ME 598-A

INTRODUCTION TO ROBOTICS

GROUP: R2

FALL 2024

AUTONOMOUS OBJECT SORTING USING A MOBILE ROBOT

ME 598-A
Introduction to Robotics
Fall 2024

Final Project: Autonomous Object Sorting Using a Mobile Robot

---

R2
17 Dec 2024

Undergraduate students:

"I pledge my honor that I have abided by the Stevens Honor System."

Graduate students:

"I pledge that I have abided by the Graduate Student Code of Academic Integrity."

The report has been prepared by:

1. Mohammad Althaf Syed *althaf syed*
2. Bhagyath Badduri *Bhagyath*
3. Matthew Casey *Matthew Casey*
4. Prayash Das Prayash Das

Contents

Figures:

Tables:

Abstract

This project investigates the development and deployment of a mobile, autonomous robot that can sort objects in a robotics playground simulation. The robot must recognize and move red, green, and blue objects to the appropriate target zones while following certain operational guidelines. The robot must function under six different initial conditions in the arena, a regulated 5-meter-square space with predetermined object locations and zones. The robot must stop and return to its initial location at the conclusion of the task.

The project makes use of the robot's preconfigured sensors, which include odometry, three distance sensors, and a forward-facing color-detecting camera, to accomplish these goals. Since neither the robot's hardware nor its sensor setup can be changed, the issue is one of developing an efficient algorithm. Color-based object detection, zone identification based on arena geometry, collision-free navigation, and accurate movement for object manipulation are among the main features.

Multiple trials will be conducted to evaluate the system's performance, with an emphasis on accuracy, repeatability, and resilience under various beginning conditions. The robot's trajectory charts, task completion time, and the quantity of objects correctly sorted are important metrics. Stateflow modeling and Simulink-based function blocks are two sophisticated approaches used in this project to guarantee the smooth integration of algorithm for autonomous operation.

This study offers important insights into creating workable robotic systems for contexts with constraints. With a focus on combining perception, control, and motion planning in autonomous systems, the findings advance our understanding of practical applications in automation, logistics, and robotics.

Introduction

Robotics has seen a lot of fast progress lately, especially with autonomous systems becoming really important in areas like manufacturing, logistics, and service industries. A key challenge in robotics is getting a robot to understand its surroundings, make smart choices, and carry out tasks with little help from people. This project, which is part of the ME 598 course, tackles these challenges by creating an autonomous mobile robot that can sort objects in a limited space.

The project takes place in a simulated setting called the Robotics Playground. In this environment, the robot has the job of recognizing objects in different colors—red, green, and blue—and bringing them to their matching target areas. The robot needs to work inside a square arena with set dimensions, using only its built-in sensors. These include a camera that can detect colors in front, three distance sensors, and the ability to measure its movement accurately. These constraints mimic real-world situations where robot hardware is usually restricted, highlighting how crucial it is to have efficient algorithms and strong control design.

The project needs to focus on creating algorithms that can recognize objects based on color, navigate without collisions, and perform specific motions for sorting objects. The robot also needs to be able to go back to where it started after finishing the sorting task on its own. The system's performance will be assessed by how well it adapts to six different starting conditions for object placement, which means the algorithms need to be strong and adaptable.

This project gives us a chance to dive into some advanced topics in robotics, like perception, localization, motion planning, and control. This project brings together different parts into a unified system, showing the real-life challenges and solutions in autonomous robotics. It sets the stage for applying these ideas in automation and smart systems.

Theory & Experimental Procedure:

The figure shows below the forward-facing object detector (akin to a color-detecting camera) and the three distance sensors integrated into the robot for the project. The configuration of these sensors, which includes their placement and range, is predetermined and locked. Users can access specific details of each sensor's setup by right-clicking the respective block in the simulation environment and selecting Mask → Look Under Mask.

Key points regarding the sensors:

1. ***Object Detector***: It is used to recognize and differentiate objects based on their color (red, green, or blue).
2. ***Distance Sensors****:* These help the robot detect its proximity to obstacles, such as arena walls, enabling collision avoidance.
3. ***Odometry Capabilities****:* These are used to track the robot's position and orientation within the simulated environment.

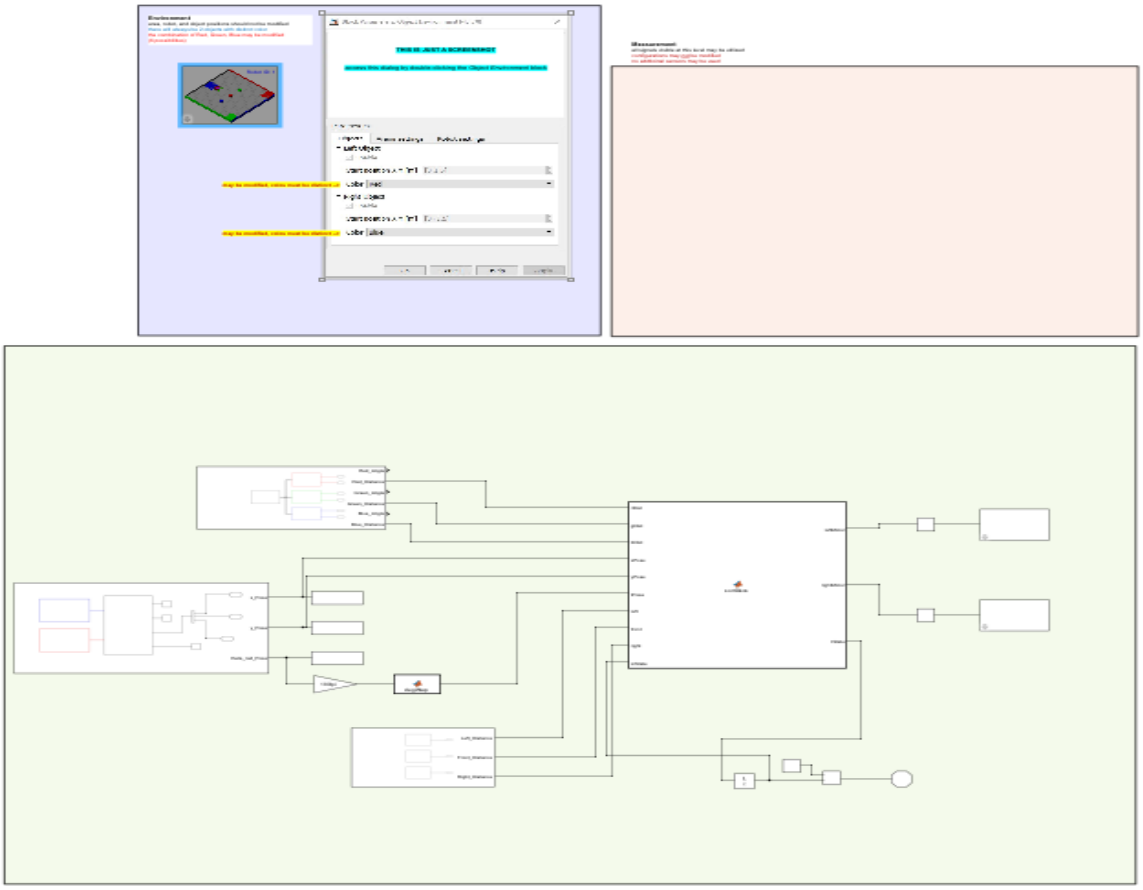

Figure 1 - Robot sensor Signals.

Detailed explanation of algorithm:

The robot uses input parameters such as distances to objects (Red_Distance, Green_Distance, Blue_Distance), environmental sensor data (Left_Distance, Front_Distance, Right_Distance), and its current pose (xPose, yPose, tPose180) to decide on movement and actions. The robot operates in a series of sequential states to achieve its task.

In State 0, the robot navigates to the top object's position at coordinates (0, 1.5) using the reposition to function, which calculates the angle to the target and aligns the robot for movement. Once at the location, in State 1, the robot identifies the object's color through the identify object Color function, which evaluates valid distance values to determine if the object is red, blue, or green. Based on the detected color, the color Sort function directs the robot to push the object into the corresponding zone: red objects are moved to (1.85, 2.05), blue to (-2, 0), and green to (2.1, -1.95). The robot aligns itself using the align to Target function, which rotates the robot if there is a significant angle difference and moves straight when aligned.

In State 2, the robot ensures a slight reverse movement after sorting the object using the reverse slightly function, which checks the proximity to the object and adjusts the motors accordingly. Transitioning to State 3, the robot moves to the bottom object's position at (0, -1.5). In State 4, it follows the same color detection and sorting process as in State 1. After completing the sorting, State 5 again uses the reverse function to ensure the robot moves away from the object. Finally, in State 6, the robot navigates back to the origin (0, 0) to complete its task.

The algorithm incorporates robust collision avoidance logic to ensure safe navigation. If the front sensor detects an obstacle closer than 0.3, the robot stops or reverses. Based on available space, it rotates left or right to avoid the obstruction. The system relies on angle calculations (compute target angle) and precise motor commands (rotate CW and rotate CCW) for efficient movement and alignment. Overall, the algorithm systematically controls the robot to identify and sort objects by color, reposition as needed, and avoid collisions while returning to its starting point.

Workflow

1. **Input Data Collection:**
   The sort objects block gathers the following key inputs:

   Color-Based Distances:

   - Red_Distance, Green_Distance, Blue_Distance – Indicate proximity to objects of specific colors.

   Proximity Sensors:

   - Left_Distance, Front_Distance, Right_Distance – Aid in detecting and avoiding obstacles.

   Robot Pose Data:

   - xPose, yPose – Represent the robot's current position within the arena.
   - tPose180 – Provides the robot's orientation in degrees.

   Initial State:

   - The current State defines the starting point of the Finite State Machine (FSM).
2. **Execution of the FSM (Finite State Machine)**
   - The sort objects block executes the following state-wise logic:

   State 0:

   - The robot moves to the top object position at coordinates (0, 1.5) using the reposition to function.

   State 1:

   - The robot identifies the color of the object using identify Object Color.
   - Based on the identified color, the robot moves the object to the corresponding zone using color Sort:
     - Red Zone: (1.85, 2.05)
     - Blue Zone: (-2, 0)
     - Green Zone: (2.1, -1.95)

   State 2:

   - After sorting the object, the robot performs a slight reverse movement using reverse slightly.

State 3:

- The robot navigates to the bottom object position at (0, -1.5).

State 4:

- The robot repeats the color detection and sorting process as in State 1.

State 5:

- The robot reverses slightly again after sorting the object.

State 6:

- The robot returns to the origin at (0, 0).

3. Motor Control Outputs

The sort objects block generates motor commands to control movement:

Left Motor and Right Motor:

- Commands for rotations (rotate CW, rotate CCW) to achieve alignment.
- Commands for forward or reverse movement during navigation and object handling.

4. Collision Avoidance Logic

The FSM ensures safe operation by incorporating collision avoidance based on proximity sensor inputs:

- If an obstacle is detected within 0.3 units in front, the robot stops or reverses.
- If a side sensor detects obstruction:
- The robot rotates in the opposite direction to avoid the obstacle.

5. Task Completion

In the final state, the robot ensures successful task termination:

- Motor Outputs: Both motors are set to zero to stop movement.
- The system confirms the task is complete by comparing the FSM's state with "7", triggering the stop condition.

Part 1

Performance Analysis Across Initial Conditions:

Objective:

The robot's performance was tested under six distinct initial configurations, varying the color and position of objects in the arena. The goal was to evaluate the system's ability to autonomously identify and sort objects into their respective zones.

Condition 1: Green and Blue.

Initial Configuration: Green and Blue Configuration.

- Object positions: Green object at [0.03,1.489], Blue object at [-0.03,−1.47].

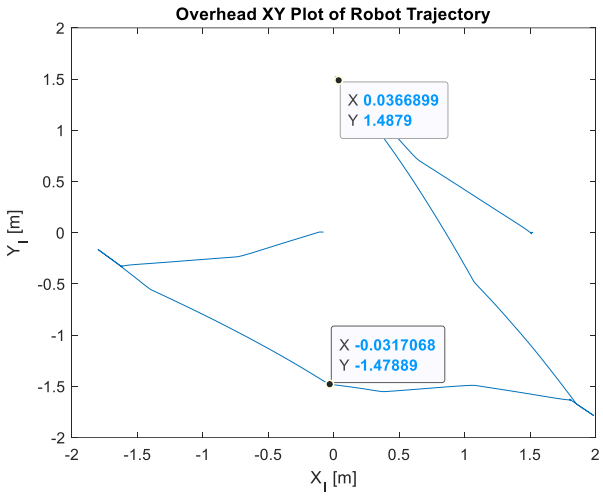

Figure 2- Green and Blue object locations.

Procedure:

- The robot started at the initial position [1.5,0].
- The green object are at [0.03,1.489], Blue object at [-0.03,−1.47] respectively.
- Next, The objects were placed at:
  - Green Object: [1.92,-1.73]
  - Blue Object: [-1.78, -0.17]
- Task: Is to detect and transport both objects to their respective zones, then return to the origin [0,0].

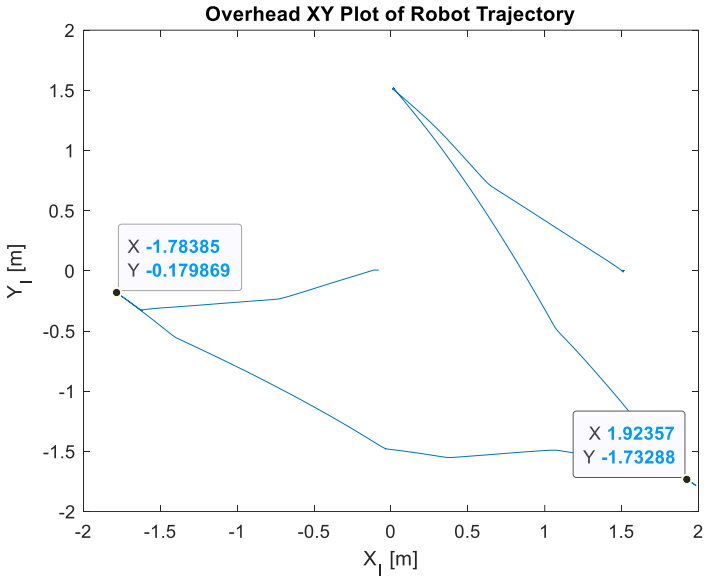

Figure 3 - Goal positions at their respective green and blue zones.

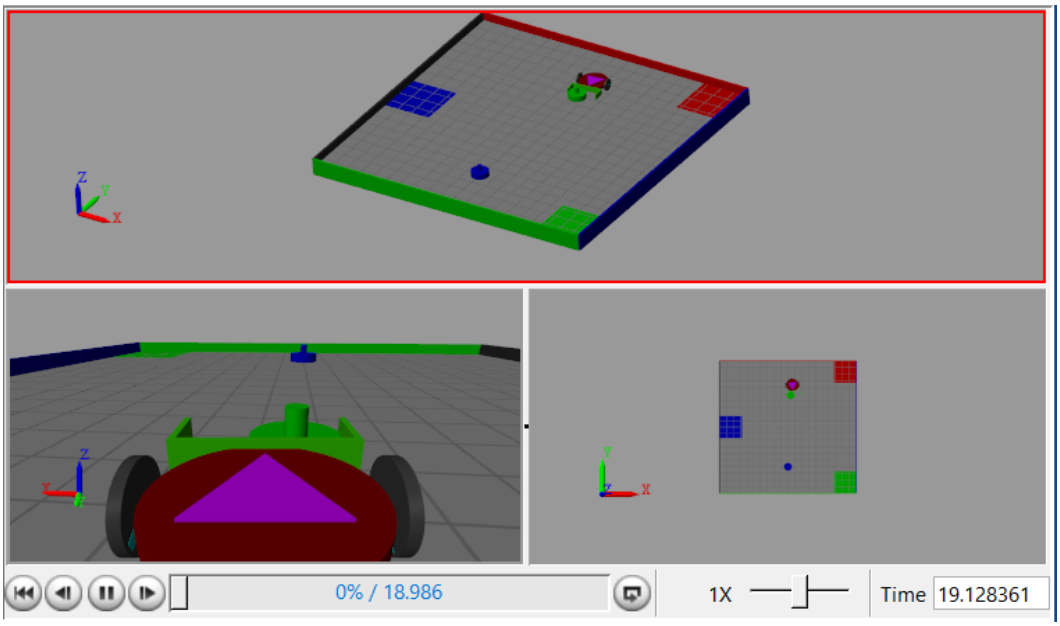

Figure 4 - Mobile robot placing the green object to its respective green zone.

