Seven Segment Display with Robot Formation of Six Robots and Gesture Control

Ansh Mehta, Nigel Fernandes, Srushti Shantanu Raste, Ysatis Tagle

Electrical and Computer Engineering, Boston University

EC545 Cyber-Physical Systems

Professor Wenchao Li

December 12, 2023

Introduction

The goal of this project was to design a robotic system which forms a physical seven segment display using six individual robots. The overall objective was to explore multi-robot coordination and localization. It was determined that this system must be able to detect and control robot movement using camera and user inputs. The robots specifically were to move until they reached a specific position and orientation. To achieve this, the project was broken into multiple facets: user input, camera localization, robot controller and motion planning, and the robot movement. ROS2 was used to communicate between these facets. The robots had sticks mounted onto servos to visualize the segments of the display for testing. After testing, we were able to prove that our system is able to move the robots using an exterior camera localization system to specific points with an accuracy up to 35cm. Proposed improvements of our system include generalizing the camera system and improving robot movement accuracy.

Design Diagram

The system is composed of four major components: user input, camera localization, robot control, and the robots themselves.

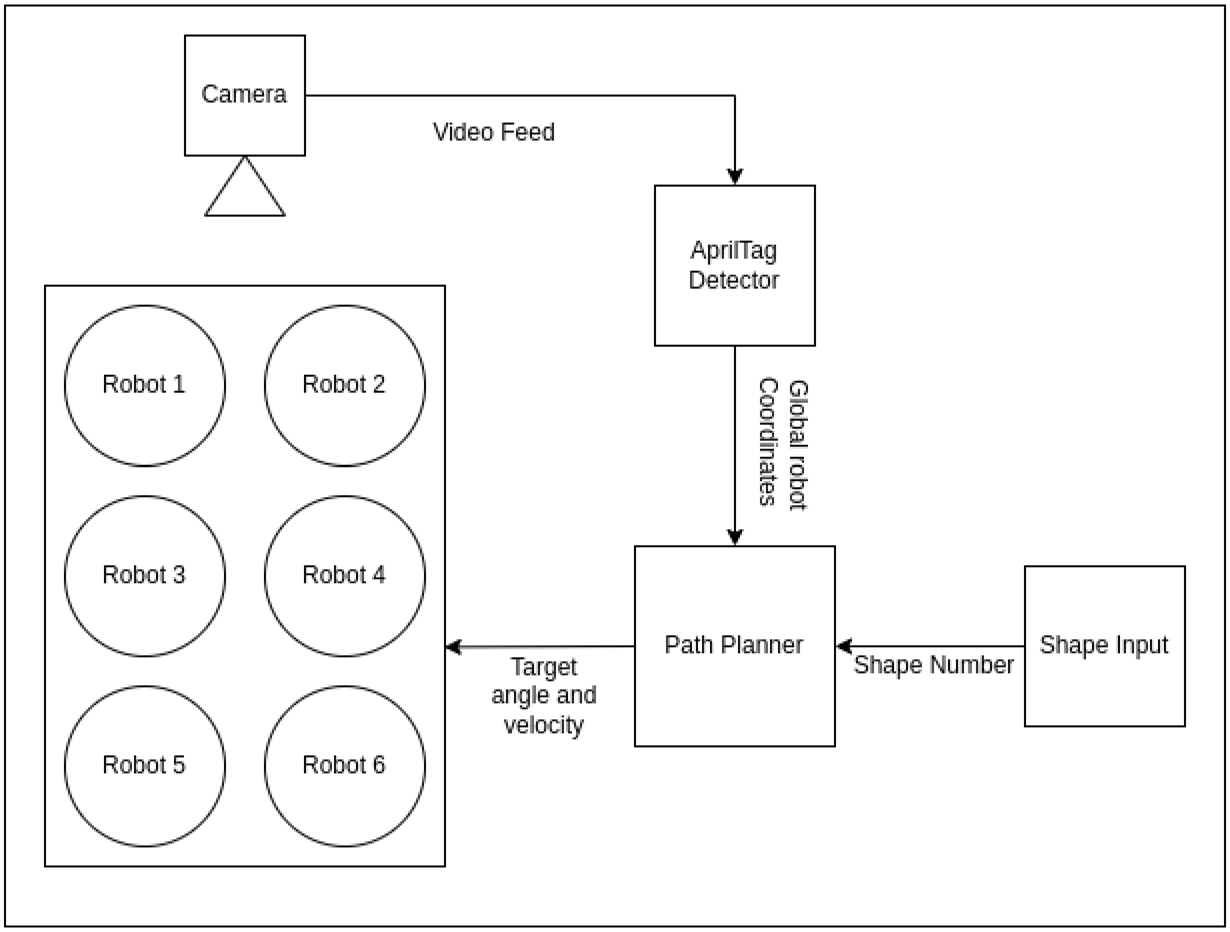


Figure 1. High Level Design Diagram

The camera feeds a continuous stream of data into the AprilTag detection block for camera localization. The output from this block comprises the current coordinates of detected robots, forwarded to the path planner. Utilizing this data in conjunction with user input from the glove interface, the path planner determines individual robot paths, calculating corresponding target angles and velocities. Subsequently, these parameters guide the robots' movements, leading to positional adjustments visible within the video stream. This process establishes a closed-loop control system, constituting a feedback mechanism in robot control. (Figure 1)

