EMBEDDED AI

ECE 5740

Robot Control Using Hand Gesture Recognition Model

Shaibal Saha
Yunge Li
Fan Li
Lucas Alves

Submitted To: Darrin M. Hanna, Ph.D

June 5, 2025

1. Introduction

In this project we will present a robot remotely controlled by hand gestures, with fuzzy and linear commands to achieve that task. Our main motivation is to achieve facilitation and security in possible dangerous and difficult industrial activities that require the use of a vehicle to be operated. Like, for example, driving small forklifts in hot warehouses or using machines in tight indoor spaces, or also when a small ground surveillance vehicle is needed for some dangerous situation. But not only this, a robot with this type of feature can have have a great impact for hazardous material handling tasks, where the user can be remotely operating the robot to do the material manipulation, avoiding accidents with humans.

Along with that, our main motivation comes from the idea to provide a lightweight AI-based remote control vehicle without a special controller device to input movement to the robot, along with fuzzy and linear commands based on hand angles, and keypoint detection. To achieve this, our robot supports two types of hand inputs, via real-time camera from a client computer that sends the video feed so the robot can process the commands, and a pre-recorded video mode, where the commands are already recorded to the robot as a video. In this track, we found this work by Singh et al. [3], that is related to the field of robots for hazardous environments, but controlled by web connection and focused on hazard detection with sensors, such as rain, gas, and flame. Or another very interesting work we found is the Libina et al. [1], where they developed a surveillance robot controlled by voice commands, and counting with some sensors to help on the task for night activities. And for this one, having a live hand controlling could be a good idea, because it is silent, making it even more secure for very dangerous situations, and can be pre-recorded as our project works. This way, we noticed that the field is lacking on research of new and different methods to control the robots in efficient way, possibly avoiding the physical controllers, like smartphones or proper wireless controller devices.

2. System Design

When developing the system, we considered some key aspects related to the systems we were about to use, targeting maximum efficiency, and accuracy to transform the commands to actions itself.

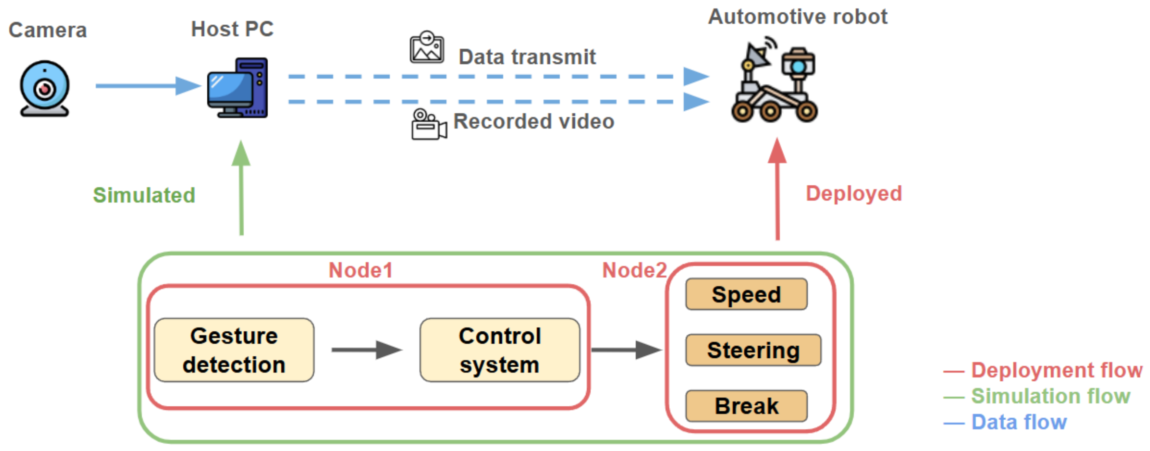


Figure 1: System overview

2.1. Overview

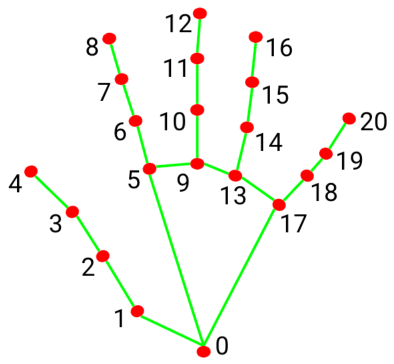
For this project, we simulated an environment to validate the efficiency of the commands, and processing time for the robot, and also implemented it to properly work on the robot, making it move based on the inputs. So our workflow basically works like this:

When running the simulation, the camera on the client PC sends the live video feed to the computing machine, the computing machine will process the image detecting the keypoints and angles, generating the input, then the simulation is processed. If the mode is live video, the simulation will work simultaneously to the commands, reacting immediately, if the input is a pre-recorded video, it will process the video first, then show the path of the simulation.

When we use the robot in real world, the workflow is basically the same. The live feed, or video are sent to the robot, it process the input and take action. However, in the robot, is now a little different, as the robot uses the ROS system, we built two nodes to make this control work, first node takes care of the Gesture Detection, and Control System. And the second node is responsible for the Speed, Steering and Break.

2.2. Hand Gesture Detection

When designing our hand gesture detection module, we utilized MediaPipe [2], an open-source framework for implementing lightweight AI and machine learning models across a wide range of platforms, including Android, iOS, and Linux. MediaPipe follows a two-stage pipeline: it first detects the hand region using a Palm Detector, and then predicts 21 hand keypoints using a Landmark Model for gesture interpretation. The following Figure 2, illustrates all 21 keypoints on the hand.



0. WRIST	11. MIDDLE_FINGER_DIP
1. THUMB_CMC	12. MIDDLE_FINGER_TIP
2. THUMB_MCP	13. RING_FINGER_MCP
3. THUMB_IP	14. RING_FINGER_PIP
4. THUMB_TIP	15. RING_FINGER_DIP
5. INDEX_FINGER_MCP	16. RING_FINGER_TIP
6. INDEX_FINGER_PIP	17. PINKY_MCP
7. INDEX_FINGER_DIP	18. PINKY_PIP
8. INDEX_FINGER_TIP	19. PINKY_DIP
9. MIDDLE_FINGER_MCP	20. PINKY_TIP
10. MIDDLE_FINGER_PIP	

Figure 2: Hand Landmark Model

MediaPipe returns the coordinates of each keypoint on the hand, which we use to design both the start and stop signs for controlling the robot.The start sign is specifically designed to trigger the robot to begin operation. Without it, the robot would start moving immediately upon detecting any hand. In the start sign, the thumb is raised. We calculate the distances between the TIP and MCP of the other four fingers, as well as the distances between the MCP and PIP. If the distance from the TIP to the MCP is smaller than the distance from the MCP to the PIP, it indicates that the finger is curled. We perform this check for all four fingers, excluding the thumb. If all four fingers are curled, the robot is allowed to start moving. In addition to the start sign, we also define a stop sign to halt the robot. When the palm is open, the TIP of the middle finger typically has the longest distance to the wrist. Therefore, we calculate this distance, and if it exceeds a predefined threshold, it indicates that the palm is open, and the robot should stop. Besides the start and stop signs, we also need to calculate the angles of each hand to control turning. To determine the angles, we first compute the direction vector of the thumb and then use the atan2 function to calculate the angle between this vector and the vertical direction.

2.3. Control System

Once we obtain the angles of each thumb, we can design our control system. In this system, the right thumb is used to control steering, while the left thumb controls speed. When the right thumb rotates to the right, the angle is positive, ranging from 0° to 90°. In this case, the control system outputs a right turn angle ranging from 0° to 45°. Conversely, when the right thumb rotates to the left, the angle is negative, ranging from -90° to 0°, and the control system outputs a left turn angle between -45° and 0°. For the left thumb, rotating either left or right corresponds to acceleration. The input angle ranges from 0° to 90°, and the output speed ranges from 0 to 6 units. When designing the control system, we considered that hand gesture detection inherently includes noise, even when the fingers appear static, there are still small shakes detected. This situation makes a fuzzy control system particularly suitable for handling such uncertainties. However, to evaluate the performance of our fuzzy controller, we also implemented a linear controller based on direct linear mappings. In addition, we explored the effect of varying the number of fuzzy rules to observe how rule complexity impacts performance. Finally, we incorporated a filter into the fuzzy controller to achieve smoother control operations.