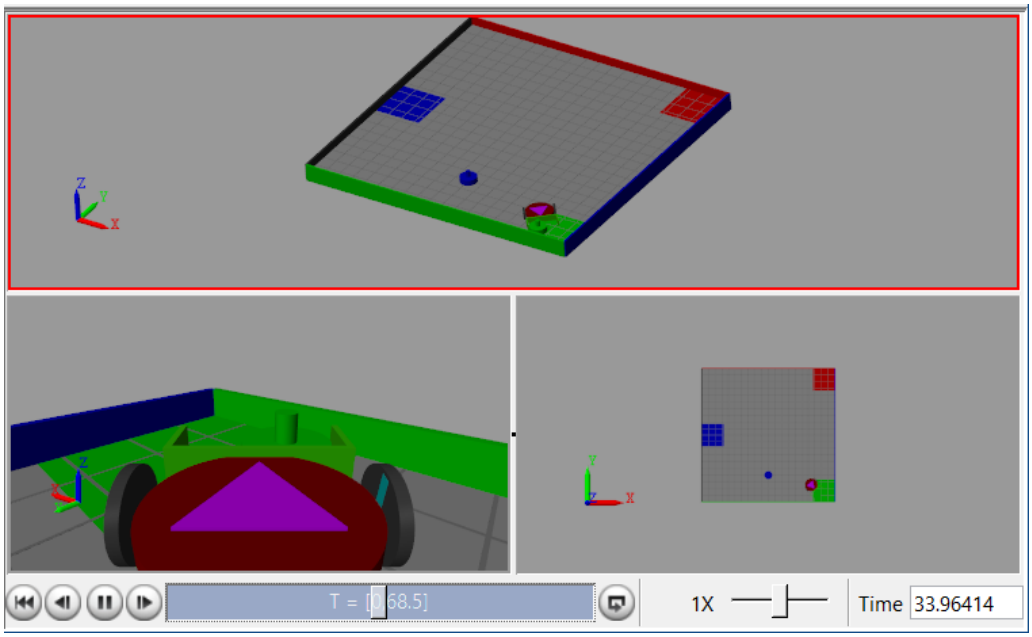

Figure 5 - Mobile robot placed the green object to its green goal position.

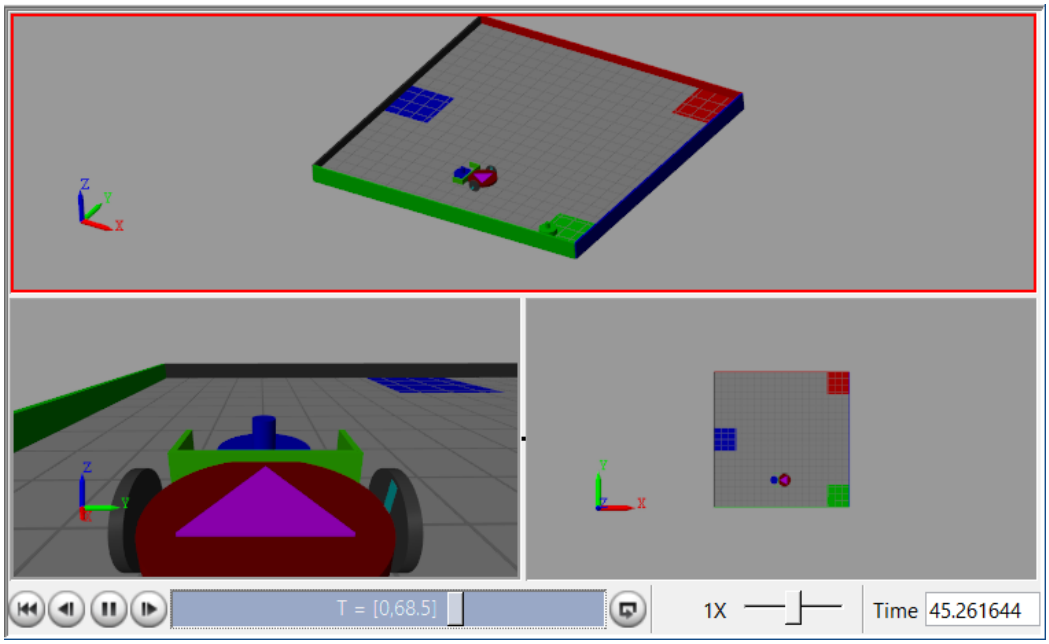

Figure 6 - After placing the green object, it placing the blue to its goal position.

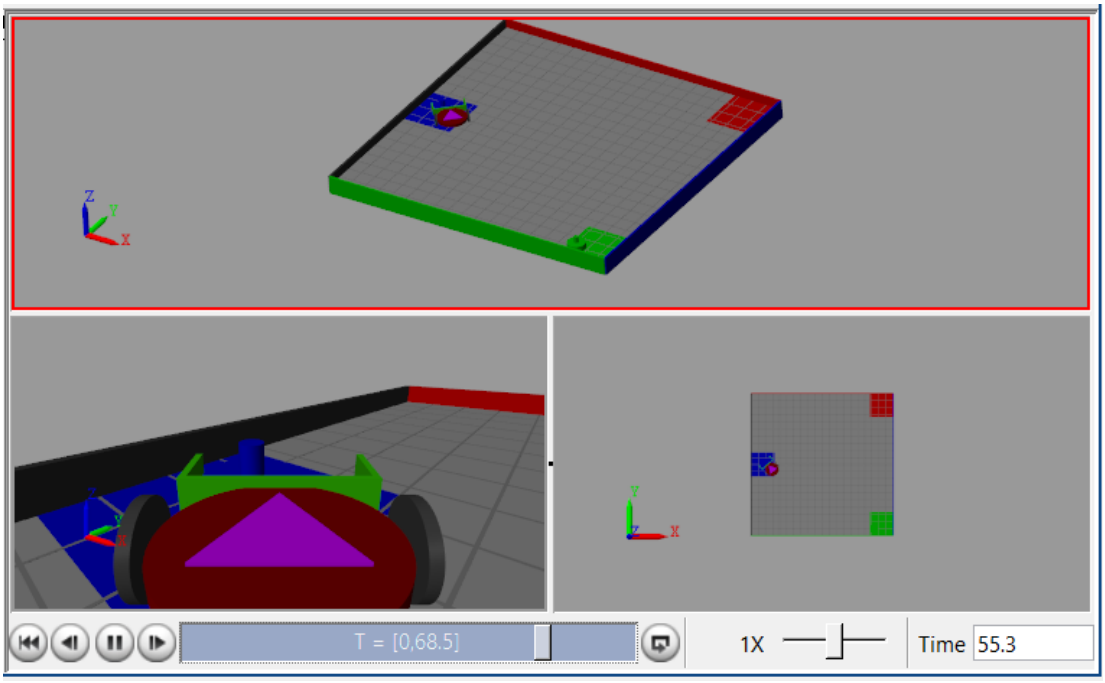

Figure 7 - The mobile robot placed the blue object at its respective blue zone.

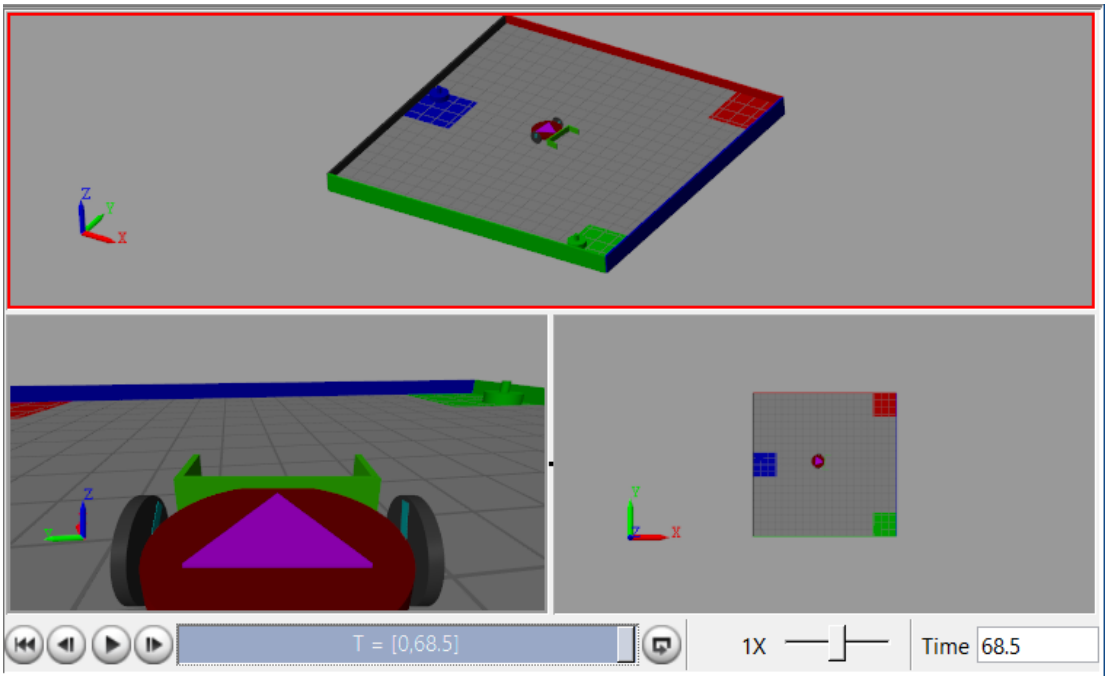

Figure 8 - Returned to the origin after completing the tasks.

Condition 2: Green and Red

Initial Configuration:

Green and Red Configuration

- Object positions: Green and red objects at [0.79,0.59], [0.78,−1.51] respectively.

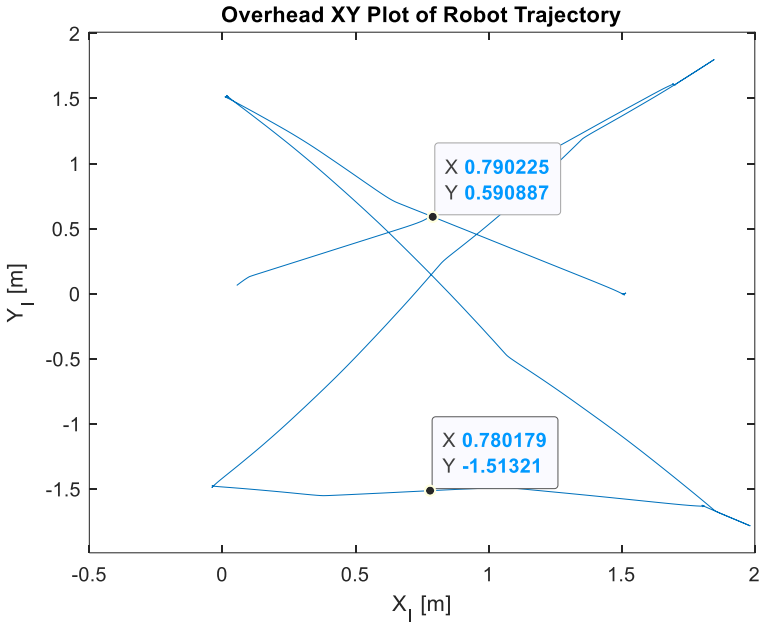

Figure 9 - Green and red object locations.

Procedure:

- The robot started at the initial position [1.5,0].
- The green and red objects are at [0.79,0.59], [0.78,−1.51] respectively.
- Next, The objects were placed at:
    - Green Object: [1.83,1.78]
    - Red Object: [1.96, -1.76]
- Task: Is to detect and transport both objects to their respective zones, then return to the origin [0,0].

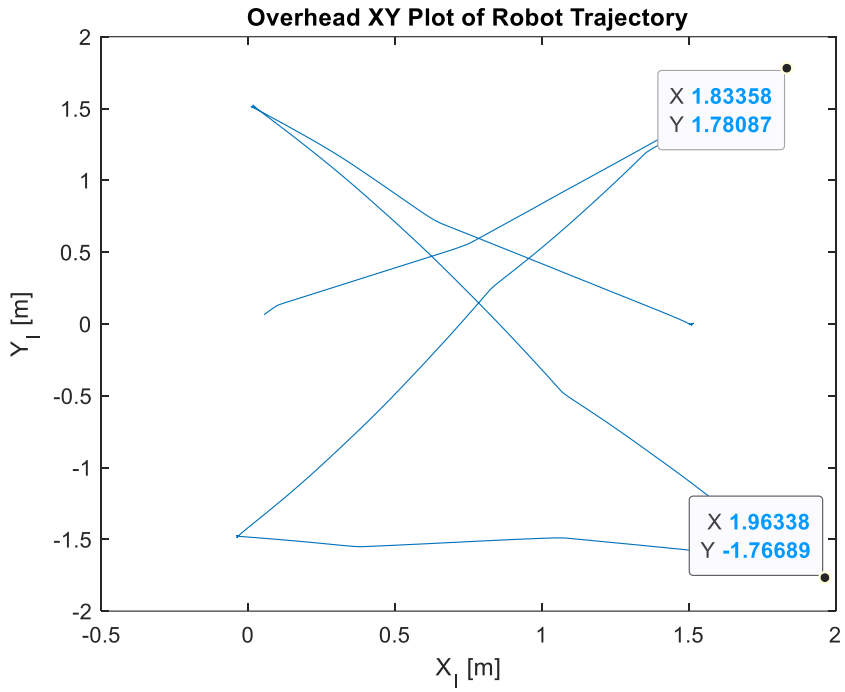

Figure 10 - Goal positions at their respective green and red zones.

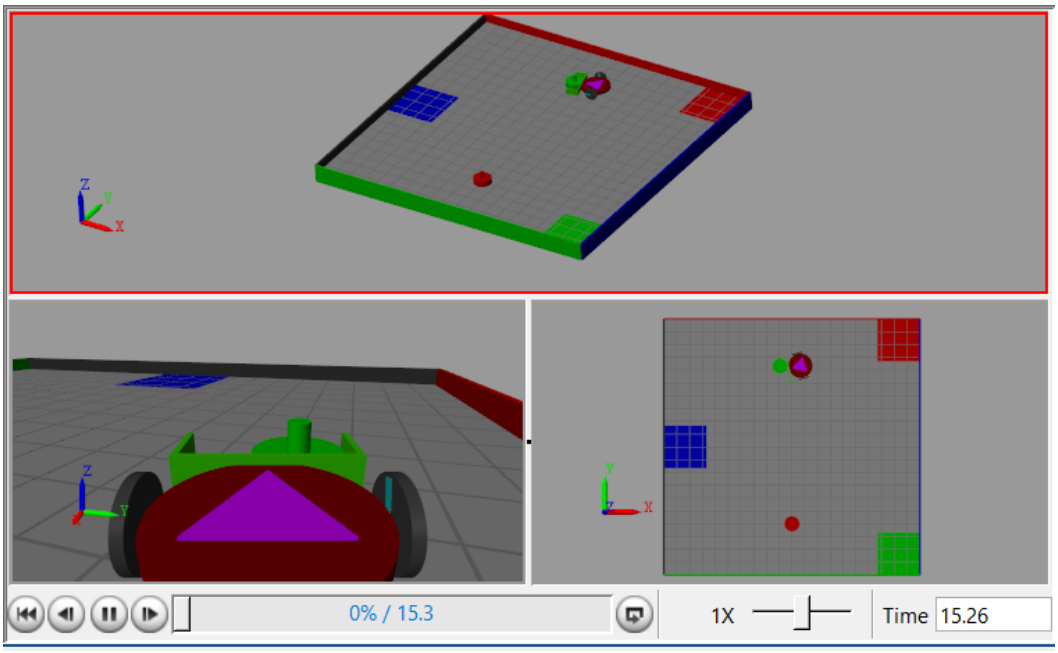

Figure 11 - Mobile robot placing the green object to its respective green zone.