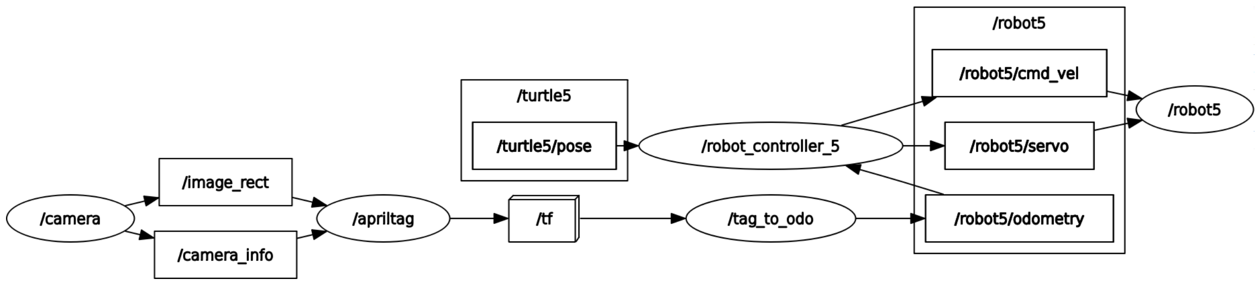


Figure 2. Ros Graph of Topics for Robot 5

The ROS graph shows how the system communicates using ROS2 and MicroROS. (Figure 2) There are mainly five nodes. For each robot, there is an odometry topic, servo topic, and cmd_vel topic. The overall RQT Graph, showing how the various components interact, establishes the parent topic for each robot, which the robot subsequently subscribes to, while the corresponding robot controller publishes to this parent topic. (Figure 3)