2.3.1. Linear Controller

In the Linear Controller design, the steering angle and the speed are directly mapped from the detected hand angles through linear functions. For steering control, the right hand angle, which ranges from -90° to 90°, is linearly mapped to a steering angle between -45° and 45° by applying a scaling factor of 0.5. Specifically, the steering angle is calculated as:

$$\text{Steering Angle} = \frac{\text{Right Hand Angle}}{2} \tag{1}$$

For speed control, the left hand angle, which ranges from 0° to 90°, is linearly mapped to a velocity between 0 and 6 units by applying a scaling factor of 6/90. The speed is calculated as:

$$\text{Speed} = \text{Left Hand Angle} \times \frac{6}{90} \tag{2}$$

Through this linear mapping, which is a crip method, provides a simple and deterministic relationship between the hand gestures and the car's movement, but this method not handle uncertainty.

2.3.2. Fuzzy Controller

In the fuzzy controller, we designed two configurations with different numbers of rules: one with 25 rules and another with 55 rules. In the 25 rules, the right-hand angle ranges from -90° to 90°, which is divided into five categories: large left, left, straight, right, and large right. These categories correspond to three levels of steering output: straight maps to none, left and right map to medium, and large left and large right map to large. For the left hand, the angle from 0° to 90° is similarly divided into five categories, each corresponding to a different speed output level. Since there are five categories for the right hand and five categories for the left hand, the total number of fuzzy rules is 25 rules. The membership functions for the right hand are shown in Figure 3. The membership functions for the left hand are shown in Figure 4.

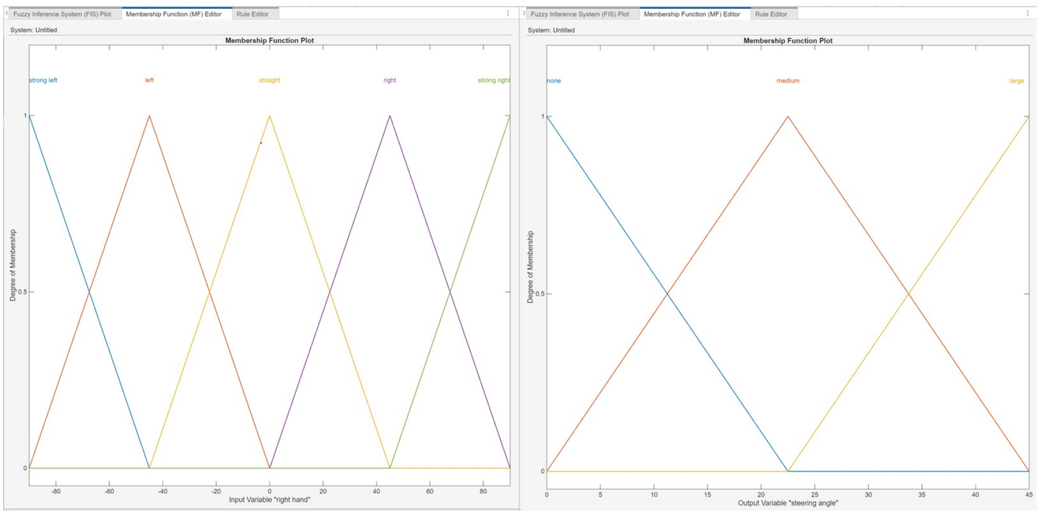


Figure 3: Membership function for Right Hand Input and Output (25 Rules)