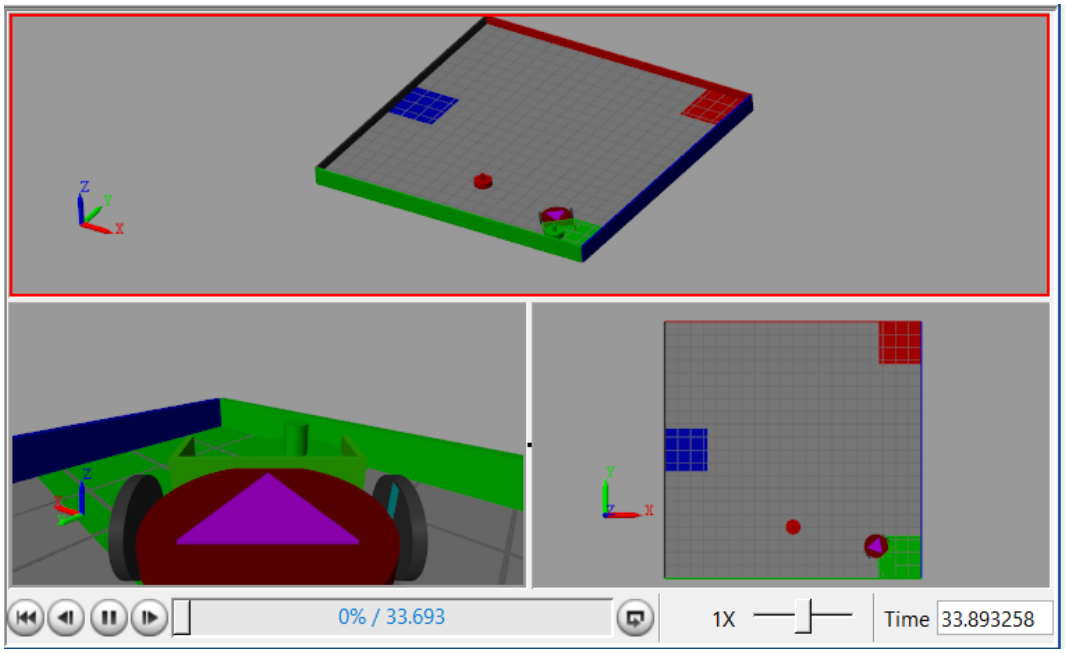

Figure 12 - Mobile robot placed the green object to its green goal position.

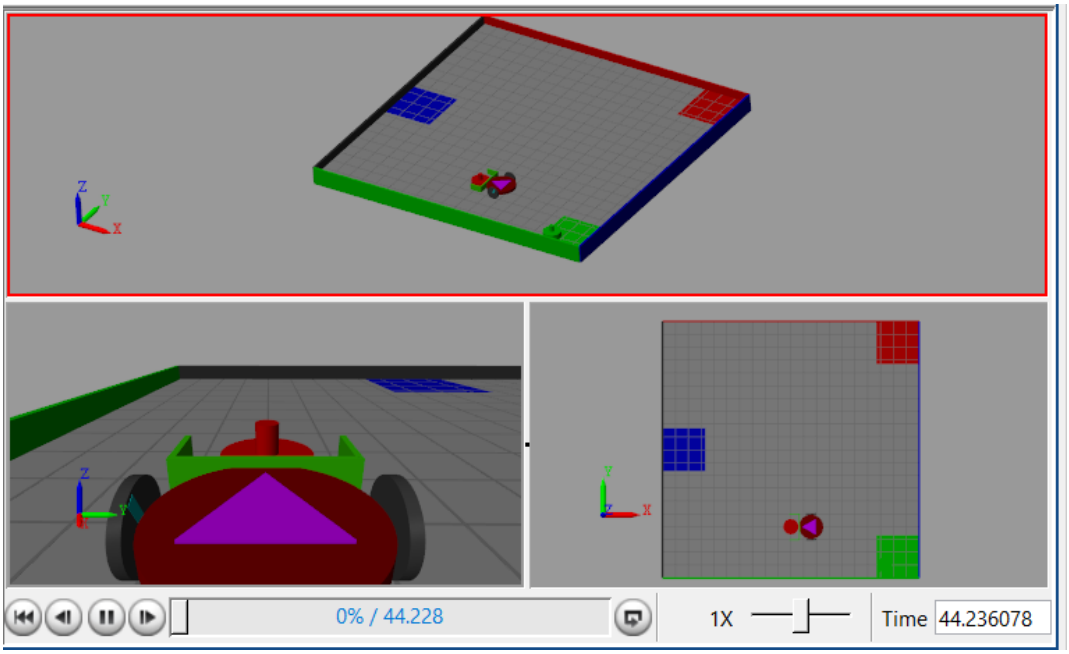

Figure 13 - After placing the green object, its placing the red to its goal position.

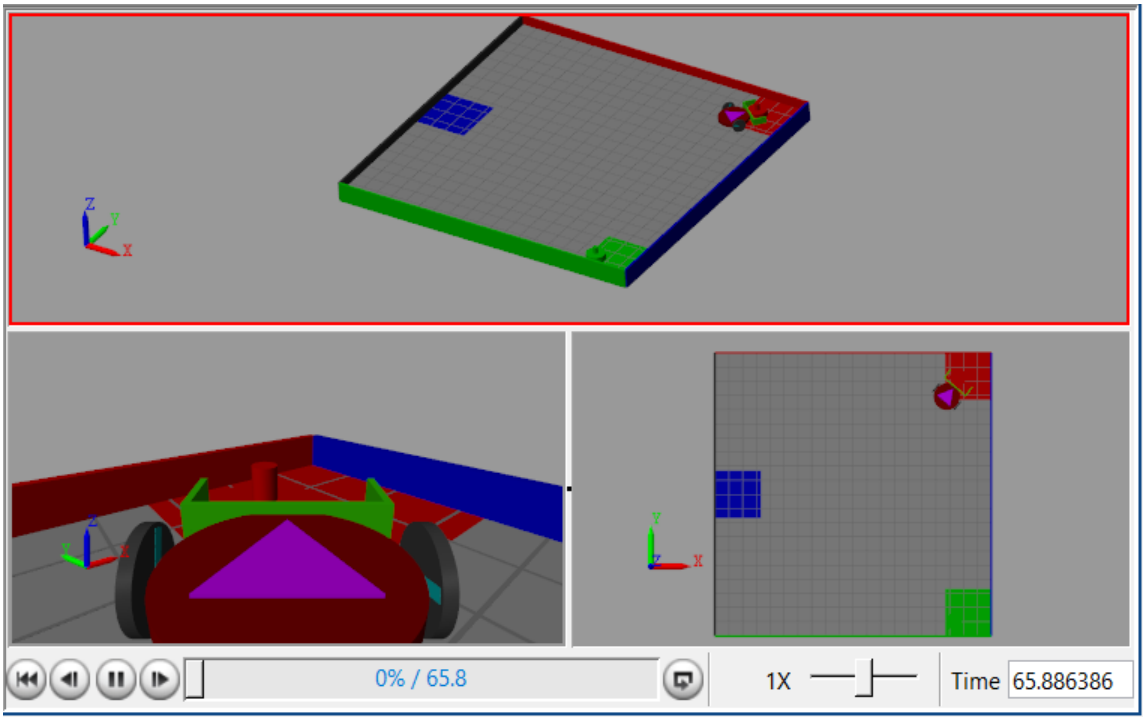

Figure 14 - The mobile robot placed the red object at its respective red zone

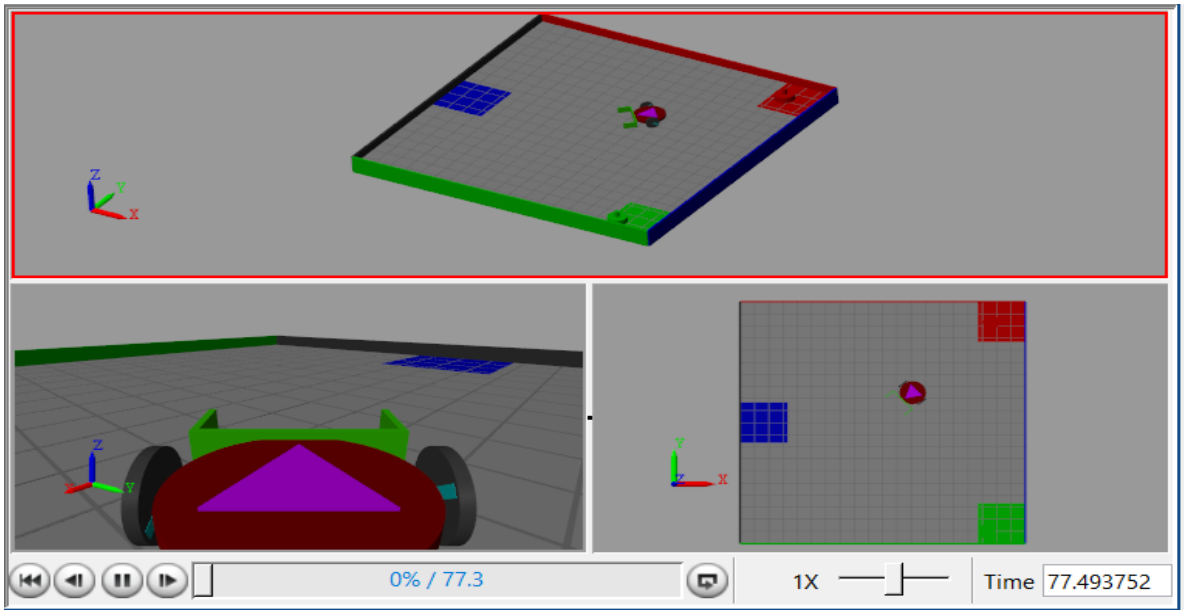

Figure 15 - Returned to the origin after completing the tasks.

Condition 3: Blue and Green

Initial Configuration:

Blue and green Configuration

- Object positions: Blue object at [0.01, 1.51], Green object at [-0.04, -1.39].

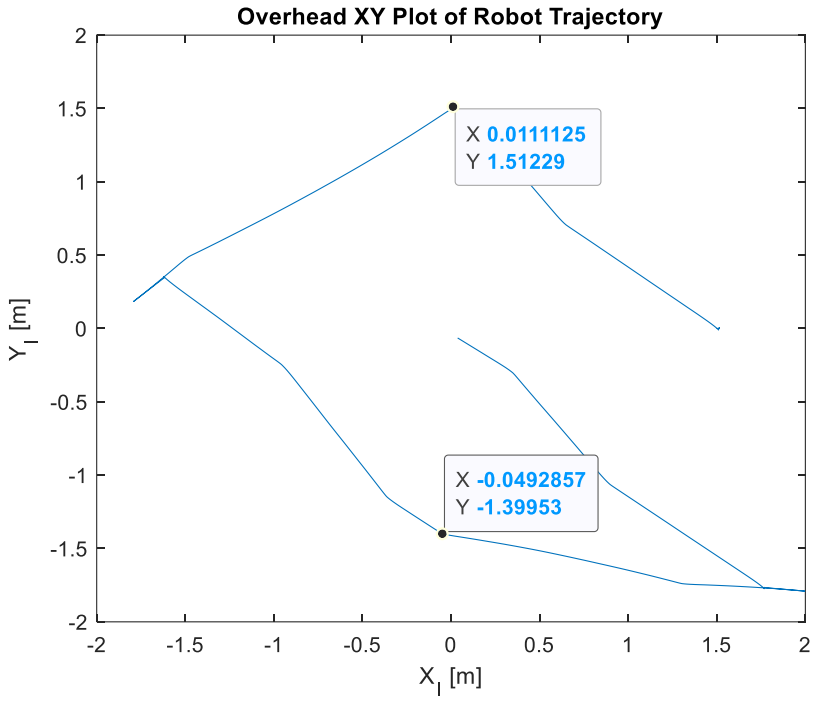

Figure 16 – Blue and green object locations.

Procedure:

- The robot started at the initial position [1.5,0].
- The blue and green objects are at [0.01,1.51], and [-0.04,−1.39] respectively.
- Next, The objects were placed at:
  - Blue Object: [-1.79, 0.18]
  - Green Object: [1.91, -1.78]
- Task: Is to detect and transport both objects to their respective zones, then return to the origin [0,0].

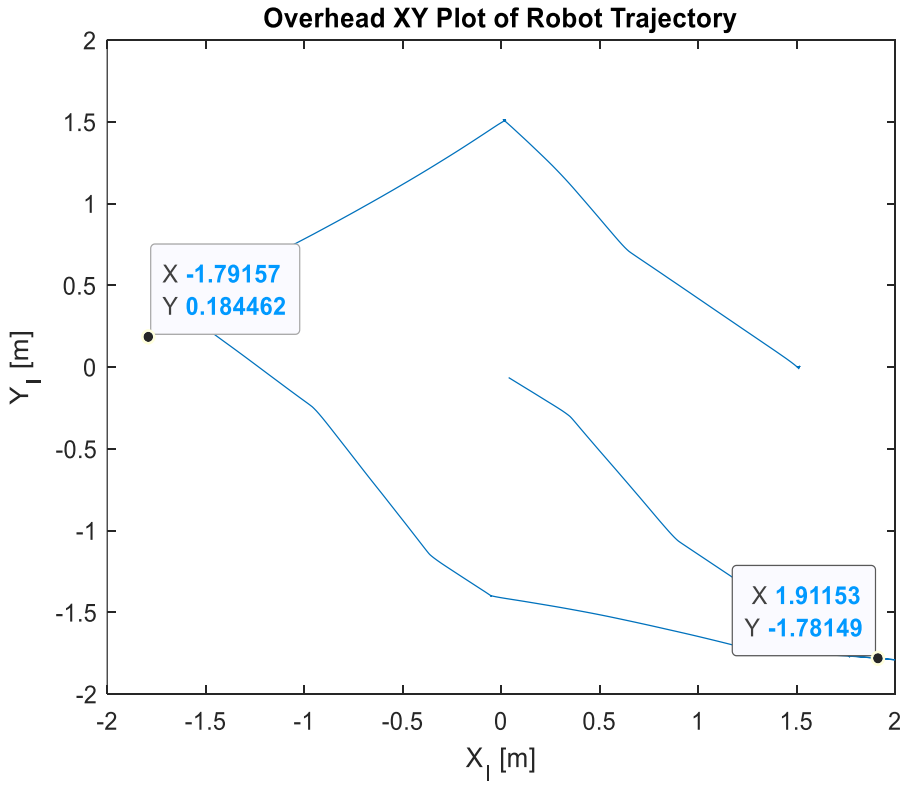

Figure 17 - Goal positions at their respective blue and green zones.

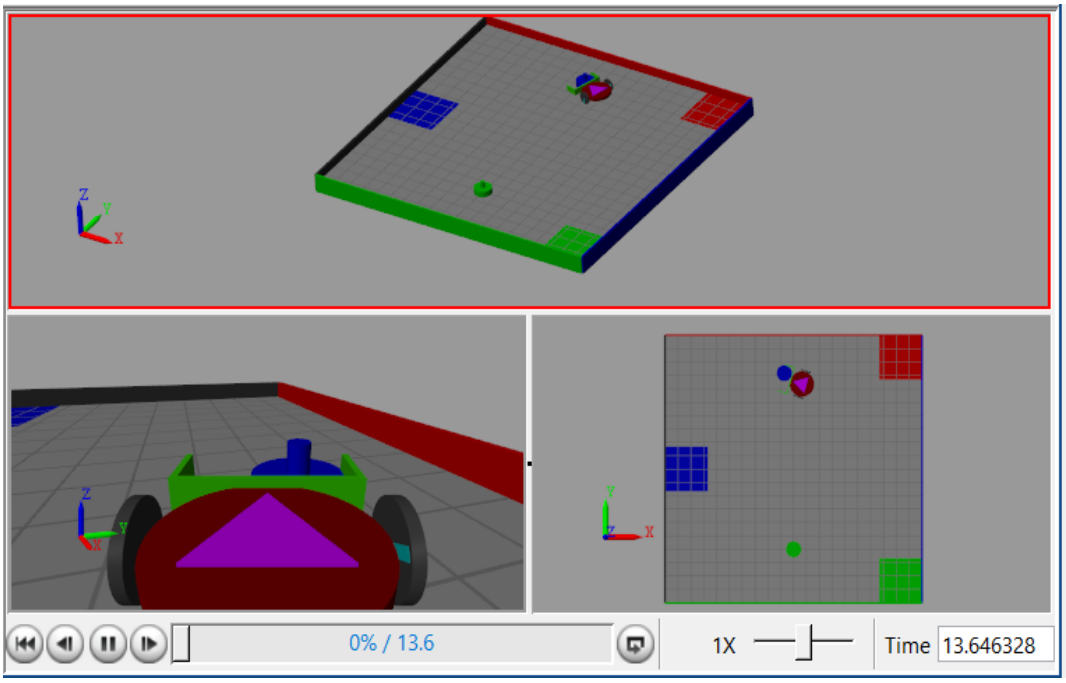

Figure 18 - Mobile robot placing the blue object to its respective blue zone.