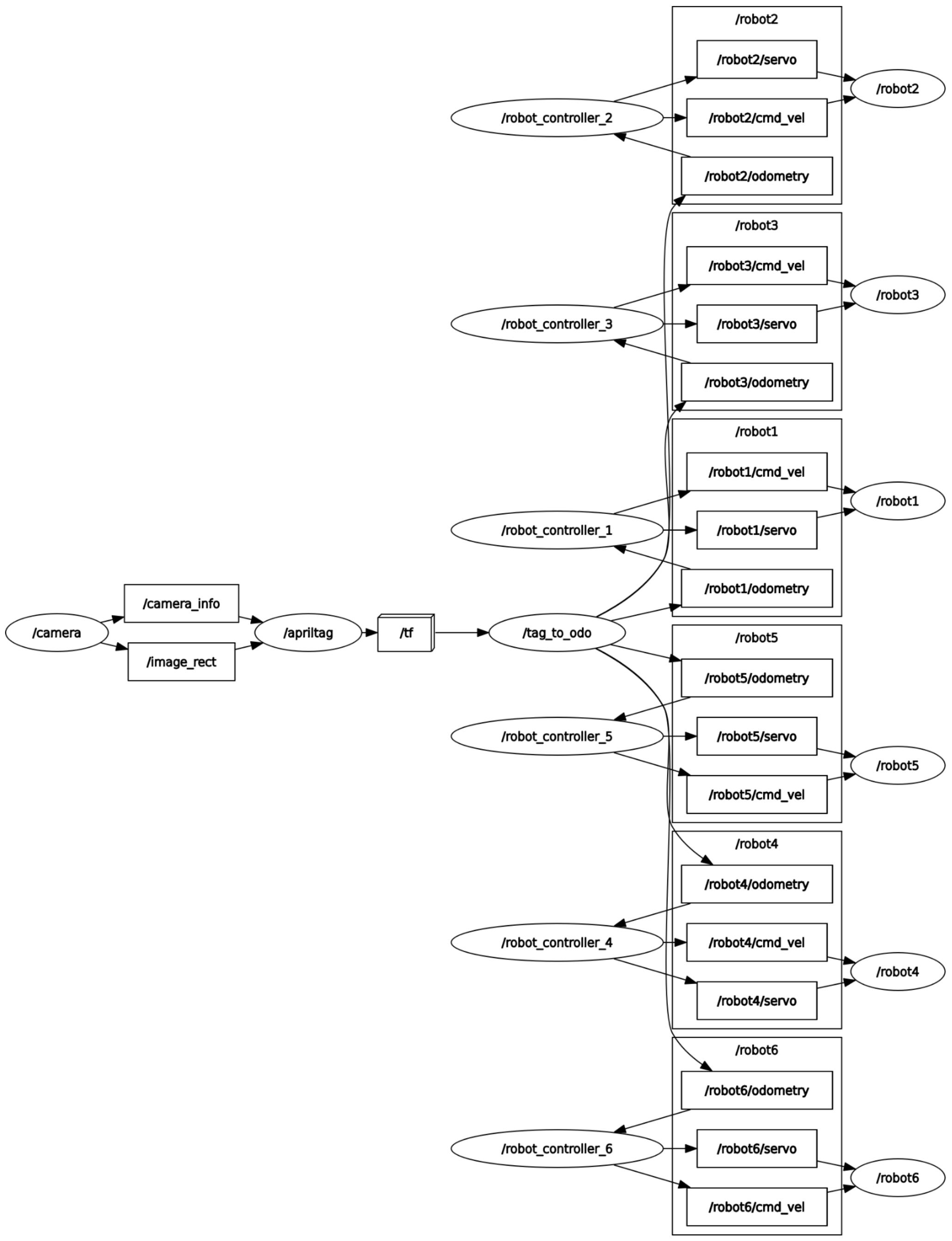


Figure 3.Complete Ros Graph of Topics

Specification / Modeling

The system can be abstracted using the FSM given below (Figure 4):