In the 55 rules fuzzy controller, we kept the same membership functions for the left hand input as in the 25 rules. However, we refined the right hand input by increasing its output ranges. As previously described, in the 25 rules setup, the right hand steering output only had three ranges. In the 55 rules configuration, we expanded this to six distinct steering output ranges, and accordingly, updated the membership functions for the right-hand input to cover six categories. This refinement enables the fuzzy controller to achieve more fine-grained control over steering. As a result, the right-hand angle is now divided into 11 intervals, while the left-hand angle remains divided into 5 intervals, leading to 55 rules. The membership functions for right hand are shown in Figure 5.

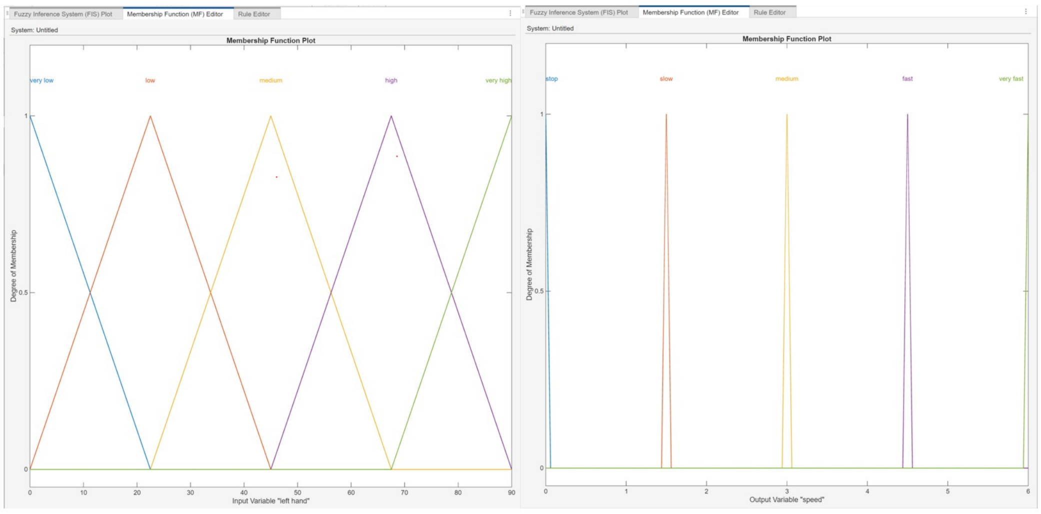


Figure 4: Membership function for Left Hand Input and Output (25 Rules)

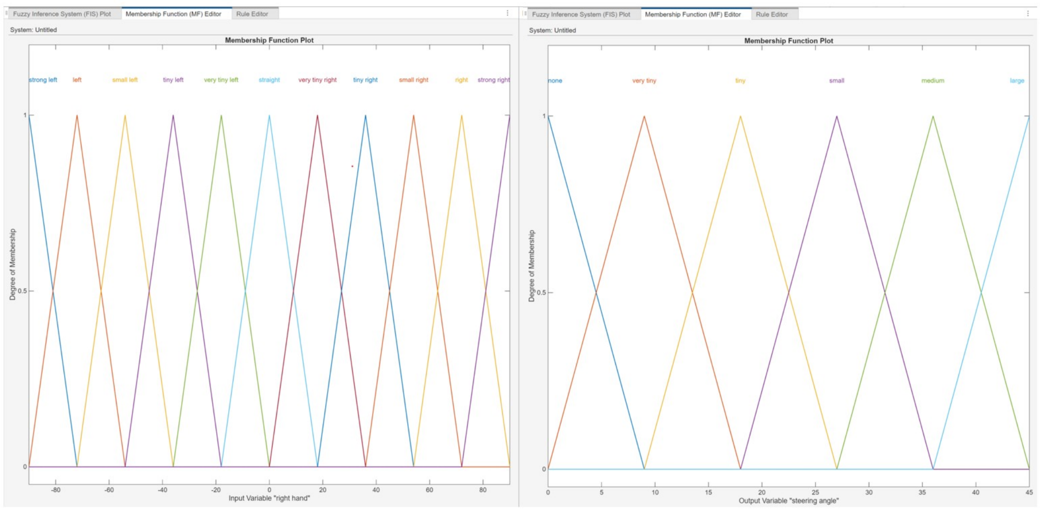


Figure 5: Membership function for Right Hand Input and Output (55 Rules)