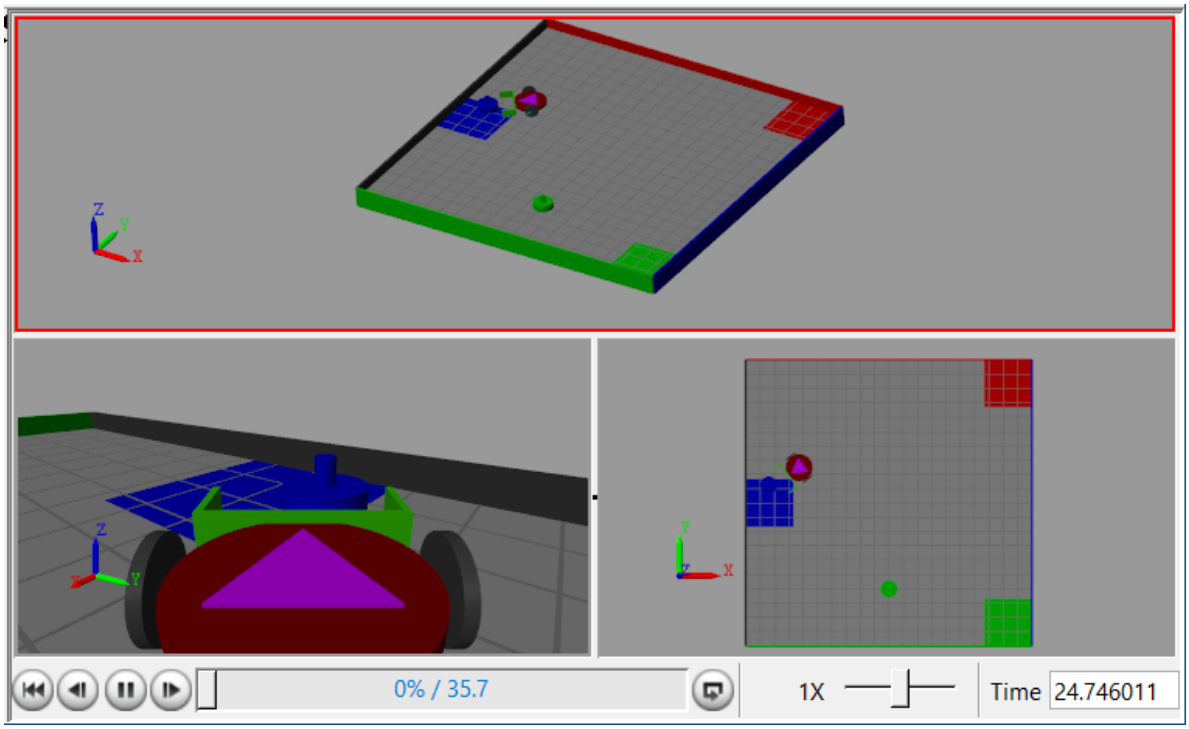

Figure 19 - Mobile robot placed the blue object to its blue goal position.

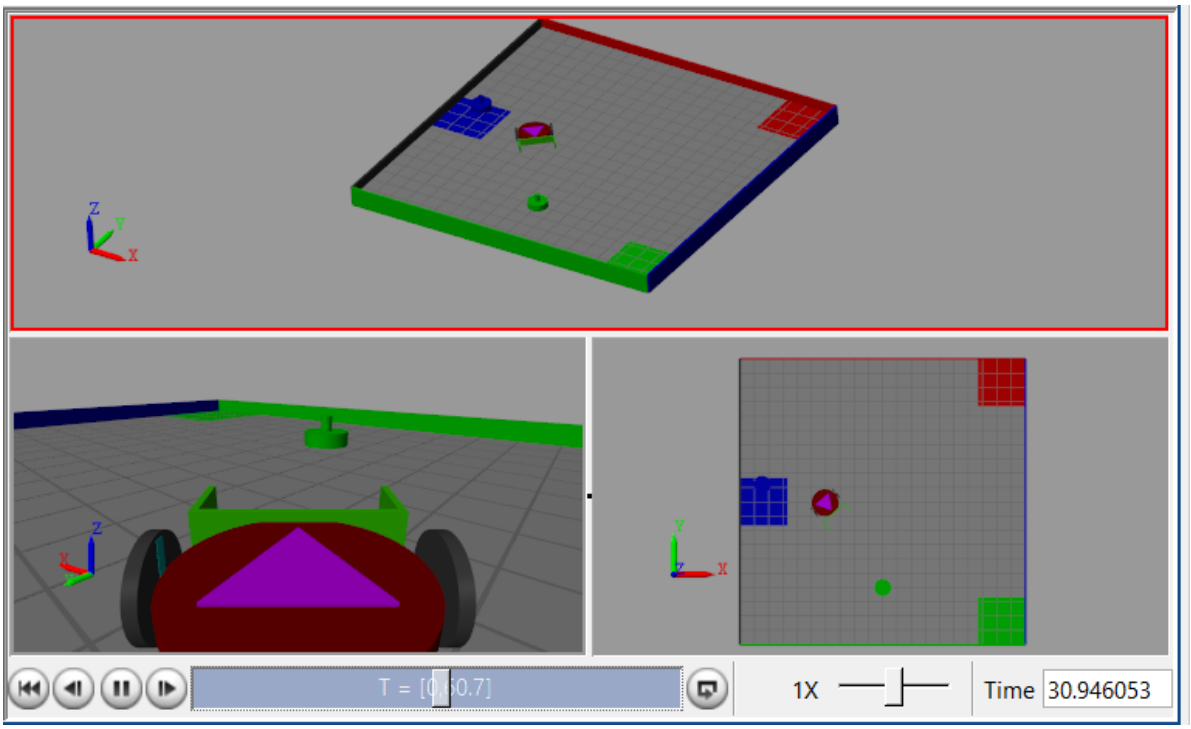

Figure 20 - After placing blue at its goal position, it's moving towards green object.

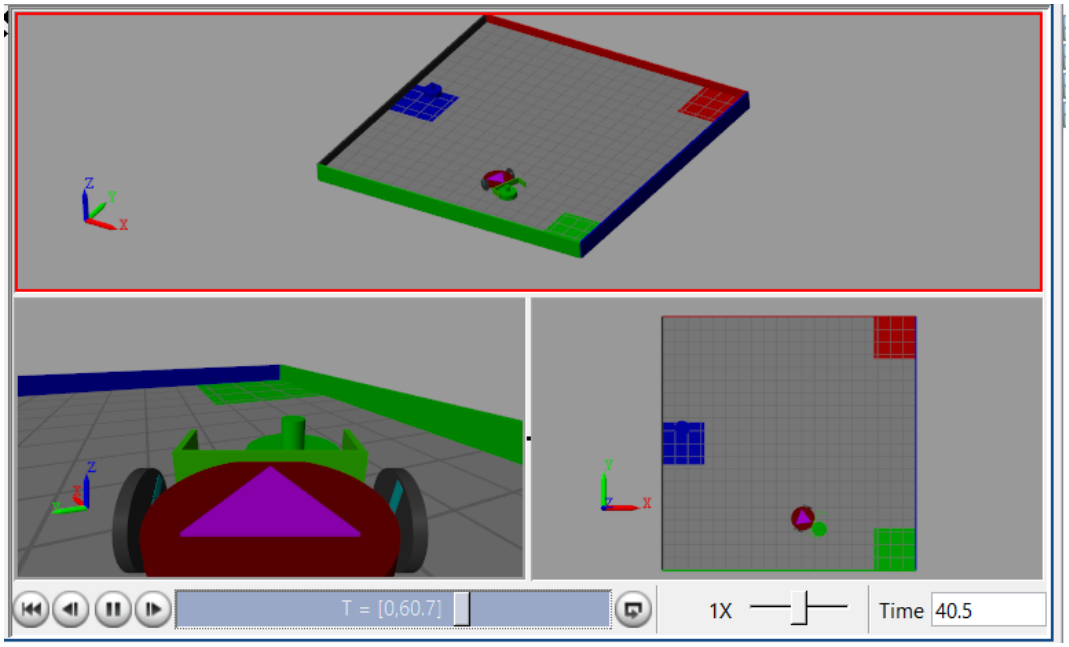

Figure 21 – The mobile robot is placing green object to its green zone.

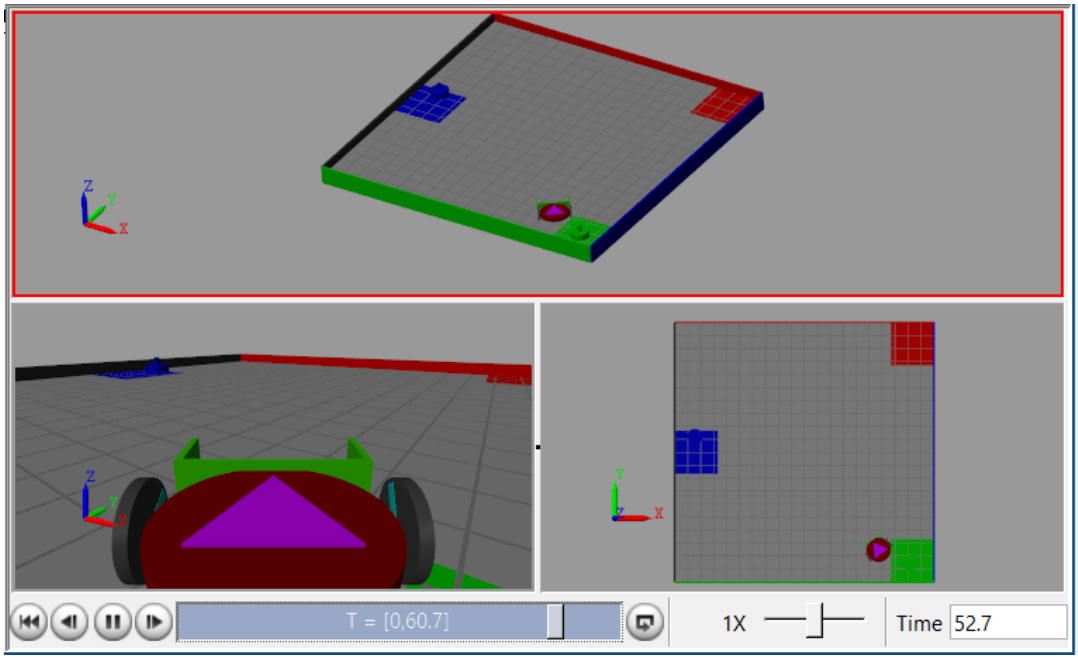

Figure 22 – Returning to the origin.

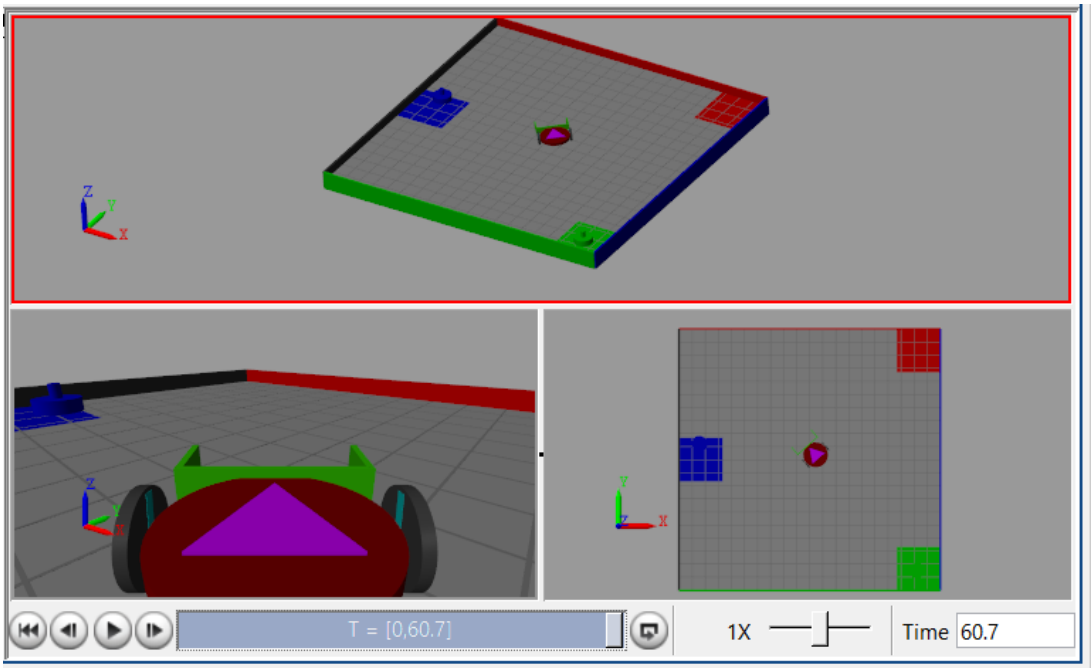

Figure 23 - Returned to the origin after completing the tasks.

Condition 4: Blue and Red

Initial Configuration:

Blue and Red Configuration

- Object positions: Blue object at [0.01, 1.50], Red object at [-0.04, -1.40].

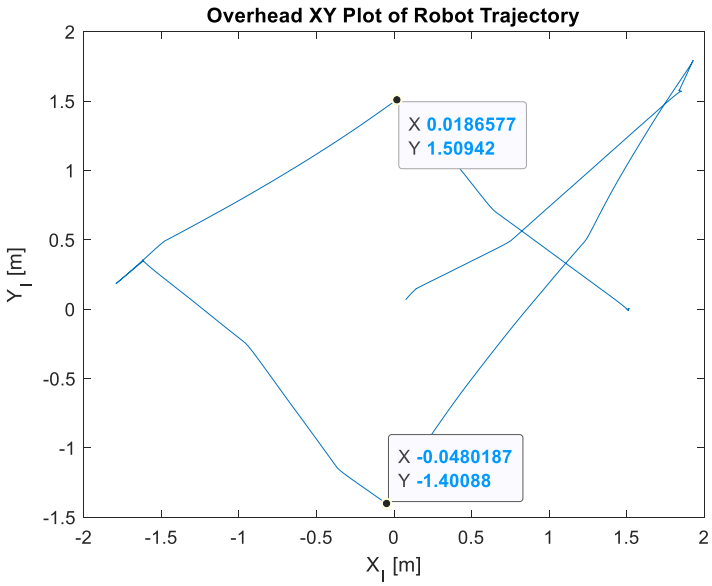

Figure 24 - Blue and Red object locations.

Procedure:

- The robot started at the initial position [1.5,0].
- The blue and red objects are at [0.01,1.50], and [-0.04,−1.40] respectively.
- Next, The objects were placed at:
    - Blue Object: [-1.79, 0.18]
    - Red Object: [1.92, -1.79]
- Task: Is to detect and transport both objects to their respective zones, then return to the origin [0,0].

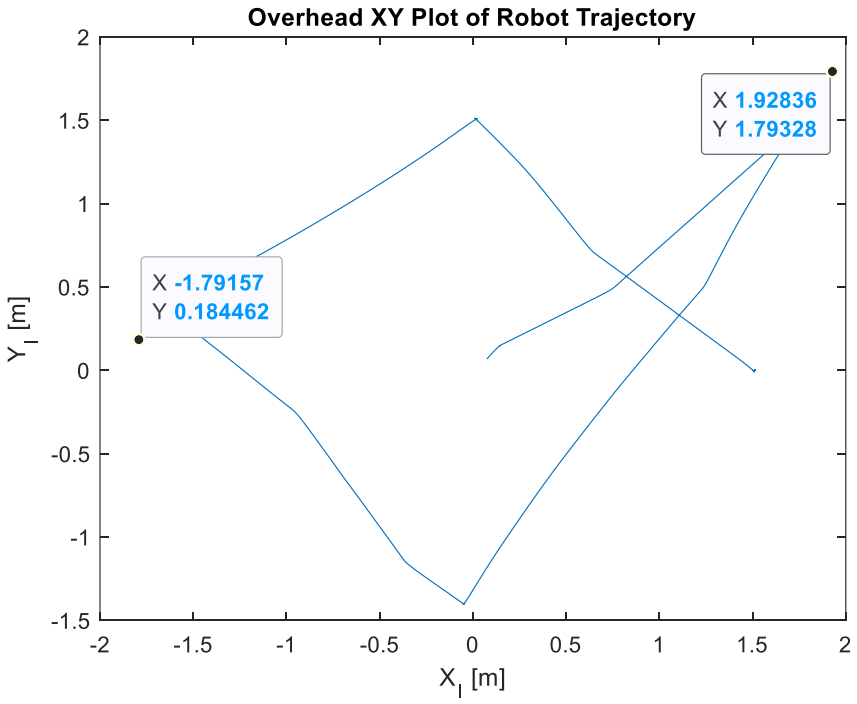

Figure 25 - Goal positions at their respective blue and red zones.