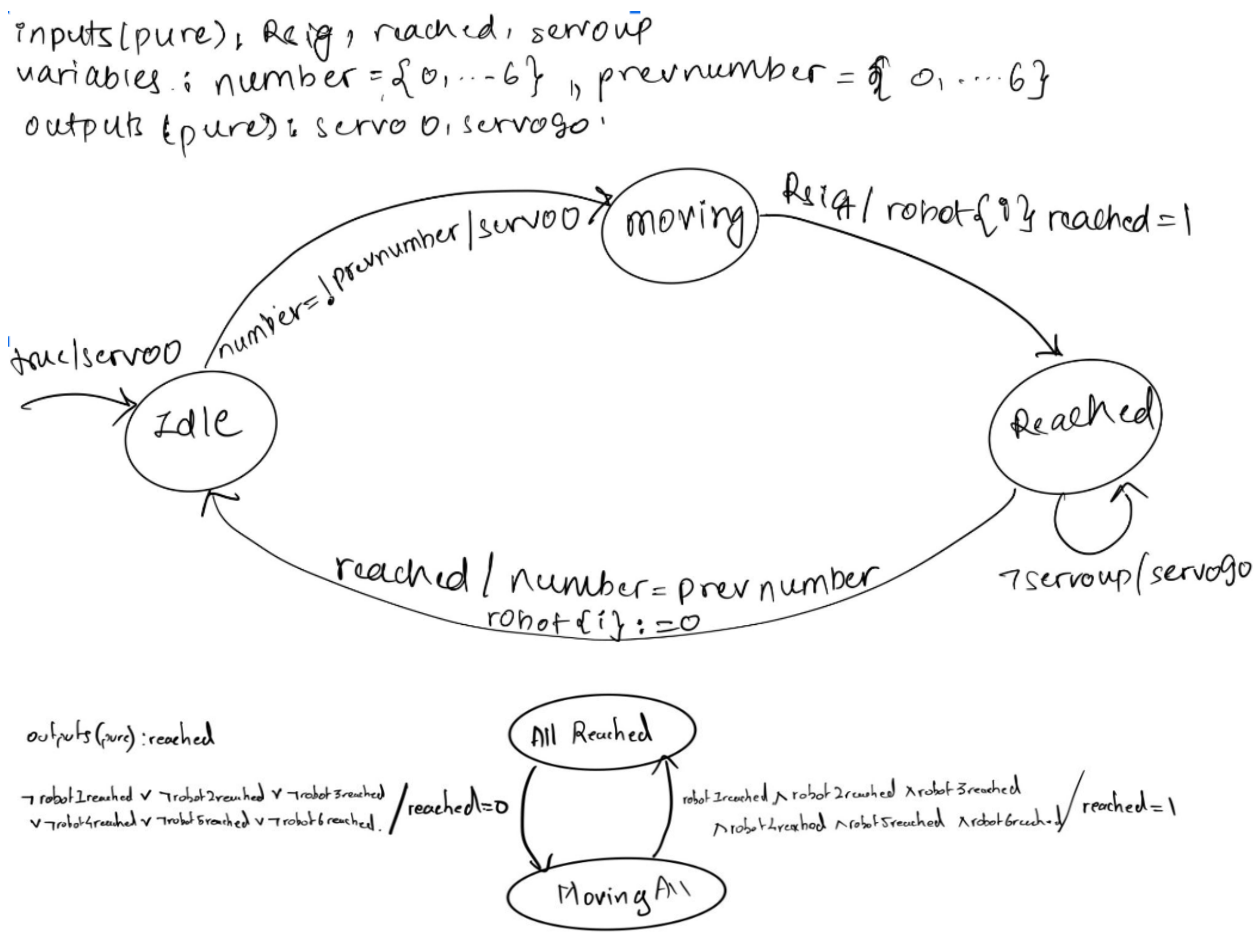

inputs(pure): Rei@, reached, servoup
variables: number = {0,...6}, prevnumber = {0,....6}
outputs (pure): servo 0, servo90
moving
Idle
Reached
true|servo0
number=!prevnumber|servo0
Rei@/ robot{i} reached=1
reached / number=prevnumber robot{i}:=0
¬servoup/servo90
outputs(pure): reached
All Reached
Moving All
¬robot1reached ∨ ¬robot2reached ∨ ¬robot3reached ∨ ¬robot4reached ∨ ¬robot5reached ∨ ¬robot6reached. / reached=0
robot1reached ∧ robot2reached ∧ robot3reached ∧ robot4reached ∧ robot5reached ∧ robot6reached / reached=1

Figure 4. FSM of System Diagram

To describe the behavior of the system, the following LTL can be used:
G((input.number=> state.moving && servo0) => F(X(reached=>servo90)).

Software/Hardware

In this section we will go more into detail on the implementations within each component of the system.

Camera Localization

This system is based on the use of AprilTags in order to identify and track robots. Compared to other fiducial markers such as QR Codes, AprilTags were chosen due to their flexibility in regards to camera performance, integration in ROS2, and because it already outputs the data needed for the system to work. [2] From the available tags, the 36h11 was chosen since it has the smallest tag width therefore faster to process in the camera frame.

The hardware used for the camera-based localization includes a Raspberry Pi 3B+ and an Intel Realsense 435. The camera and raspberry pi are mounted onto a pole that elevates and stabilizes the camera position such that the camera view can see all the tags from overhead. (Figure 4) The Raspberry Pi runs Ubuntu Server 22.04 on arm64 architecture. This required the Intel Realsense SDK, which contains drivers for the camera, to be built from source. The builds were configured such that they are text only programs, therefore reserving more memory and space for runtime. The apriltag packages (apriltag_ros and apriltag_msgs) were installed from the distro package.

For communication in ROS2, there were three nodes used for camera localization. There is the realsense node, the AprilTag node, and the *tagToOdo* node. The Intel Realsense SDK is supported by ROS2 and by default outputs topics such as camera_info and image_raw. These are both subscribed to by the AprilTag node. From user configuration files, the AprilTag detects each tag within the frame and outputs where it is relative to the camera frame. This is published as /tf, which is a list of detected tag's transforms. The *tagToOdo* node then takes the tf data and remaps it such that the actual message type is of odometry rather than transforms.