2.3.3. Fuzzy and Filter Controller

While using the fuzzy controller, we observed that even with fuzzy logic, slight hand gesture variations could still cause abrupt changes in the output. To address this issue, we further integrated a filter on top of the fuzzy controller. The filter blends the new input value with the previous output, making transitions gradual rather than instantaneous. The filtering process follows the formula:

$$\text{filtered} = \alpha \times \text{current\_value} + (1-\alpha) \times \text{previous\_filtered} \tag{3}$$

By adjusting α, we can control the balance between responsiveness and smoothness. A larger α results in a faster but less smooth response, while a smaller α produces smoother output but with slower responsiveness.

3. Results

3.1. Performance

The performance evaluation of the model takes place in simulation, because the same model is used in simulation and real deployment. The evaluation in simulation is easier to perform, and we can collect and analyze results more easily. In this project, the simulation is performed on a local computer, specifically a 12th Gen Intel Core i9-12900K CPU. All the code is based on Python, and various libraries, including OpenCV, MediaPipe, NumPy, scikit-fuzzy, and Matplotlib, are used. Our simulation is conducted in three different scenarios: Case 1, Case 2, and Case 3. These three cases are straightforward and generate basic instructions for the robot.

3.1.1. Case 1

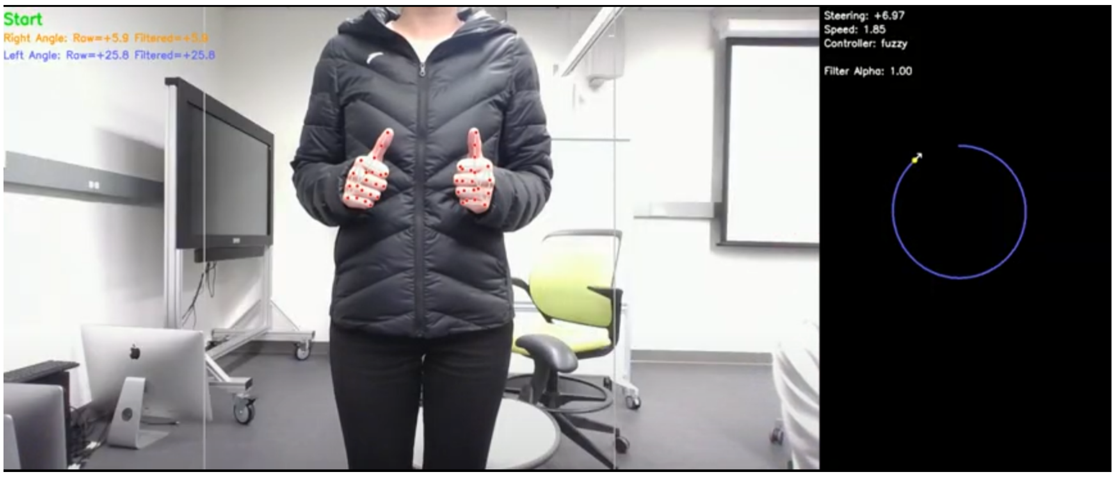

Start
Right Angle: Row=+5.9 Filtered=+5.9
Left Angle: Row=+25.8 Filtered=+25.8
Steering: +6.97
Speed: 1.85
Controller: fuzzy
Filter Alpha: 1.00

Figure 6: Case 1

In case 1, as shown in Figure 6, we keep both the left and right hands fixed at a constant angle, so both control inputs remain unchanged. This guarantees a constant forward speed and angular velocity, and ideally, the robot's path should be a perfect circle.