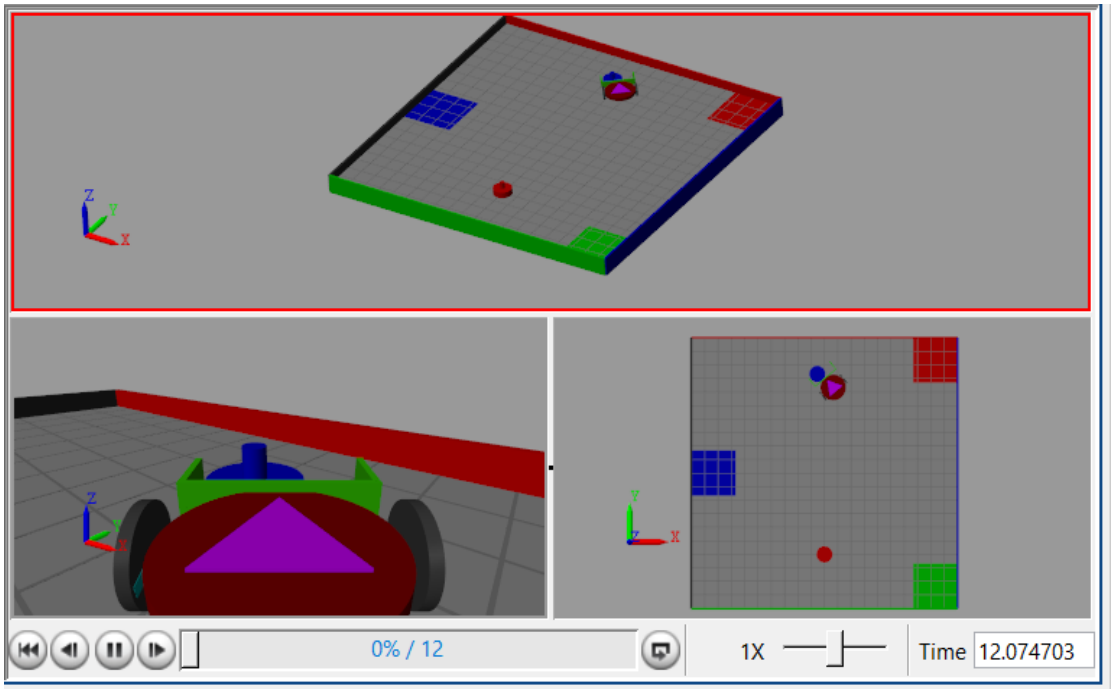

Figure 26 - Mobile robot placing the blue object to its respective blue zone.

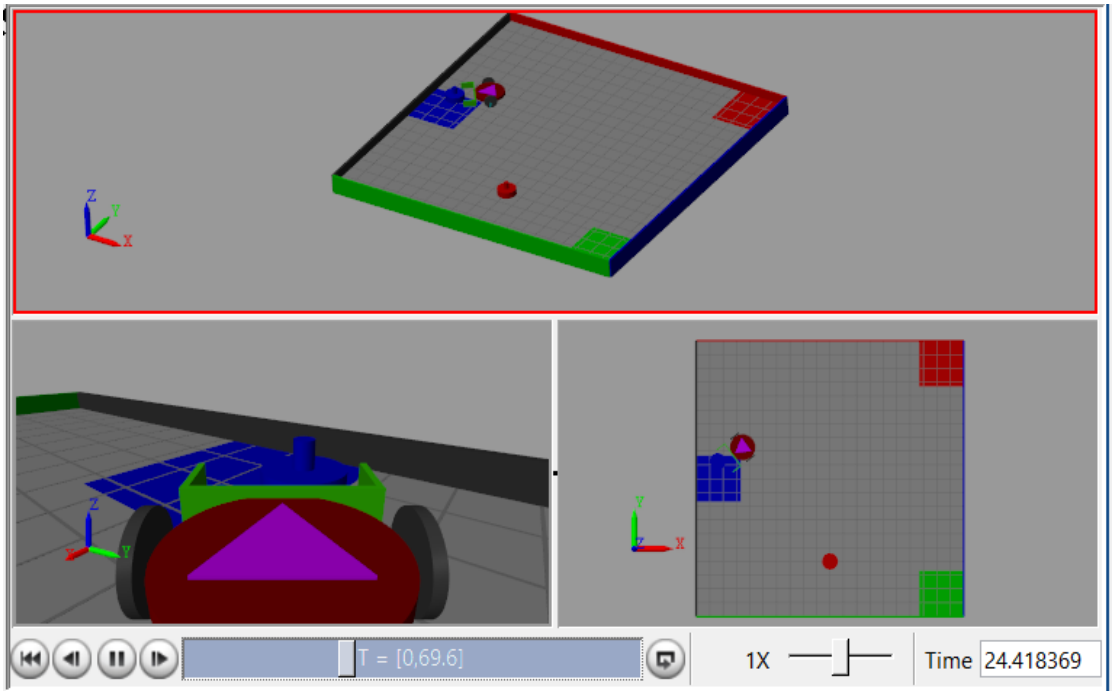

Figure 27 - Mobile robot placed the blue object to its blue goal position.

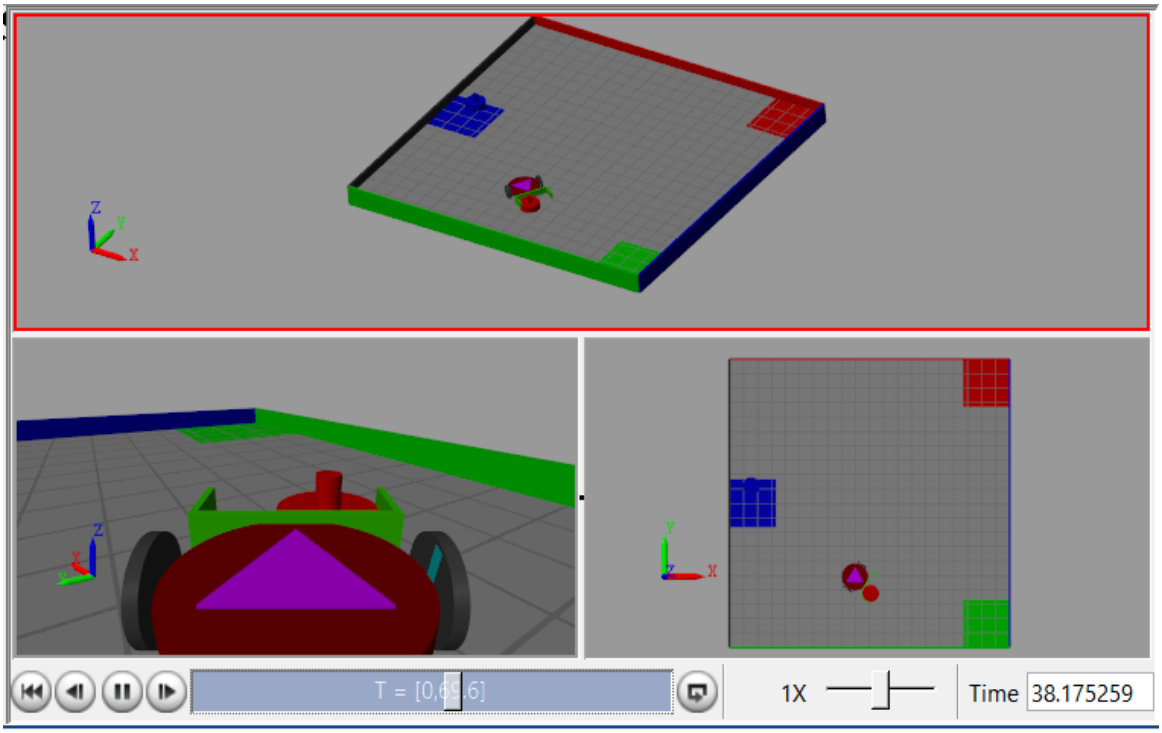

Figure 28 - After placing the blue object, it is placing the red object to its goal position.

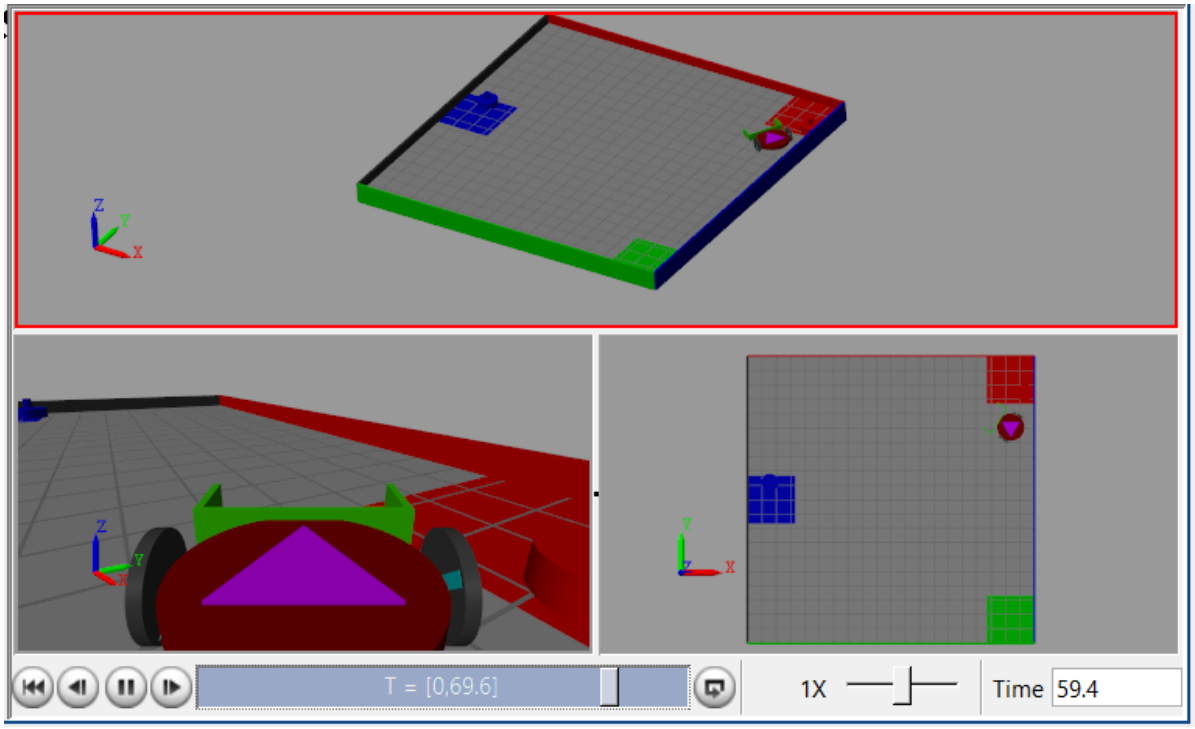

Figure 29 - The mobile robot placed the red object at its respective red zone

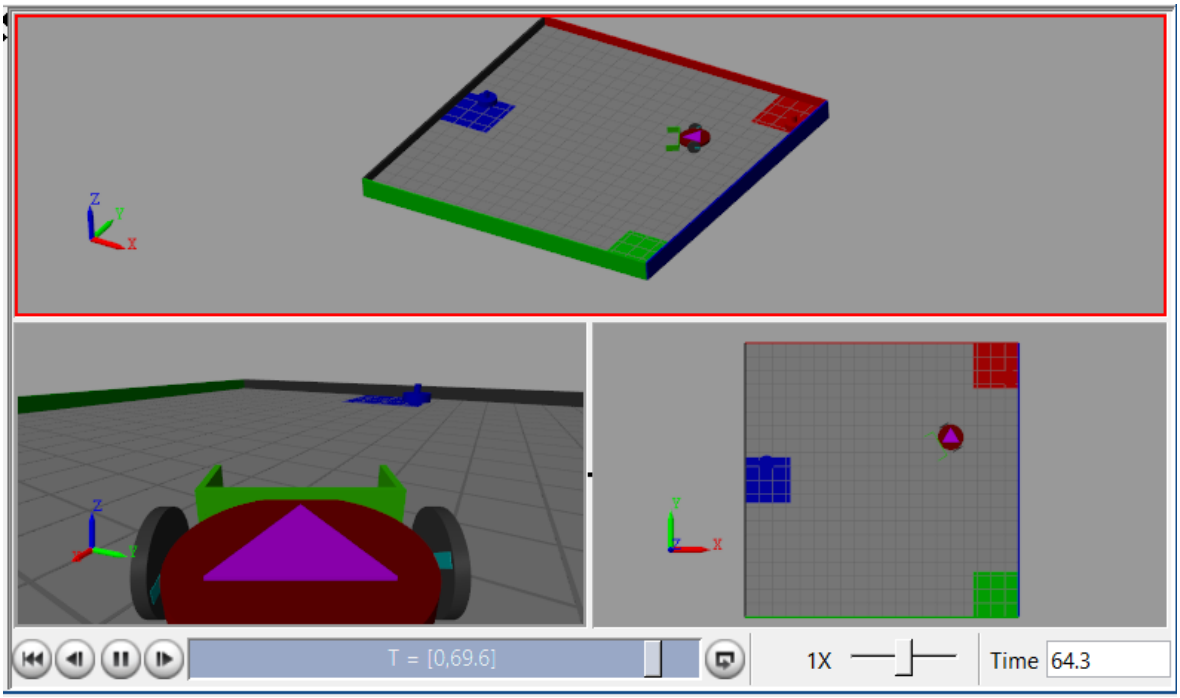

Figure 30 - Returned to the origin after completing the tasks.

Condition 5: Red and Green

Initial Configuration:

Red and Green Configuration

- Object positions: Red object at [0.77, 0.59], Green object at [0.64, -0.95].

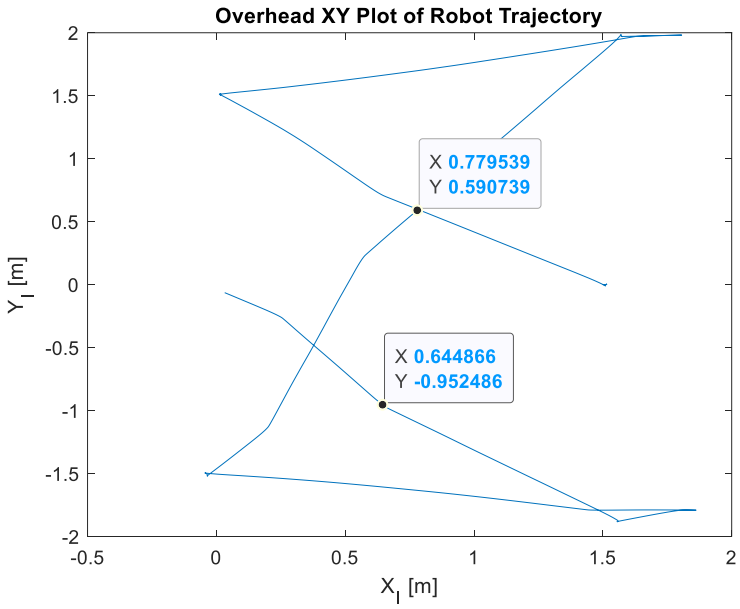

Figure 31 – Red and green object locations.

Procedure:

- The robot started at the initial position [1.5,0].
- The red object is at [0.77, 0.59], Green object at [0.64, -0.95].
- Next, The objects were placed at:
    - Red Object: [1.77, 1.97]
    - Green Object: [1.85, -1.78]
- Task: Is to detect and transport both objects to their respective zones, then return to the origin [0,0].

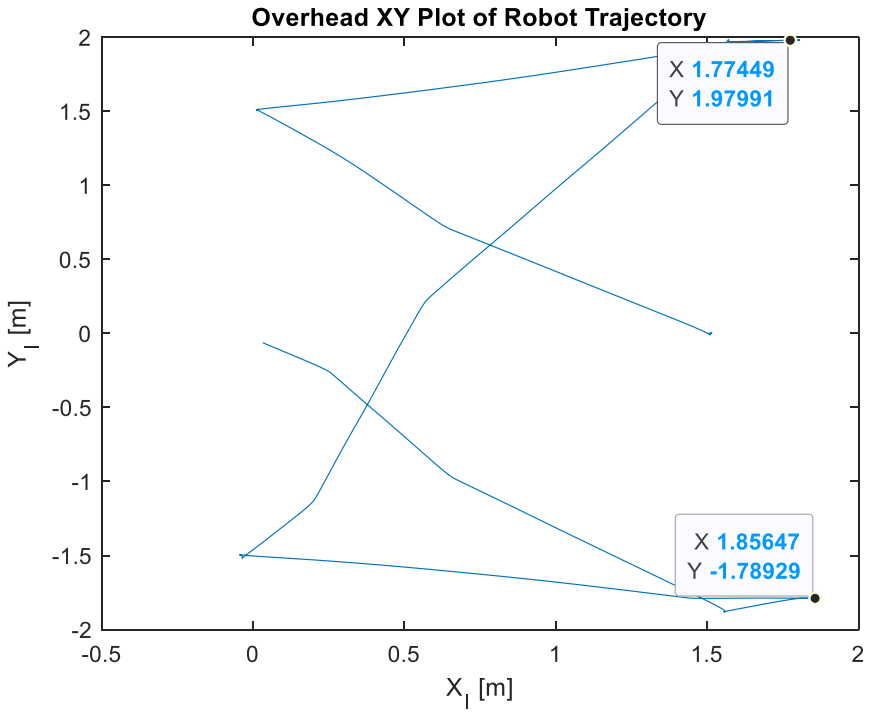

Figure 32 - Goal positions at their respective red and green zones.