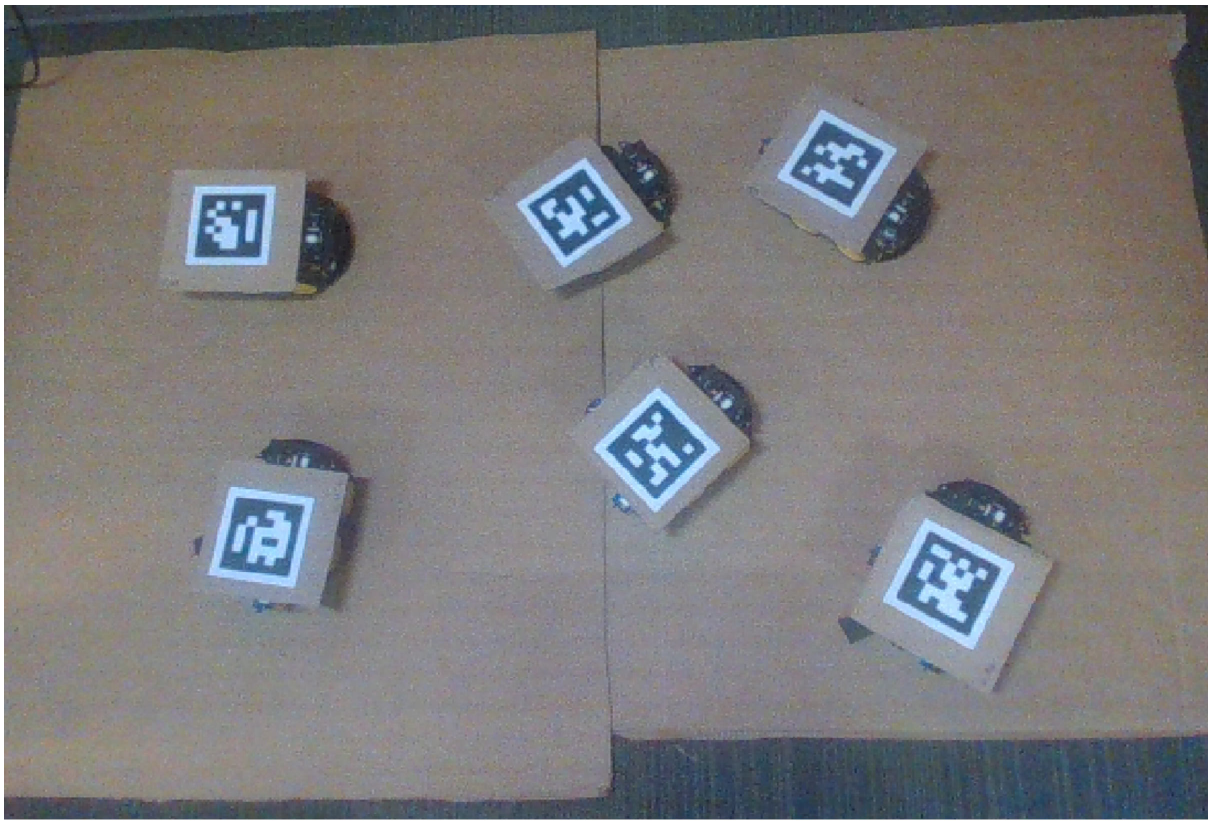


Figure 5. Camera Perspective of the Tags

Robot Controller

The robot controller was developed in Python, and included multiple nodes which published and subscribed to data to and from the rest of the nodes (The robot and the camera localization).

The controller was designed and tested using turtlesim, which is a lightweight simulator for ROS2. Turtlesim proved to be a good platform for testing the planner, and helping verify the system, since the kinematic of the turtles were the same as the physical robot, which was a differential drive.

Subscriptions:

1. turtle{self.robot_id}/pose
2. robot{self.robot_id}/odometry

Publishers:

1. turtle{self.robot_id}/cmd_vel
2. robot{self.robot_id}/cmd_vel
3. robot{self.robot_id}/final_waypoint
4. robot{self.robot_id}/debugger
5. robot{self.robot_id}/servo

A way to implement waypoint following was integrated into the robot controller code, which helped in defining constraints to paths. Two approaches were tested, one using splines, the other using lines. The approaches were tested for accuracy and temporal performance. While splines provided better control, space constraints forced the implementation of a line path, which makes the robots rotate intermittently to head to the target to the next point in the path.
However, both approaches published data to the robot in similar ways, with the points along the path being the differentiating factor. Once a path is calculated, a query for whether the robot has reached its target orientation is checked. If this query is satisfied, the robot sends out a linear velocity, predetermined to be 40.0 units. However, if the query is not satisfied, the robot sends the amount of angle to rotate, relative to the current orientation, to the robot over the cmd_vel topic. This approach, while unconventional, proved to work better than sending raw values of angular velocity to the robot. The reasons for this are discussed in the technical challenges section.
As soon as a message is received from the odometry topic of a robot, the controller updates the pose of the corresponding robot, and proceeds to calculate the path from the current position to the last waypoint in the waypoints array, making sure to include each waypoint in the path.
This goes on until the robot has reached its final waypoint, at which point a query to check the final orientation of the robot with the intended orientation specified by the waypoint is run. This query helps the robot to keep changing its position until it reaches the waypoint.

Robot

The robot's chassis and main components are sourced from the Elegoo Smart Robot Car Kit V4.0 (Figure 5). The hardware used includes a ESP32 running MicroROS, an IMU, and an arduino to drive the on-board servos. The robots have an Apriltag, printed and secured to a cardboard mounted on the top; unique to each robot for identification, localization, and orientation. The robots also have two servos, where one is mounted on the front and the other on the side, with sticks attached to them to represent the segment lengths.