We evaluate the speed and angular velocity outputs using their standard deviations, since in case 1 they should be constant, so the smaller the standard deviation, the better. For the trajectory, we fit a circle to the recorded points using the Kasa circle-fitting method to get the circle's center and radius.

$$d_i = \sqrt{(x_i - a)^2 + (y_i - b)^2} \tag{4}$$

$$e_i = |d_i - r| \tag{5}$$

Then, using formula 4, we calculate the distance of each point to the fitted center. With formula 5, we compute the difference between this distance and the fitted radius as the error. We do this for every point and then take the average error.

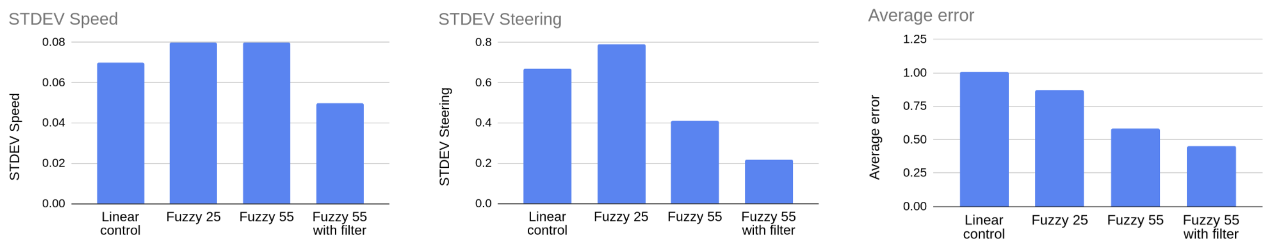
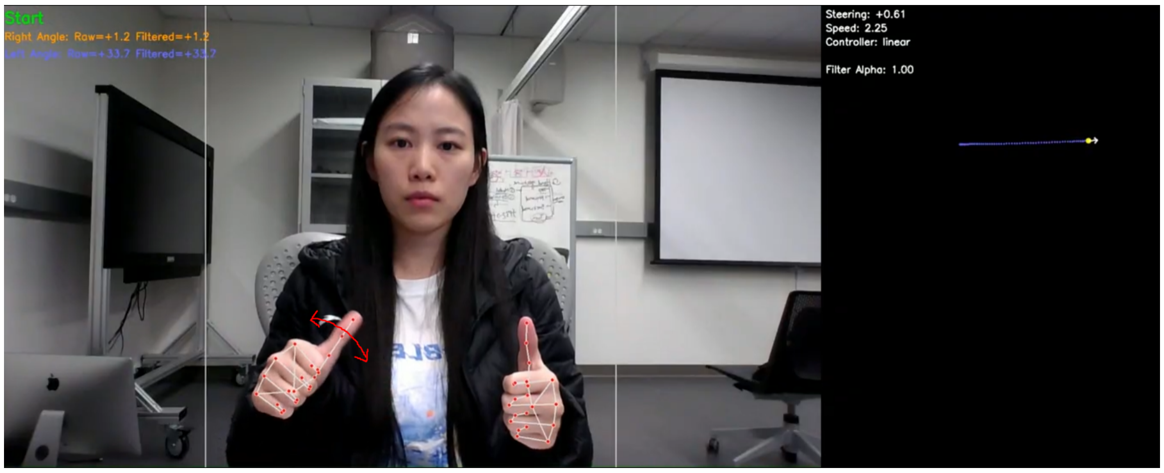


Figure 7: Case 1 results

Finally, Figure 7 presents the standard deviations of speed and angular velocity outputs, as well as the average trajectory error. We can see that the fuzzy controller with 55 rules and filters achieves the lowest standard deviations and average error, and that adding more fuzzy rules and filters further improves the performance.