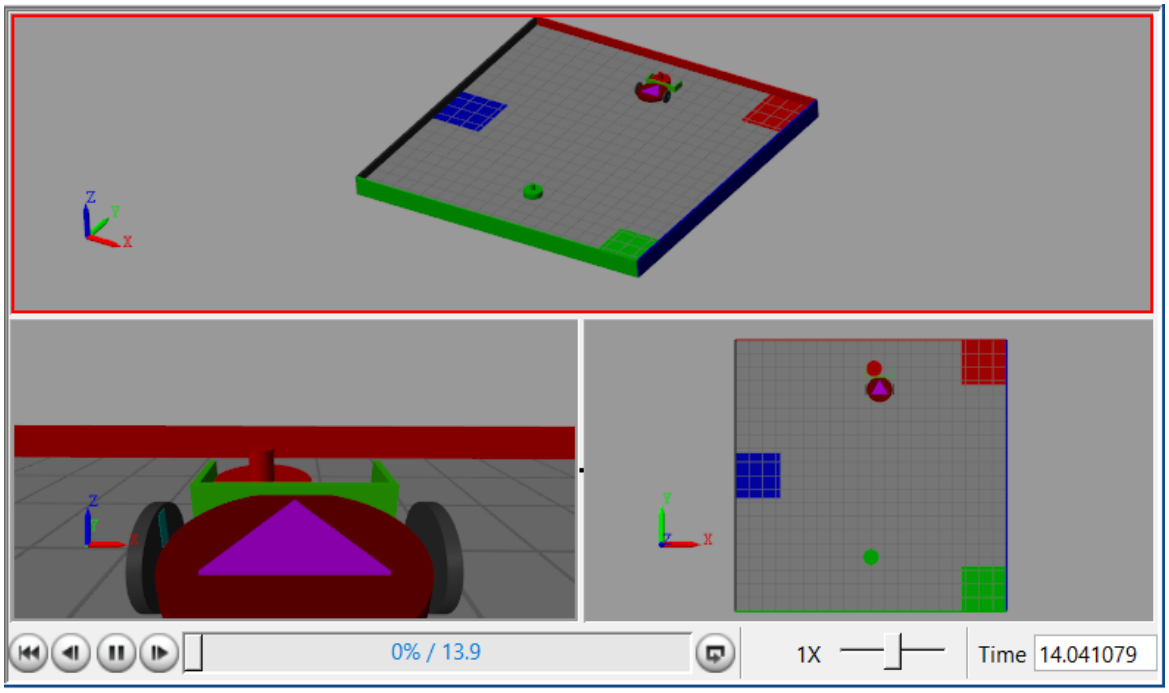

Figure 33 - Mobile robot placing the red object to its respective red zone.

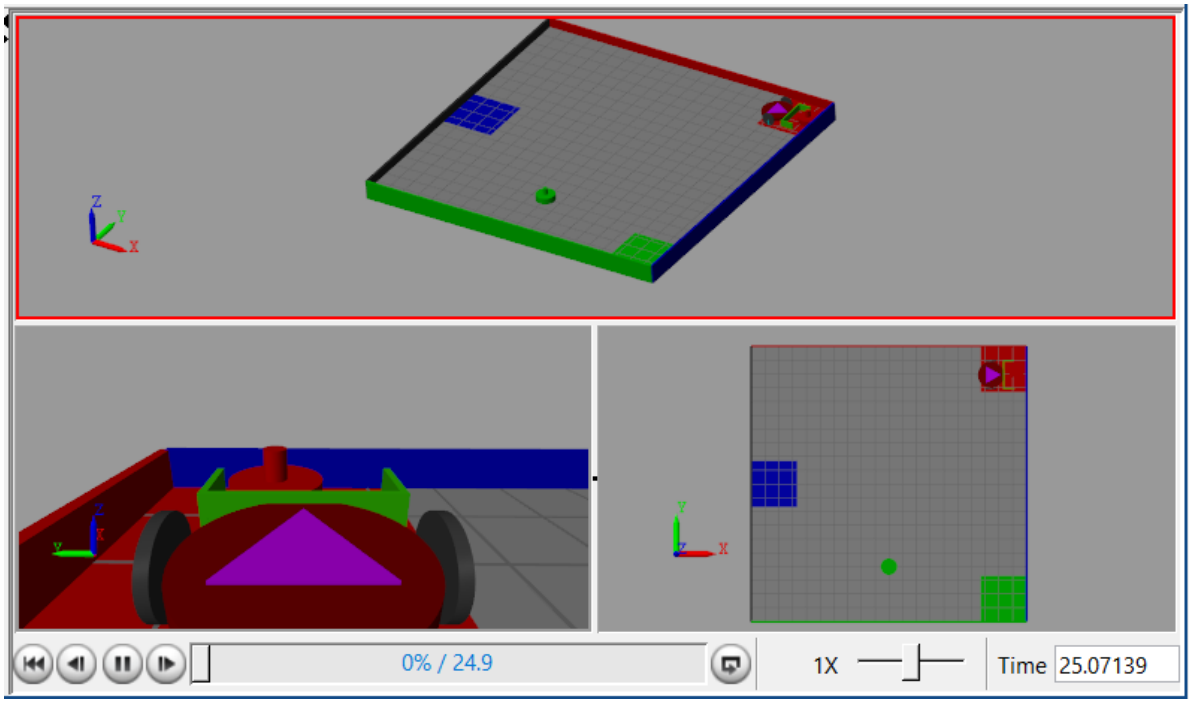

Figure 34 - Mobile robot placed the red object to its red goal position.

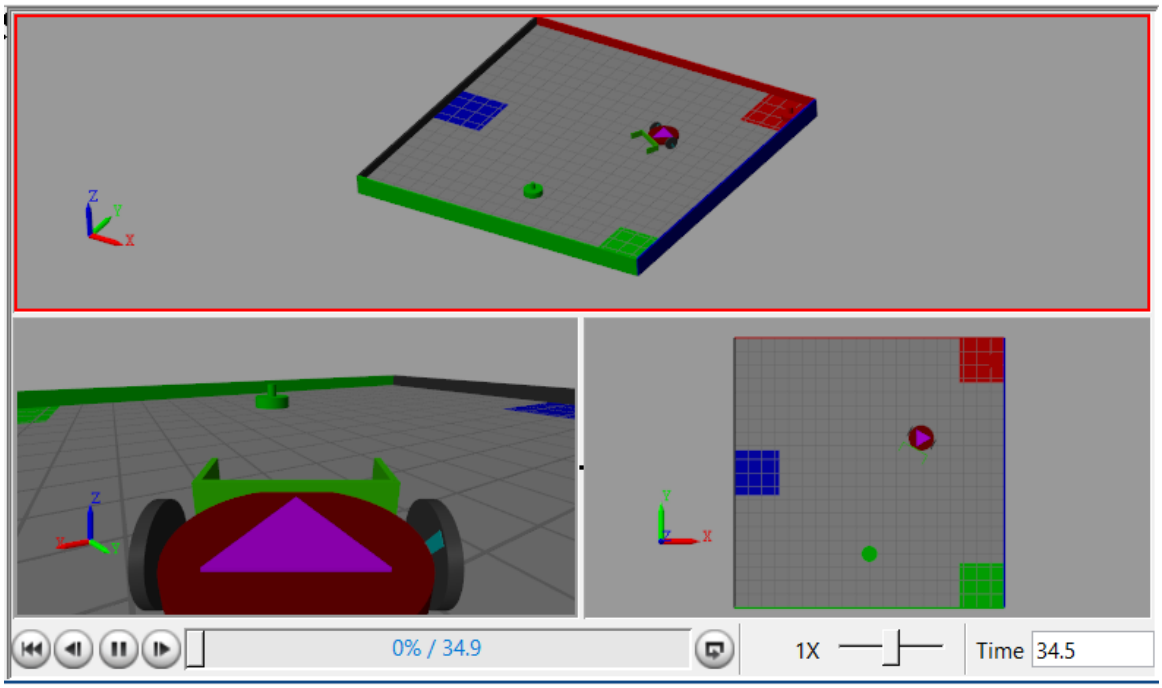

Figure 35 - After placing the red object, it is placing the green object to its goal position.

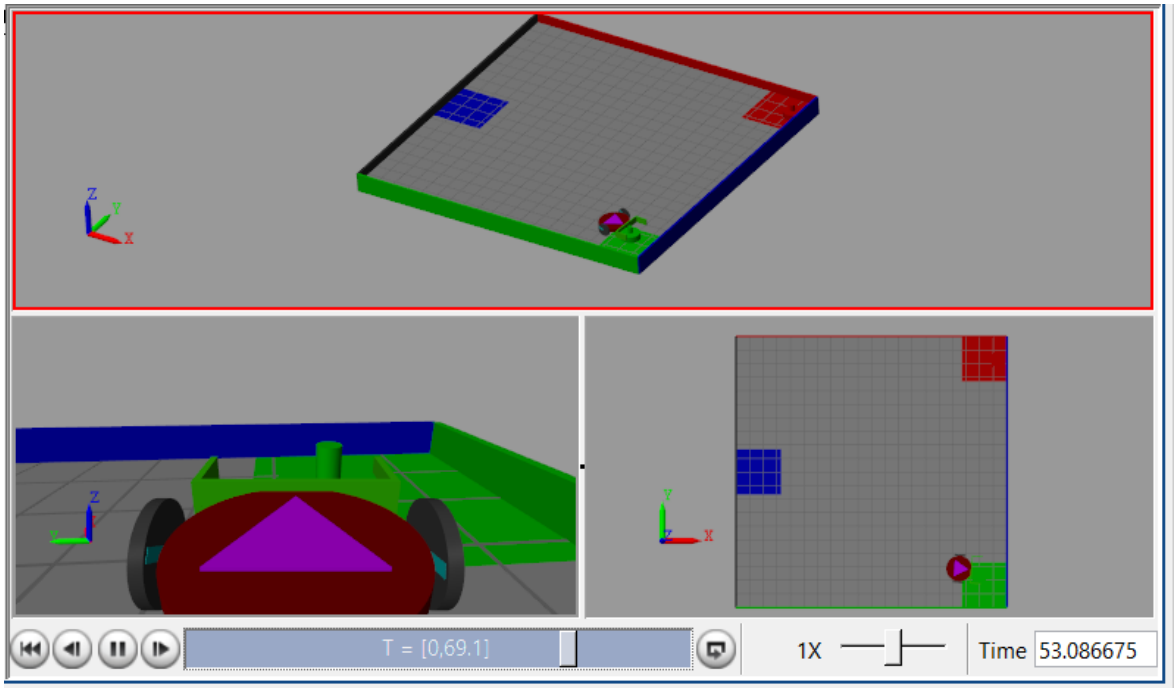

Figure 36 - The mobile robot placed the green object at its respective green zone

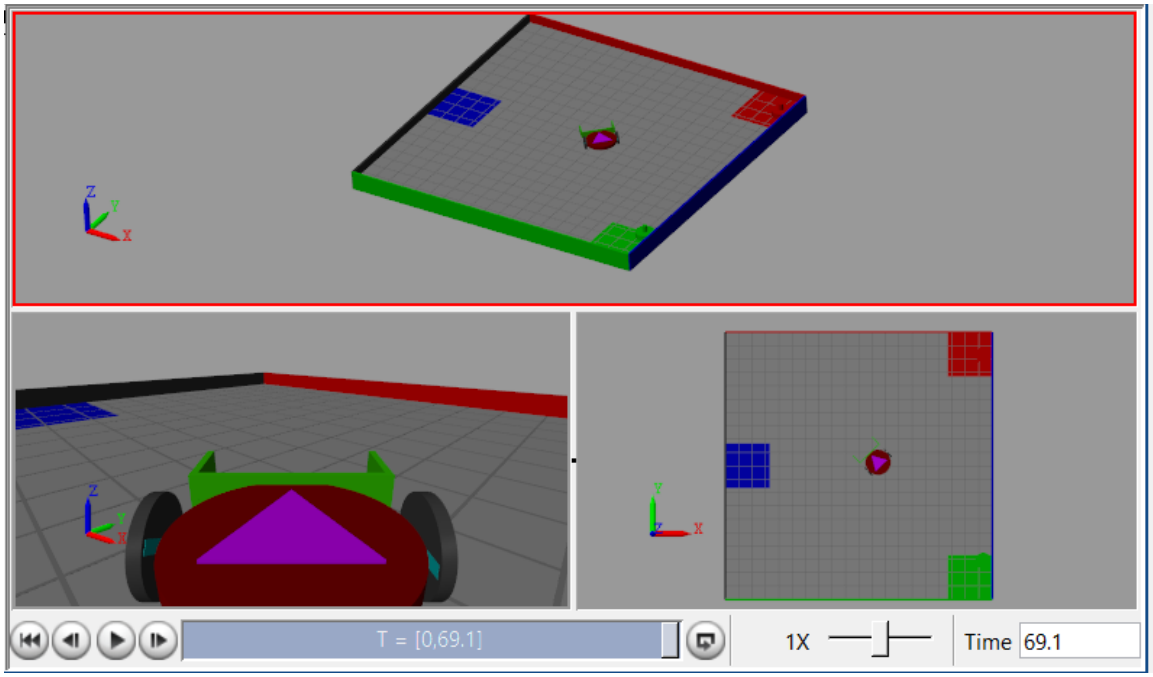

Figure 37 - Returned to the origin after completing the tasks.

Condition 6: Red and Blue

Initial Configuration:

Red and blue Configuration

- Object positions: Red and blue objects at [0.01, 1.50], [-0.04, -1.51] respectively.

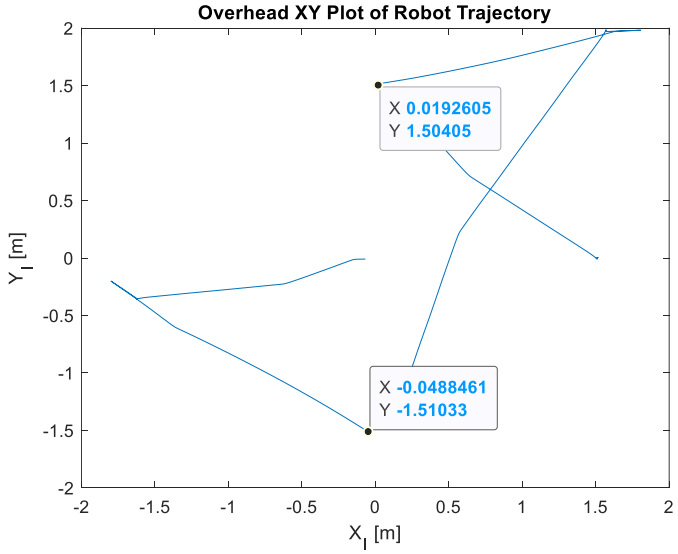

Figure 38 - Red and Blue object locations.

Procedure:

- The robot started at the initial position [1.5,0].
- The red and blue objects are at [0.01, 1.50], [-0.04, -1.51] respectively.
- Next, The objects were placed at:
    - Red Object: [1.78, 1.98]
    - Blue Object: [-1.79, -0.20]
- Task: Is to detect and transport both objects to their respective zones, then return to the origin [0,0].

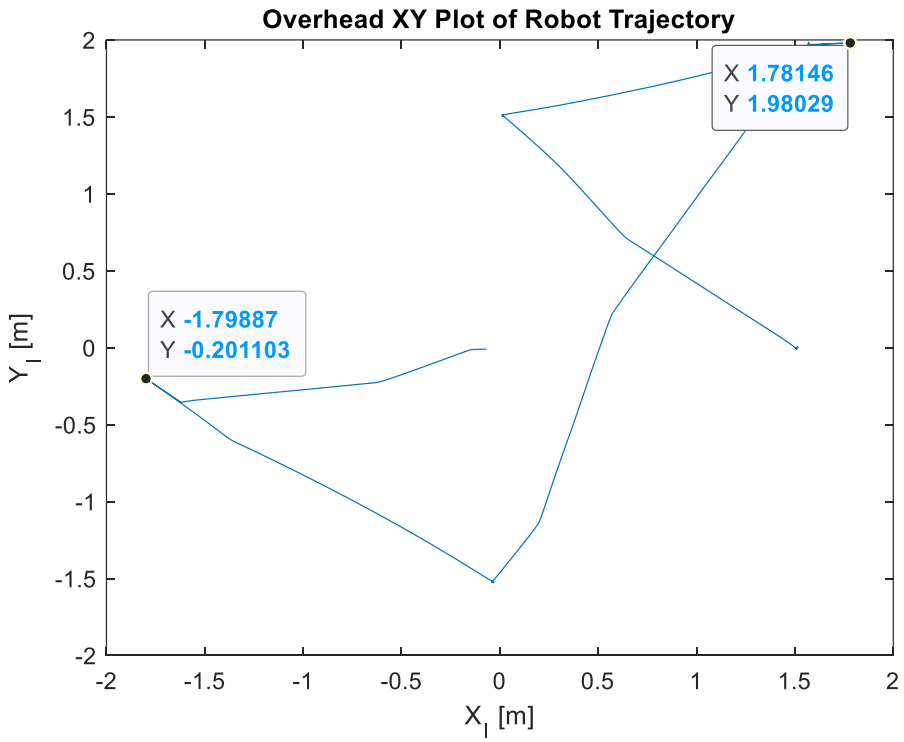

Figure 39 - Goal positions at their respective red and blue zones.

