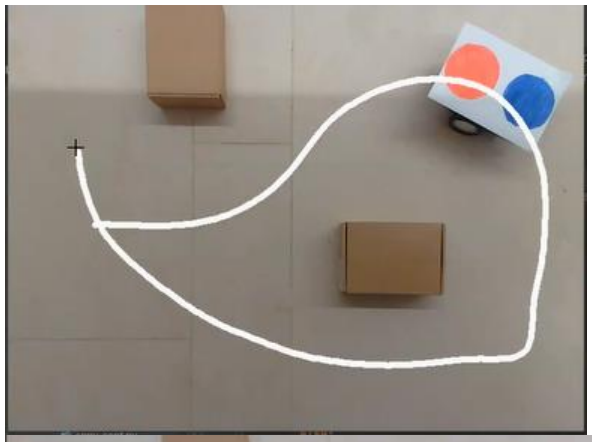
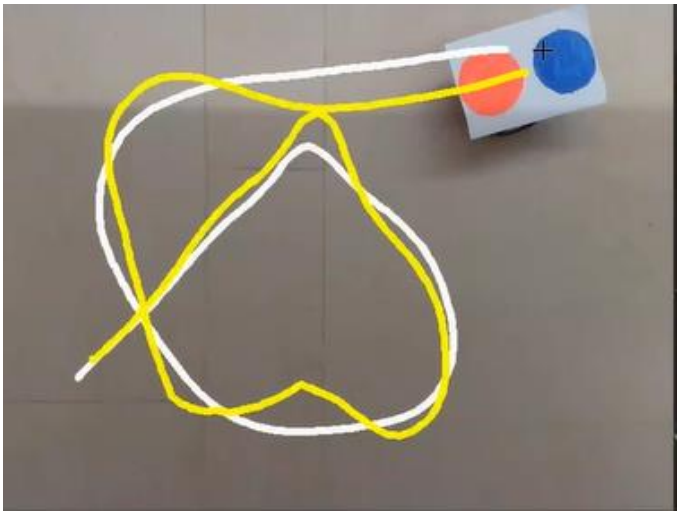
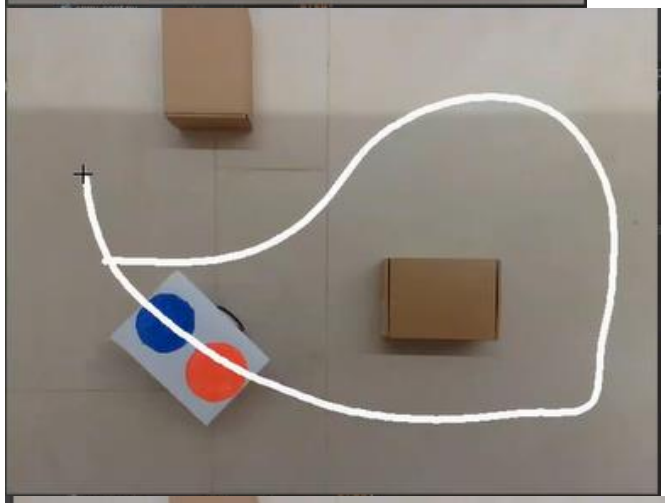


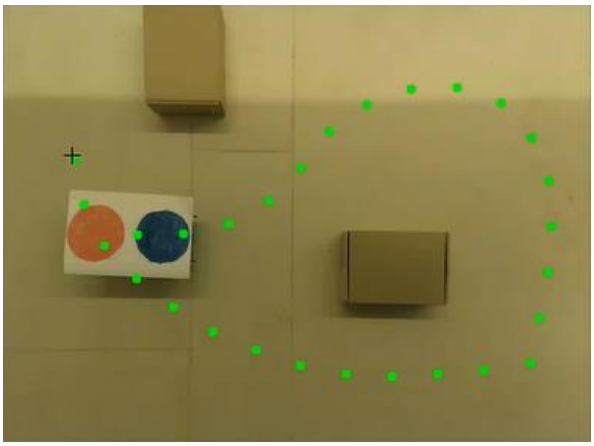
Figure 7.3 At the end, it moves along the desired path