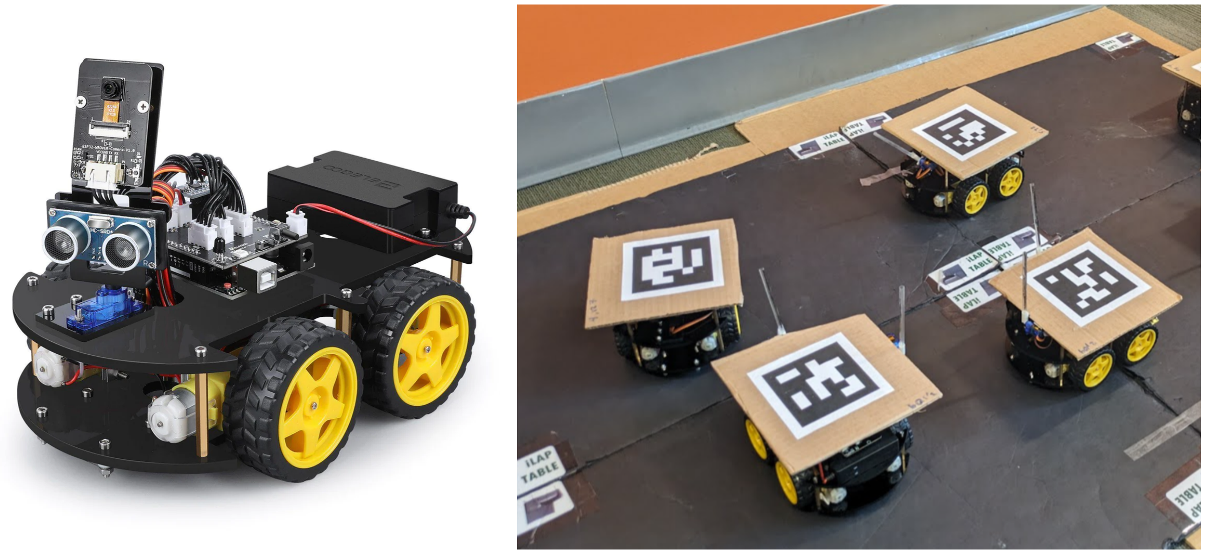


Figure 6. Image of Robot used During Project (Left) and Robots Deployed(Right)

ESP32

The main role of the ESP32 modules was to act as a bridge between ROS2 and the arduino. The ESP32s were programmed for microROS and subscribed to the command topics generated by the planner. Then, they would encapsulate the received data into a JSON string and pass it to the Arduino via Serial communication. JSON strings were chosen because they are more readable and easier to parse than other methods of sending data. The ESP32 also ensures the robots stop if their connection to ROS is lost.

Arduino

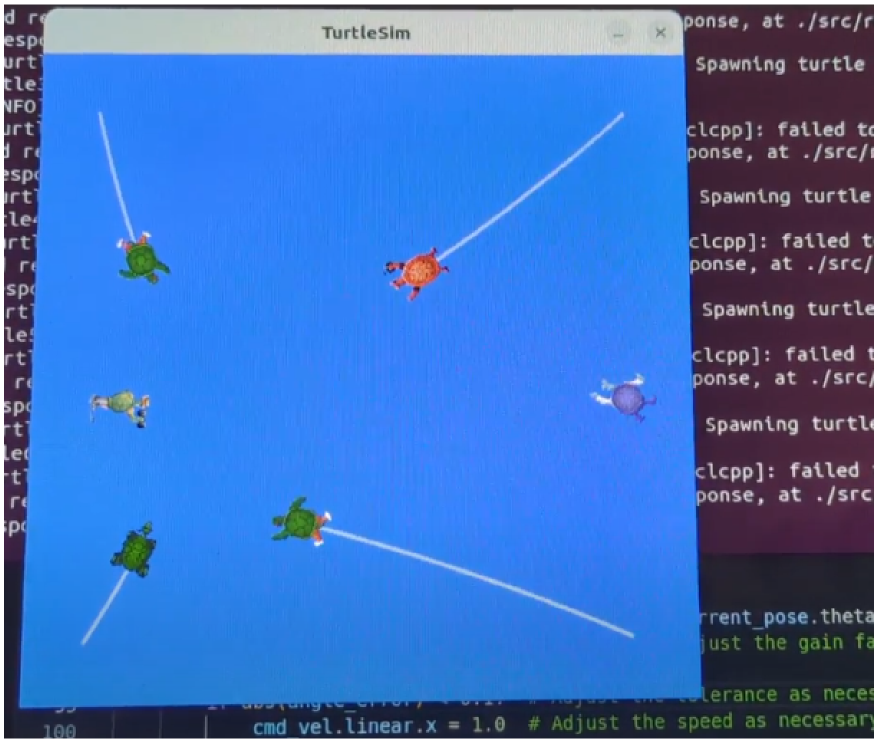
There were three libraries used on the Arduino: ArduinoJSON, Servo, and MPU6050_light. ArduinoJson allows the Arduino to read and parse the received JSON string from the ESP32. Once the data is extracted, the Arduino maps the velocity data into PWM (Pulse Width Modulation) signals. These PWM signals are then used to control motors, adjusting their speed or position based on the received velocity data. There are two servos mounted on the robot, which are controlled with the Servo library. The servo motors are controlled by the Arduino using the servo value obtained from the ESP32. If the servo value is zero, both servos are set to zero degrees. When the value is two, one of them moves to ninety, and when the value is three, the other goes to ninety. When servovalue equals four, both servos rotate ninety degrees. These servos help extend the 'segments' of the segment display.