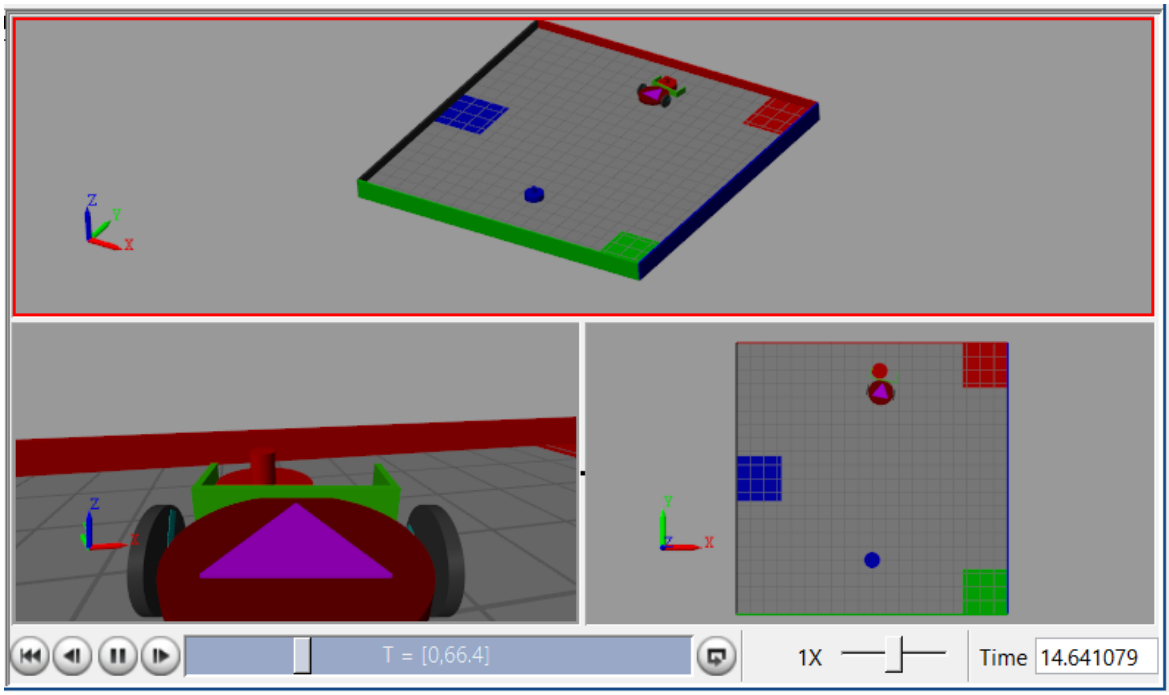

Figure 40 - Mobile robot placing the red object to its respective red zone.

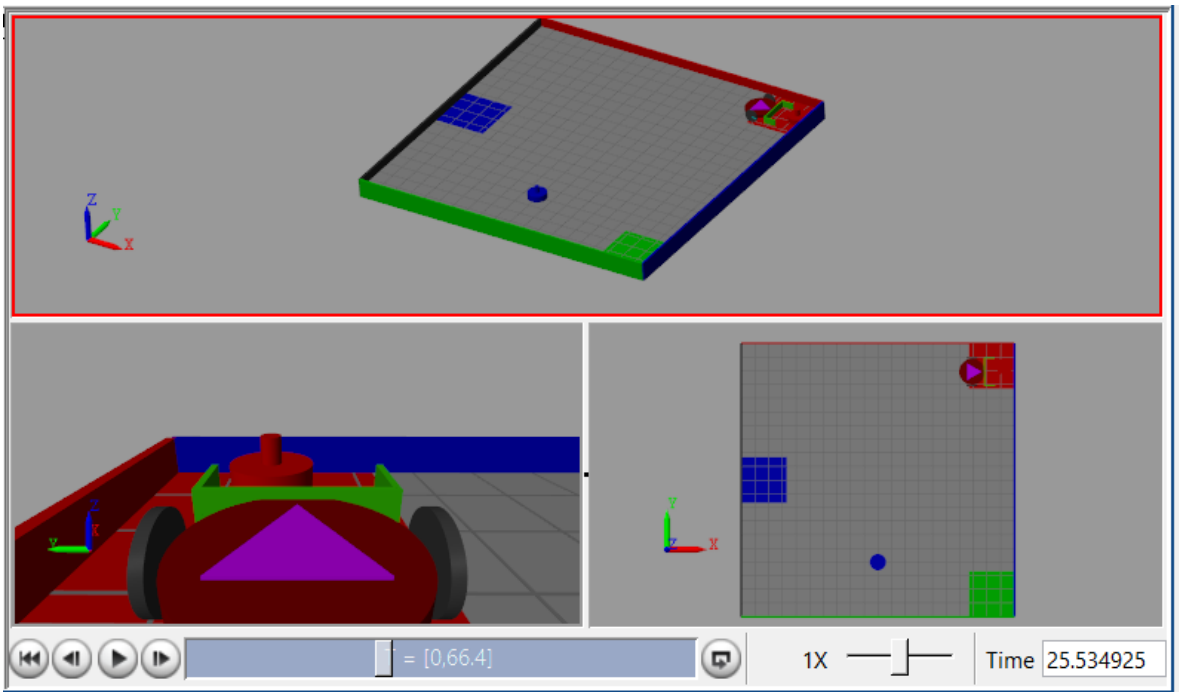

Figure 41 - Mobile robot placed the red object to its red goal position.

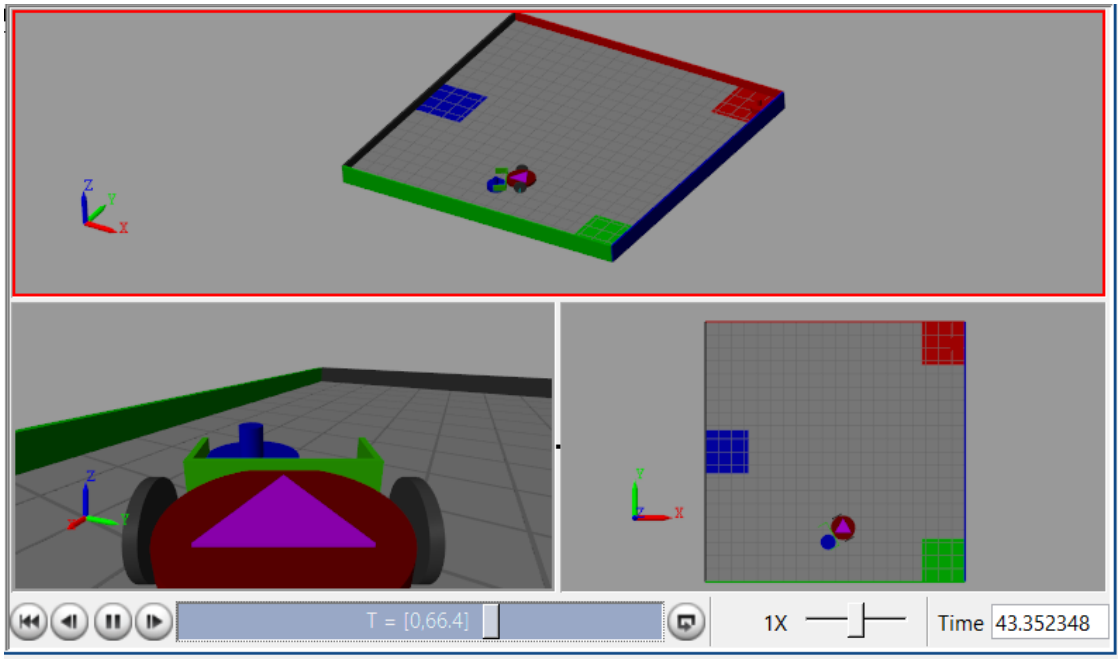

Figure 42 - After placing the red object, its placing the blue to its goal position.

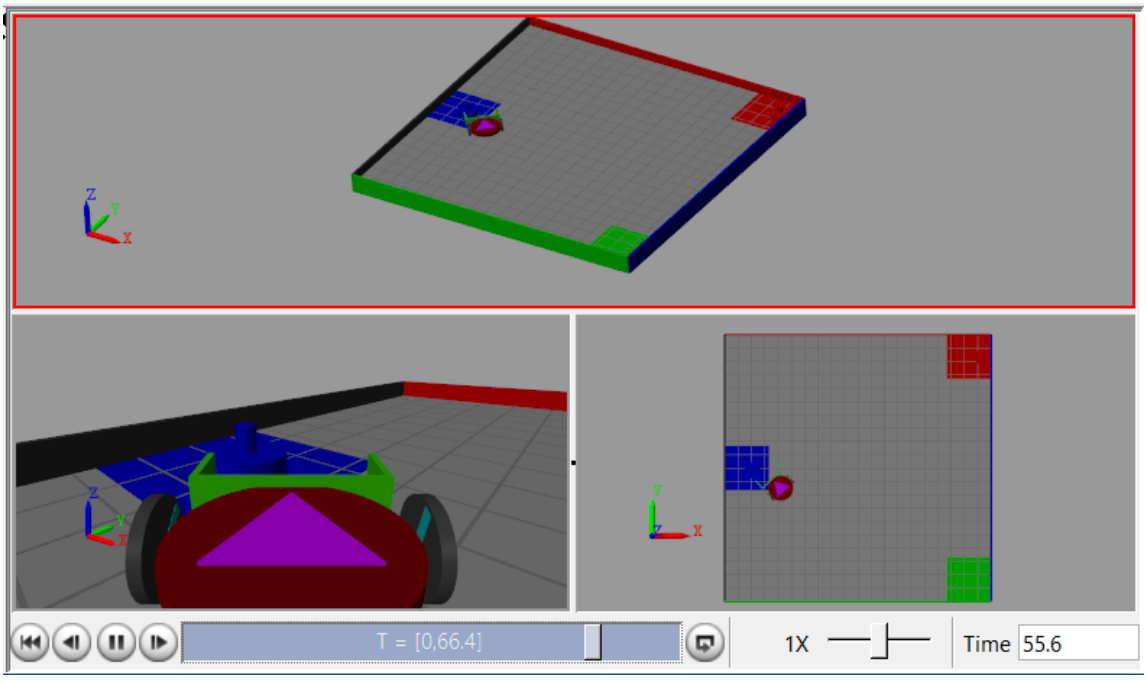

Figure 43 - The mobile robot placed the blue object at its respective blue zone

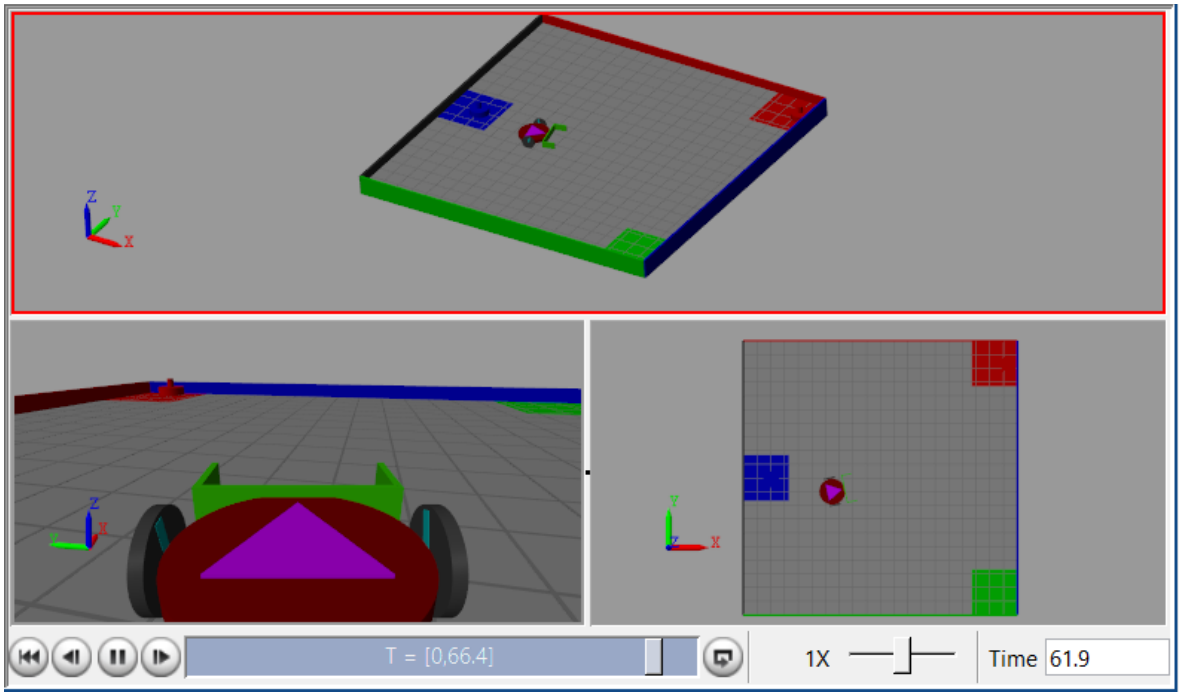

Figure 44 - Returned to the origin after completing the tasks.

Initial Conditions:	Top Color:	Bottom Color:	Success?	Time (seconds)	Best
1	Green	Blue	Yes	61.5	
2	Green	Red	Yes	73.2	
3	Blue	Green	Yes	52.7	
4	Blue	Red	Yes	63	
5	Red	Green	Yes	62.1	
6	Red	Blue	Yes	60.2	YES

Table 1 - Time taken by the robot to complete the given tasks.

How Accurate / repeatable was your system's performance?

The system demonstrated a 100% success rate across all six initial conditions, as the robot consistently identified, sorted, and placed objects in their designated zones successfully.

1. Accuracy:
   The robot consistently achieved the task of sorting objects with accurate color recognition and correct placement into respective zones. Minor variations in time were observed, but no failures occurred, indicating robust and reliable sensor performance and navigation algorithms.
2. Repeatability:
   Across six trials, the system exhibited consistent behaviour. Task completion times varied slightly due to the initial object positions and environmental factors. The best performance was achieved in Condition 3, with a completion time of 52.7 seconds, while the highest recorded time was 73.2 seconds in Condition 2. Despite this variation, the overall task was completed successfully in all conditions.
3. Performance Summary:
   - Average completion time: 62.1 seconds.
   - Standard deviation: Approximately 6.4 seconds.
   - Success rate: 100%.
4. Key Observations:
   - The system's performance was reliable, with no task failures.
   - Variations in time were influenced by object positions and robot path adjustments.
   - The robot demonstrated consistent repeatability, successfully handling all test conditions.

Overall, the system performed accurately and reliably, meeting the project objectives with consistent results.

Observations:

1. **Performance Consistency:**
   - The robot successfully completed all trials, achieving a 100% success rate.
   - This demonstrates the robustness of the system's object detection and navigation algorithms.
2. **Best Performance:**
   - Trial 6 (Red and Blue configuration) was identified as the best trial, with the shortest time of 60.2 seconds.
   - The robot's trajectory and execution during this trial were optimized for minimal delays.
3. **Execution Times:**

- The fastest time was 52.7 seconds in Trial 3 (Blue and Green configuration).
- The longest time was 73.2 seconds in Trial 2 (Green and Red configuration).
- Variations in time can be attributed to differences in object placement, color detection efficiency, or path planning adjustments.

4. **Color Configuration Impact:**
    - Trials involving the Green object as "Top" (Trials 1 and 2) showed slightly higher execution times compared to other configurations.
    - Red and Blue combinations (Trial 6) were particularly efficient, possibly due to reduced navigation complexity or improved detection speed.

Insights:

- **Algorithm Efficiency:** The system displayed high consistency across all trials, successfully navigating and transporting objects to their respective zones.
- **Optimization Potential:**
    - Further improvements can be made by analysing path deviations or optimizing the trajectory planning algorithm.
    - Time variations suggest room for enhancement in object recognition and dynamic decision-making.