3.1.2. Case 2


Start
Right Angle: Raw=+1.2 Filtered=+1.2
Left Angle: Raw=+33.7 Filtered=+33.7

Steering: +0.61
Speed: 2.25
Controller: linear

Filter Alpha: 1.00

Figure 8: Case 2

In case 2, as shown in Figure 8, we keep the right thumb pointing up and fixed, so the steering angular velocity stays at zero. Then I rotate the left hand 90° clockwise and back counterclockwise to make the speed first increase and then decrease. Because the steering angle remains zero and only the speed changes, the robot's path should be as straight as possible. In this case, to evaluate the speed output we only look at how smooth it is.

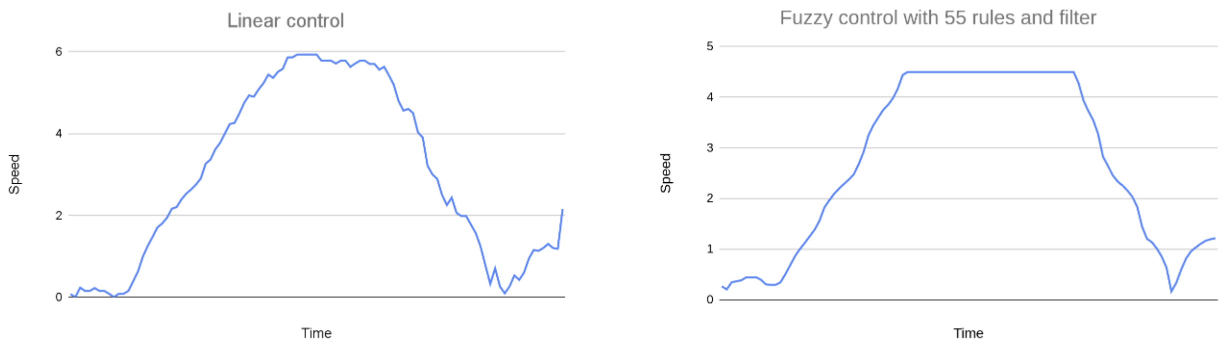


Figure 9: The output speed in case 2

As shown in Figure 9, the fuzzy controller's output is smoother than the linear controller's. For the steering angular velocity output, we still use the standard deviation because it should stay constant; the smaller the standard deviation, the better. For the trajectory, we use curvature to evaluate it. A straight line has zero curvature, so the smaller the average curvature, the better.

In Figure 10, we can see that the linear controller has both a lower standard deviation and a lower average curvature. This happens because our right hand is held very steadily at about 0°, so the linear controller's steering output also stays around 0°, whereas the fuzzy controller's output is spread over its lowest range, causing the robot to steer slightly.

3.1.3. Case 3

In case 3, our right thumb is still pointing up at 0°, but in real situations, it usually can not stay perfectly still—external factors can make it tremble and add noise to our control inputs. In case 3, we keep the right thumb up but introduce a slight shake to increase noise and simulate this scenario, while the left thumb follows the same motion as in case 2, where speed first increases and then decreases. We still want the steering angular velocity to remain as stable as possible despite the added noise, and we still want the trajectory to be as straight as possible.