The MPU6050_light library is then used to get the current angle of the robot from the IMU. This is fed into the PID controller, which maintains the robot's orientation and keeps the robot going straight. The PID controller was tuned using the Ziegler Nichols method [13]. The way this method works is that initially the Kp, Ki, and Kd values are set to zero. Then, the Kp value is increased until the oscillations are damped. Then, the Kd value is increased until overshoot is minimized. Ki remains set at zero. From this method, it was determined that the optimal values Kp = 10, Ki = 0 and Kd =2.5.

Analysis

Turtlesim and UPPAAL were used to perform the analysis of the system, as presented in the specification.

While UPPAAL was used to verify properties which may result in unfavorable states, turtlesim was able to verify the paths taken by the path planner. We were able to verify that the planner was able to command the robots to their goals effectively in turtlesim.



```
ponse, at ./src/r
 Spawning turtle

clcpp]: failed to
ponse, at ./src/r

 Spawning turtle

clcpp]: failed t
ponse, at ./src/

 Spawning turtle

clcpp]: failed t
ponse, at ./src/

 Spawning turtl

clcpp]: failed
ponse, at ./src
```

```
rrent_pose.theta
just the gain fa
erance as neces
    cmd_vel.linear.x = 1.0  # Adjust the speed as necessary
```

Figure 7. Screenshot of TurtleSim

The physical system was tested using a setup represented in Figure 8.

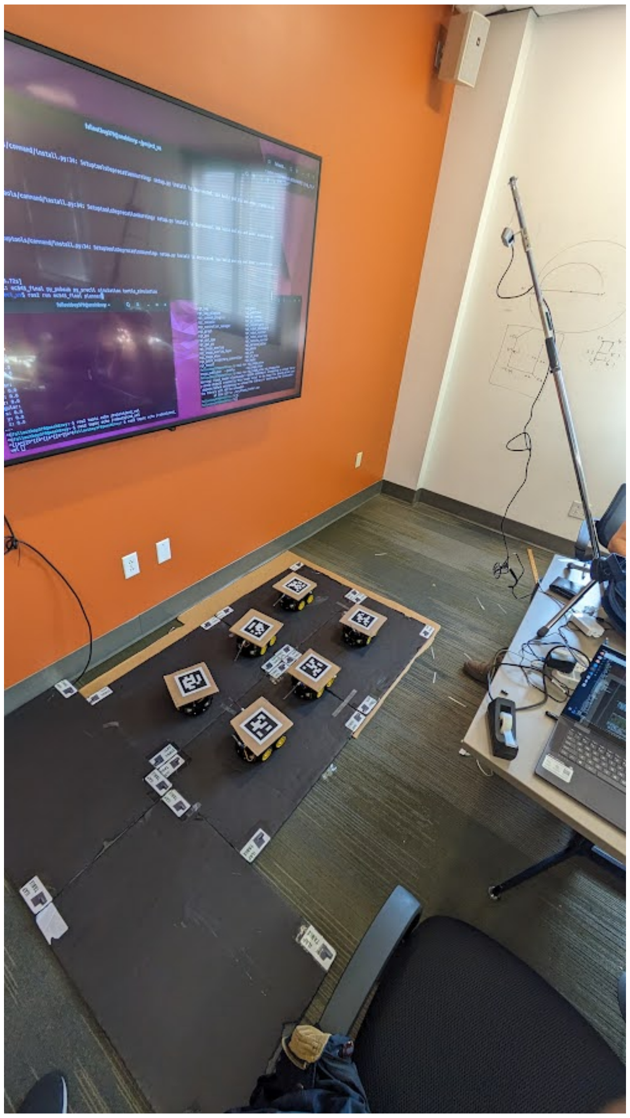


Figure 8. Physical Testing of System

Using UPAAL, we verified the following properties:

1. The servos are never actuated when the robots have not reached their destination.
2. The robots are moving when a change in number is detected.
3. The entire system specification is also satisfied.

Technical Challenges

One of the key technical issues was dealing with the camera mounting. Originally, the height and angle were calculated from the camera field of view to be twelve degrees at five feet, however, in practice it was found to skew the AprilTag detection such that the X and Y coordinates weren't accurate. To fix this, the camera was mounted overhead of the robot area, but it wasn't designed therefore there aren't accurate numbers in order to reproduce the tests.