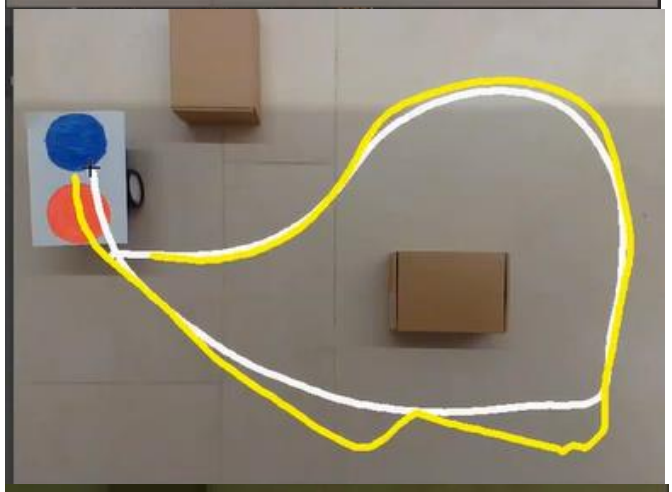


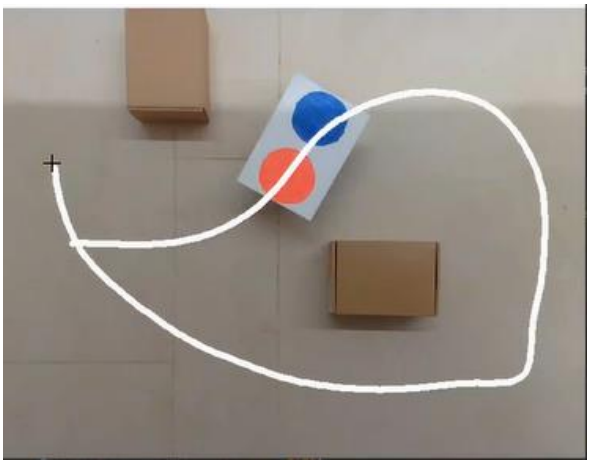


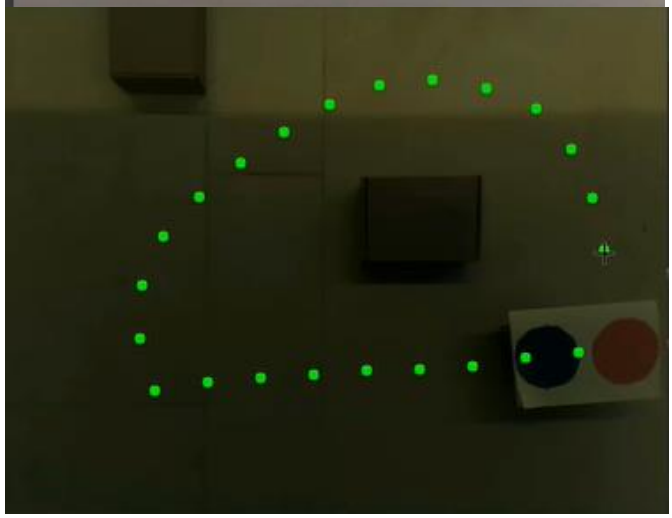


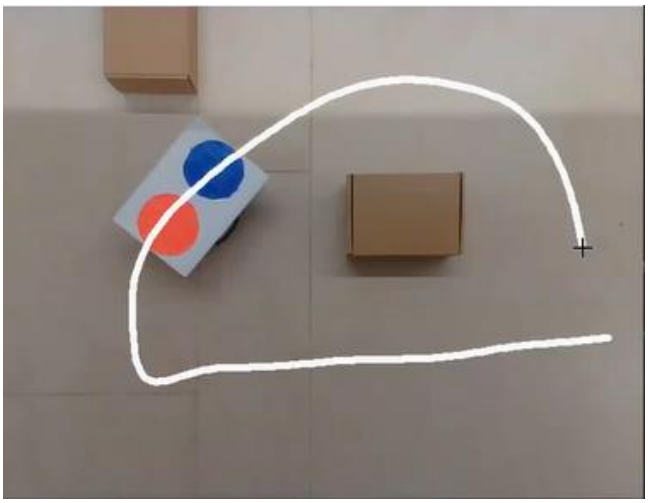












a set of paths through testing and implementation. Our comparisons measured the certainty of the robot's navigation system by assessing how it moves along paths with different obstacles placed on it. We had to refine the control algorithm and the properties for the robot to move correctly. This was essential to secure effective operation. The next trace in our project indicates that the mobile robot reaches the final destination by overcoming the obstacles encountered while moving along a predefined path. This allowed us to challenge new conditions to be added to make the robot work better and increase its safety. Obstacle tracking compromised modifications to clear trajectories. By examining various obstacle situations, we valuated the robot's obstacle avoidance methods and efficiency. The system must be focused on safe navigation. The requirements for the project delivers the rapid response of the robot through the long paths. Also, we inspected the reaction time of the robot in different activities. This is considered significant when the robot replies quickly. Besides, we checked both the Bluetooth connection and the the exactness of the data transfer for system setup and proper data management. To confirm security, a strong communication the robot and the operator conveys essential factor.

The system's performance was assessed based on its ability to meet project objectives, including navigation accuracy, obstacle avoidance, real-time responsiveness, and communication reliability. Thorough testing and analysis provided insights into its capabilities and areas for improvement. Throughout the project, we encountered successes and areas for improvement. One success was achieving precise navigation through control algorithm tuning.

5. Conclusion

At the beginning, our team noticed how the control algorithm the robot impacts necessarily through the navigation system and functionalities related to following the robot's path. By adjusting its bounds, we may reform the enactment of the robot. Moreover, color recognition and following path properties enable the robot to better understand its environment and shifting to some cases to conform. Color recognition guides the robot to navigate the environment in an effective way. Bluetooth connectivity allows the mobile robot to easily convey