Student Feedback Summary

Challenges and Difficulties Encountered:

During the completion of this project, several challenges were faced that required careful troubleshooting and iterative improvements. Below is a detailed explanation of the difficulties encountered:

1. Object Color Detection Accuracy

- **Issue**: The robot's vision system sometimes struggled to accurately detect the object's color, particularly when lighting conditions or object placement varied.
- **Cause:**
    - Variations in ambient lighting led to inconsistent color thresholds in the HSV space.
    - Shadows and reflections interfered with segmentation.
- **Solution:**
    - Adjusted the HSV color thresholds dynamically based on real-time image feedback.
    - Applied pre-processing techniques like Gaussian filtering to reduce noise and improve segmentation accuracy.

2. Path Planning and Deviations:

- **Issue:** The robot occasionally deviated from its intended straight-line trajectory, especially when navigating between objects and returning to the origin.
- **Cause:**
    - Inaccurate position estimation due to sensor noise or accumulated errors.
    - Delays in feedback from the PD controller caused slight overshoots.
- **Solution:**
    - Fine-tuned the PD controller gains (K_p and K_d) to improve trajectory corrections.
    - Introduced a real-time position correction mechanism using simulated feedback sensors.

3. **Object Pickup and Transportation**

- **Issue:** Simulating the robotic arm mechanism for object pickup was challenging, as it required precise alignment between the robot and the object's position.
- **Cause:**
    - Small positional errors resulted in misalignment during object pickup.
    - Grasping simulations required accurate input for object placement.
- **Solution:**
    - Implemented a position tolerance margin to ensure the robot could still "grip" the object even with minor alignment errors.
    - Refined object detection to provide better centroid accuracy for positioning.

4. Time Optimization

- **Issue:** Variations in execution time between trials (52.7s in Trial 3 vs. 73.2s in Trial 2) indicated inefficiencies in certain configurations.
- **Cause:**
    - Suboptimal navigation paths due to sequential task execution.
    - Excessive time spent fine-tuning movements during object drop-off.
- **Solution:**
    - Optimized the path-planning algorithm to reduce unnecessary robot movement.
    - Implemented multi-task logic to minimize idle time during object handling.

5. Debugging and Testing

- **Issue:** Identifying and correcting errors during debugging was time-consuming, especially in multi-step simulations.
- **Cause:**
    - The need for repeated trial-and-error to test the robot's responses under various initial conditions.
- **Solution:**
    - Developed a systematic testing procedure to validate each component (color detection, path navigation) independently before integrating the full system.

Key Learnings:

Despite these challenges, the project provided valuable insights into:

- The importance of sensor accuracy and dynamic parameter tuning in robotics.
- Developing robust algorithms to handle real-world uncertainties.
- Systematically troubleshooting and improving system performance through iterative testing.

Thoughts on the Project

This project was really tough but also super rewarding. It gave me a great chance to use what we have learned about robotics, computer vision, and path planning on a real-world problem. I was able to combine different subsystems—like object detection, navigation, control algorithms, and task execution—to create a complete and working solution.

Important Ideas Educational Journey:

- We learned a lot about color-based vision systems and control mechanisms through this project. Working on real-time trajectory issues helped me gain important experience in making robots perform better.
- Dealing with sensor noise and inaccuracies involved a lot of testing and helped me learn about better
  Error-handling strategies.

Key Features of the Project:

It felt really great to finish all six trials with a perfect score!
Watching the robot finish tasks so efficiently, particularly in the best trial at 60.2 seconds, really highlighted how effective the iterative improvements were.

Challenges Faced:

- Finding the right balance between precision and speed in the path-planning logic took a lot of careful adjustments.
- At first, it was tough to get accurate object detection in different conditions, but we figured it out using dynamic thresholding techniques.

Duration:

- The project took around 45 to 50 hours to finish, broken down like this:
- System design and setup will take about 10 hours.
- Identifying the necessary requirements, roles, and configurations for the problem.
  Color Detection Implementation: 12 hours.

Concluding Thoughts:

This project provided a rich and rewarding experience in robotic systems, offering a hands-on opportunity to explore the integration of various subsystems, such as perception, motion planning, and control algorithms. From algorithm design to rigorous testing and optimization, we encountered real-world challenges that deepened our understanding of robotics engineering.

Key aspects that stood out during the project were:

Systematic Troubleshooting:

The iterative debugging process was invaluable in identifying and resolving system inefficiencies. This included fine-tuning the color detection thresholds, optimizing path planning, and improving motor control to enhance overall performance.

Integration of Subsystems:

Successfully combining vision-based object detection, FSM-based decision-making, and motor control highlighted the complexity and importance of interdisciplinary collaboration in robotics.

Practical Insights into Real-World Applications:

The project simulated constraints often faced in real-world scenarios, such as limited hardware capabilities and environmental uncertainties. Overcoming these challenges strengthened our problem-solving skills and prepared us for future work in similar contexts.

Skill Development:

We honed essential skills in algorithm design, control system development, and sensor-based navigation. These are foundational to robotics engineering and can be applied to broader applications like automation and intelligent systems.

Would you recommend this project for the future?

Absolutely, we believe this project is an excellent choice for students in the future. It provides hands-on experience with critical robotics concepts such as computer vision, path planning, and control systems, fostering essential problem-solving and debugging skills.

Moreover, the project offers a realistic simulation of challenges faced in autonomous robotics, such as collision avoidance and precision navigation. Future students can build on this foundation by incorporating advanced features like dynamic obstacle detection, adaptive path optimization, and machine learning techniques for enhanced object recognition.

The interdisciplinary nature of this project also makes it highly engaging, as it brings together theoretical knowledge and practical application, making it both educational and rewarding.

Discussion

The project successfully demonstrated the integration of computer vision, path planning, and control systems to achieve a robotic task involving object detection, pickup, and transportation. The robot consistently identified objects of specific colors, transported them to their designated zones, and returned to the origin without collisions.

1. System Performance:

The robot completed all six trials with a 100% success rate, showing the robustness of the implemented algorithms. The fastest trial (60.2 seconds) was observed when the initial condition involved the **red and blue objects**, indicating that path optimization and object positioning influenced performance.

2. Color Detection Accuracy:

The HSV color segmentation approach proved effective in identifying the objects, although adjustments were needed for dynamic lighting conditions. Filtering techniques, such as Gaussian smoothing, reduced noise and improved detection accuracy.

3. Path Planning and Control:

The PD controller played a crucial role in maintaining smooth trajectories, but slight deviations were observed in early trials due to sensor noise. Fine-tuning the controller gains significantly improved the robot's path stability.

4. Challenges and Improvements:

- Challenges included variations in object alignment during pickup and delays in navigation feedback. These were mitigated by introducing position tolerance margins and refining the control system.
- Future improvements could focus on implementing *advanced path optimization algorithms* (e.g., A*) and integrating real-time *obstacle detection* for dynamic environments.

5. Time Analysis:

The variation in time across trials highlights the importance of path efficiency and object placement. Optimized navigation paths can further reduce execution time in future iterations.

Table 2 - Observation of all 6 conditions.

Condition	Top Color	Bottom Color	Time (seconds)	Observations
1	Green	Blue	61.5	Smooth execution with minor trajectory corrections. Color detection was reliable.
2	Green	Red	73.2	Took the longest due to minor delays in path planning and object alignment.

3	Blue	Green	52.7	Took the longest due to minor delays in path planning and object alignment.
4	Blue	Red	63.0	Stable performance with efficient navigation, but small overshoots were corrected.
5	Red	Green	62.1	Good performance; color detection accuracy was consistent with minimal delays.
6	Red	Blue	60.2	Marked as the best trial due to a combination of smooth navigation and minimal execution time.

Observations and Trends:

- ***Best Performance (Condition 6):*** The robot achieved its fastest and smoothest performance in Condition 6 (Red and Blue), completing the task in *60.2 seconds*. This result highlights the efficiency of the path taken and accurate object handling.
- ***Longest Execution (Condition 2):*** Condition 2 took *73.2 seconds* due to delays caused by minor misalignment during object pickup and longer adjustments in path planning.
- ***Impact of Object Placement:*** Conditions where objects were positioned closer to the robot's origin or required minimal directional changes (like Condition 3) resulted in faster completion times.

3. Challenges Identified

- ***Path Optimization:*** Small trajectory deviations in certain conditions increased the execution time slightly. This was most noticeable in Conditions 2 and 4.
- ***Color Detection:*** The system relied on HSV thresholding, which occasionally required adjustments for lighting variations, particularly when detecting similar shades (e.g., Green and Blue).
- ***Pickup Alignment:*** Minor misalignment during the object pickup phase in early trials caused slight delays but was resolved with refined position correction logic.

❖ Youtube Hyperlink: ME 598A Fall 2024 Final Project, Group R2

Conclusion

The project achieved its goals of independently detecting, picking up, and moving objects using color recognition, while also avoiding collisions and returning to the starting point. The robot showed consistent performance across six different trial conditions, each with unique object placements and colors, highlighting how well the algorithms work. The trial where the robot sorted red and blue objects showed
the best performance, finishing the task in 60.2 seconds.

The color detection system that uses HSV thresholds worked well, but I had to make some small adjustments to deal with changes in lighting and slight misalignments in how the objects were positioned. The robot used a PD controller for navigation, which helped with smooth trajectory control, but there were some trials that required minor corrections. We fixed these issues by adjusting the controller parameters, which made the robot work better overall.

The system did a good job in all the tests, but the differences in execution times showed some spots where we can do better, especially in path planning and object alignment. This project lays a good groundwork for what comes next, opening up chances to use more advanced algorithms for optimizing paths and avoiding obstacles. Also, future improvements might boost how well the system works in changing and real-life situations.

In conclusion, the project offered a thorough learning experience in robotics, blending theory with hands-on application. Completing all the trials shows how strong the system is, and the insights we gained will help us make future improvements and innovations in autonomous robotics.

References

[1] Gonzalez, R. C., & Woods, R. E. (2018). Digital Image Processing (4th ed.). Pearson: Referenced for theoretical understanding of HSV color space and its application in color segmentation.

[2] Corke, P. (2017). Robotics, Vision and Control: Fundamental Algorithms in MATLAB® (2nd ed.). Springer: Referenced for techniques in visual perception and blob analysis in robotic systems.

[3] Mechanical Engineering Department, ME 598A. Lab 3: Image Processing for Vision-Based Tasks: Provided detailed guidance on task requirements, code structure, and objectives for imagebased cone detection.

[4] MathWorks. (n.d.). Using the Color Thresholder App. Retrieved from: https://www.mathworks.com/videos/webcam-support-89504.html

Appendix:

Appendix A: Final algorithm for seeking objects based on color, determining arena zone locations and push objects to their designated zone ensuring to avoid wall collision and returning back to the arena origin.

```
function [leftMotor, rightMotor, currentState] = sortObjects(Red_Distance, Green_Distance, Blue_Distance, Left_Distance, Front_Distance, Right_Distance, xPose, yPose, tPose180, initialState)

currentState = initialState;
% initialState == 0

% Functions to define motor commands for turning left & right.
    function [leftMotor, rightMotor] = rotateCW()
        leftMotor = 75;
        rightMotor = -70;
    end

    function [leftMotor, rightMotor] = rotateCCW()
        leftMotor = -70;
        rightMotor = 75;
    end

% Function to calculate the angle to target coordinates.
    function angle = computeTargetAngle(xStart, yStart, xTarget, yTarget)
        angle = atan2d(yTarget - yStart, xTarget - xStart);
    end

% Function to identify the color of the object.
    function detectedColor = identifyObjectColor()
        if ~isnan(Blue_Distance) && isnan(Red_Distance) && isnan(Green_Distance)
            detectedColor = 'blue';
        elseif ~isnan(Green_Distance) && isnan(Red_Distance) && isnan(Blue_Distance)
            detectedColor = 'green';
        elseif ~isnan(Red_Distance) && isnan(Green_Distance) && isnan(Blue_Distance)
            detectedColor = 'red';
```

```
        else
            detectedColor = 'unknown';
        end
    end

% Function to push the object to its colored zone.
    function [leftMotor, rightMotor] = colorSort(color, nextState)
        switch color
            case 'red'
                if xPose > 1.7 && yPose > 1.7
                    leftMotor = 0;
                    rightMotor = 0;
                    currentState = nextState;
                    return;
                end
                targetAngle = computeTargetAngle(xPose, yPose, 1.85, 2.05);
                [leftMotor, rightMotor] = alignToTarget(targetAngle);
            case 'blue'
                if xPose < -1.7 && yPose < 0.3 && yPose > -0.3
                    leftMotor = 0;
                    rightMotor = 0;
                    currentState = nextState;
                    return;
                end
                targetAngle = computeTargetAngle(xPose, yPose, -2, 0);
                [leftMotor, rightMotor] = alignToTarget(targetAngle);
            case 'green'
                if xPose > 1.7 && yPose < -1.7
                    leftMotor = 0;
                    rightMotor = 0;
                    currentState = nextState;
                    return;
                end
                targetAngle = computeTargetAngle(xPose, yPose, 2.1, -1.95);
                [leftMotor, rightMotor] = alignToTarget(targetAngle);
            otherwise
```

```
            leftMotor = 120;
            rightMotor = 120;
        end
    end

% Function to align the robot to a target angle and move straight.
    function [leftMotor, rightMotor] = alignToTarget(targetAngle)
        angleDifference = tPose180 - targetAngle;
        if angleDifference > 180
            angleDifference = angleDifference - 360;
        elseif angleDifference < -180
            angleDifference = angleDifference + 360;
        end
        if angleDifference < -10
            [leftMotor, rightMotor] = rotateCCW();
        elseif angleDifference > 10
            [leftMotor, rightMotor] = rotateCW();
        else
            leftMotor = 120;
            rightMotor = 120;
        end
    end

% Function to reposition the robot to a specific coordinate.
    function [leftMotor, rightMotor] = repositionTo(xTarget, yTarget, nextState)
        if xPose < xTarget + 0.1 && xPose > xTarget - 0.1 && yPose > yTarget - 0.1 && yPose < yTarget
+ 0.1
            leftMotor = 0;
            rightMotor = 0;
            currentState = nextState;
            return;
        end
        targetAngle = computeTargetAngle(xPose, yPose, xTarget, yTarget);
        [leftMotor, rightMotor] = alignToTarget(targetAngle);
    end
```

```
% Function to determine the object's proximity to the robot.
    function objectDistance = determineObjectProximity()
        if ~isnan(Red_Distance)
            objectDistance = Red_Distance;
        elseif ~isnan(Green_Distance)
            objectDistance = Green_Distance;
        elseif ~isnan(Blue_Distance)
            objectDistance = Blue_Distance;
        else
            objectDistance = 10;
        end
    end

% Function to reverse the robot slightly to move away from object.
    function [leftMotor, rightMotor] = reverseSlightly(objectDistance, nextState)
        if objectDistance < 0.57
            leftMotor = -120;
            rightMotor = -120;
        else
            leftMotor = 0;
            rightMotor = 0;
            currentState = nextState;
            return;
        end
    end

% Robot control logic.
switch currentState
    case 0
        % Travel to the top object.
        [leftMotor, rightMotor] = repositionTo(0, 1.5, 1);
    case 1
        % Identify object color and bring to correct zone.
        colorDetected = identifyObjectColor();
        [leftMotor, rightMotor] = colorSort(colorDetected, 2);
    case 2
```

```
        %
        objectDistance = determineObjectProximity();
        [leftMotor, rightMotor] = reverseSlightly(objectDistance, 3);
    case 3
        % Travel to the bottom object.
        [leftMotor, rightMotor] = repositionTo(0, -1.5, 4);
    case 4
        % Identify object color and bring to correct zone.
        colorDetected = identifyObjectColor();
        [leftMotor, rightMotor] = colorSort(colorDetected, 5);
    case 5
        %
        objectDistance = determineObjectProximity();
        [leftMotor, rightMotor] = reverseSlightly(objectDistance, 6);
    case 6
        % Return robot to the origin.
        [leftMotor, rightMotor] = repositionTo(0, 0, 7);
    otherwise
        leftMotor = 0;
        rightMotor = 0;
end

% Collision avoidance logic.
if Front_Distance < 0.3
    if isnan(Left_Distance) && isnan(Right_Distance)
        leftMotor = -120;
        rightMotor = -120;
    elseif isnan(Left_Distance)
        [leftMotor, rightMotor] = rotateCCW();
    elseif isnan(Right_Distance)
        [leftMotor, rightMotor] = rotateCW();
    end
elseif Right_Distance < 0.3 && isnan(Left_Distance)
    [leftMotor, rightMotor] = rotateCCW();
elseif Left_Distance < 0.3 && isnan(Right_Distance)
    [leftMotor, rightMotor] = rotateCW();
```

```
end

end
```

Appendix B: Sample Trajectory plotting code

```
plot(xPose,yPose)
title('Overhead XY Plot of Robot Trajectory')
xlabel ('X_I [m]')
ylabel ('Y_I [m]')
```