Another technical issue found on the hardware side was configuring the ESP32 to effectively run ROS2 nodes due to the proprietary nature of the ESP32 module. The board definitions, provided by the manufacturer to program the module through Arduino IDE, do not support microROS. After some trial and error we were able to find a board definition compatible with both microROS and the module. Moreover, the setup of the power supply to the module required it to be connected to the battery for power while programming it, adding complications while troubleshooting the issue.

A bottleneck in the design was induced due to the use of a raspberry pi to interface with the camera. The location of the camera made it difficult to use a full fledged computer with it so we used a Raspberry Pi to capture the video feed and publish the images on a topic. This induced a higher latency than anticipated due to the limited processing power of the Pi.

Division of Work

- Ansh: Motion planning, simulation
- Nigel: ROS communication with hardware and glove setup.
- Srushti: Arduino to ROS on robot, specifications and analysis
- Ysatis: Camera localization

Conclusion

In summary, this project aimed to construct a physical seven-segment display through the orchestrated movement of six individual robots, focusing on the exploration of multi-robot coordination and localization. The system was engineered to detect and govern robot movements through camera inputs and user directives, with precise positioning and orientation as key objectives. Divided into distinct components encompassing user input, camera-based localization, robot control, motion planning, and actual robot movement, the project leveraged ROS2 as a communication framework. The robots, equipped with servo-mounted sticks to simulate display segments, underwent rigorous testing, culminating in the validation of their ability to maneuver accurately to predefined points utilizing an external camera-based localization system. Recommendations for system enhancements encompass refining the camera system's versatility and augmenting robot movement precision to further optimize its performance.

References

[1] Bahceci, E., Soysal, O. and Sahin, E., 2003. A review: Pattern formation and adaptation in multi-robot systems. *Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-03-43*.
[2] Olson, E. (2011). *AprilTag: A robust and flexible visual fiducial system*. [online] IEEE Xplore. doi:https://doi.org/10.1109/ICRA.2011.5979561.
[3] Ma, Z., Zhu, L., Wang, P. and Zhao, Y. (2019). *ROS-Based Multi-Robot System Simulator*. [online] IEEE Xplore. doi:https://doi.org/10.1109/CAC48633.2019.8996843.
[4] Allu, N. and Toding, A. (2020). Tuning with Ziegler Nichols Method for Design PID Controller At Rotate Speed DC Motor. *IOP Conference Series: Materials Science and Engineering*, 846, p.012046. doi:https://doi.org/10.1088/1757-899x/846/1/012046.
[5] Mai, C.-Y. and Lian, F. (2006). Analysis of Formation Control and Communication Pattern in Multi-Robot Systems. doi:https://doi.org/10.1109/sice.2006.315713.
[6] Alonso-Mora, J., Breitenmoser, Rufli, Siegwart, R. and Beardsley (2011). Multi-robot system for artistic pattern formation. doi:https://doi.org/10.1109/icra.2011.5980269.
[7] Arcos, L., Cristian Calala, Maldonado, D. and Cruz, P.J. (2020). rosbased Experimental Testbed for Multi-Robot Formation Control. doi:https://doi.org/10.1109/andescon50619.2020.9272073.
[8] Yu, X., 2020. Localization for autonomous construction vehicles using monocular camera and AprilTag.
[9]Kaminka, G.A. and Glick, R. (n.d.). Towards robust multi-robot formations. *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* doi:https://doi.org/10.1109/robot.2006.1641773.'
[10] Zhao, B., Li, Z., Jiang, J. and Zhao, X. (2020). Relative Localization for UAVs Based on April-Tags. doi:https://doi.org/10.1109/ccdc49329.2020.9164563.
[11] Bareis, A., Bareis, M. and Bettstetter, C. (2019). Robots that Sync and Swarm: A Proof of Concept in ros22. *arXiv (Cornell University)*. doi:https://doi.org/10.1109/mrs.2019.8901095.
[12] P. Nguyen, "MICRO-ros2FOR MOBILE ROBOTICS SYSTEMS," Dissertation, 2022.
[13] Hang, C.C., Åström, K.J. and Ho, W.K., 1991, March. Refinements of the Ziegler–Nichols tuning formula. In IEE Proceedings D (Control Theory and Applications) (Vol. 138, No. 2, pp. 111-118). IET Digital Library.

Appendix

Github

https://github.com/FalloutBoy379/EC545_FormationControl

Video

https://drive.google.com/drive/folders/18b6wNEwKlHxctcgNDkdGPFzD2laP3Po9?usp=drive_li

nk