with other devices. In addition, it supports valid control, motion tracking and use in different platforms. In the lab room, we demonstrated numerous tests to check the robot's movement in various cases. They proved our robot and the system to be more powerful. In different conditions, our system meets the main objectives. İn essence, the implementation and testing phases confirmed the effectiveness of sensors, the control algorithm, and the navigation connectivity. Our findings support the development of robotic systems applications in the real world.

The system's enactment is investigated and met the qualifications to obtain principles for vital points. Initially, the main purpose is to ensure the establishment of the mobile robotic system, which requires precise navigation along predetermined paths. The system evaluates its ability to travel

Effective obstacle avoidance mechanisms were also implemented, enhancing the robot's safety and autonomy. Reliable communication via Bluetooth was established, enabling seamless interaction between the robot and external devices.
One of the main challenges encountered in the project was related to the system's responsiveness in real-time. Despite efforts to improve response times, occasional delays were observed in the robot's reactions to stimuli or commands, impacting its performance in dynamic environments. Additionally, there were limitations in camera coverage, leading to blind spots in obstacle detection. Developing and optimizing complex control algorithms also proved challenging, requiring significant time and resources. While the project achieved successes in navigation and communication, addressing issues with responsiveness, sensor coverage, and algorithm complexity is crucial for enhancing performance. Insights from the project suggest several areas for future research, including optimizing control algorithms, improving sensor capabilities, enhancing autonomous decision-making, exploring advanced learning techniques, improving human-robot interaction, and addressing ethical and societal considerations. Pursuing these areas could lead to significant advancements in robotics technology, creating more capable and socially responsible systems.

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-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_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-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-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/