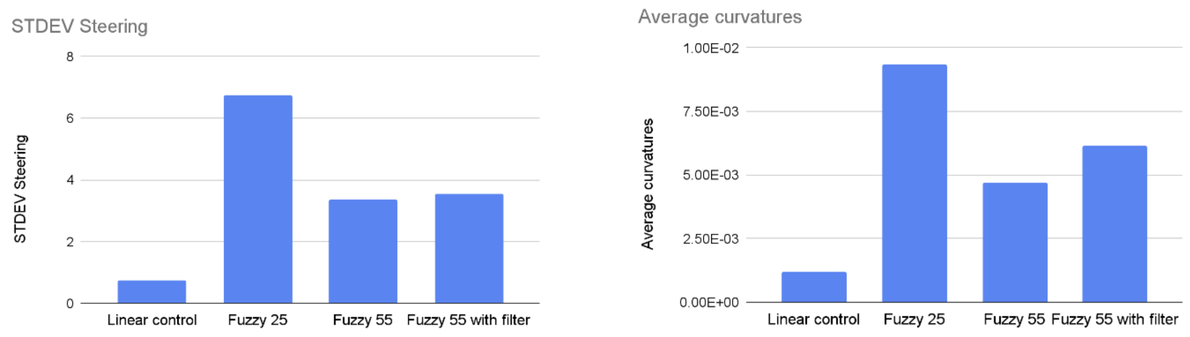


Figure 10: Case 2 results

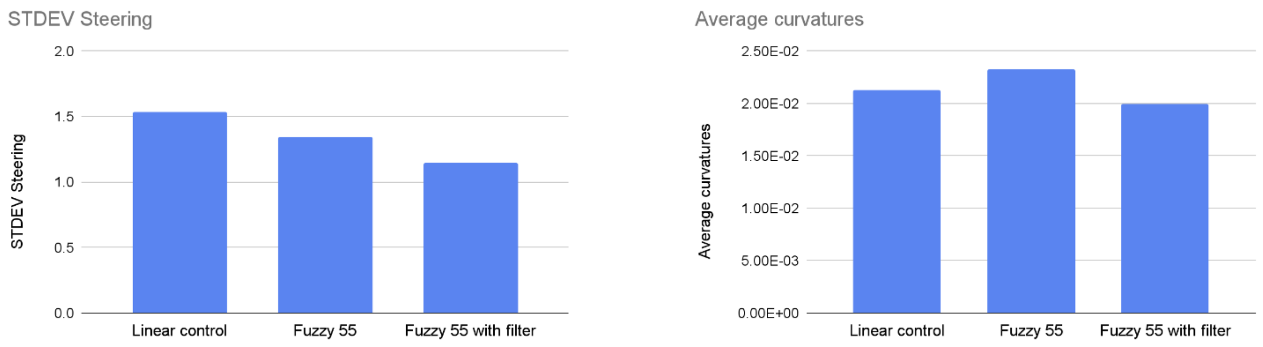


Figure 11: Case 3 results

Figure 11 shows the results: for both the standard deviation of steering angular velocity and the average trajectory curvature, the fuzzy controller with 55 rules and filters again achieves the best performance. We therefore conclude that fuzzy control with filtering handles noise better than simple linear control in this case.

3.2. Efficiency

The Jetson Nano is equipped with a quad-core ARM Cortex-A57 MPCore processor and a 128-core NVIDIA Maxwell GPU. We ran our performance tests on both CPU and GPU using tools and libraries such as Jtop, Psutil, and Python's time module, focusing on latency and energy consumption.

3.2.1. Latency

For latency, we measured both the MediaPipe delay and the end-to-end delay (which includes MediaPipe plus the controller). As shown in Figure 12, mediapipe on the GPU was about 2.4 times faster than on the CPU,

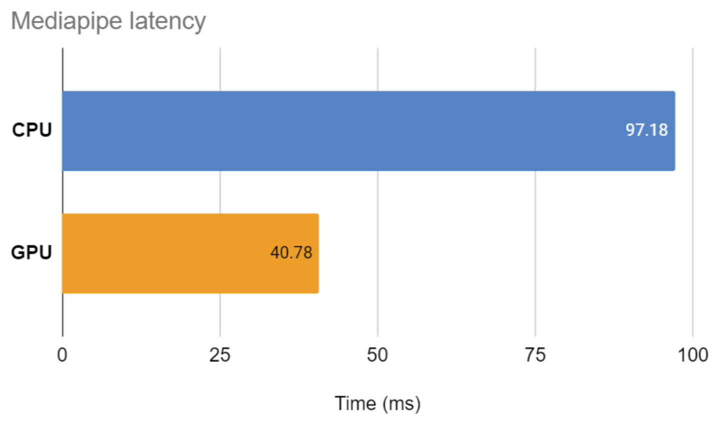


Figure 12: The latency of mediapipe

so we deployed mediapipe on the GPU. When we measured the end-to-end latency, the GPU outperformed

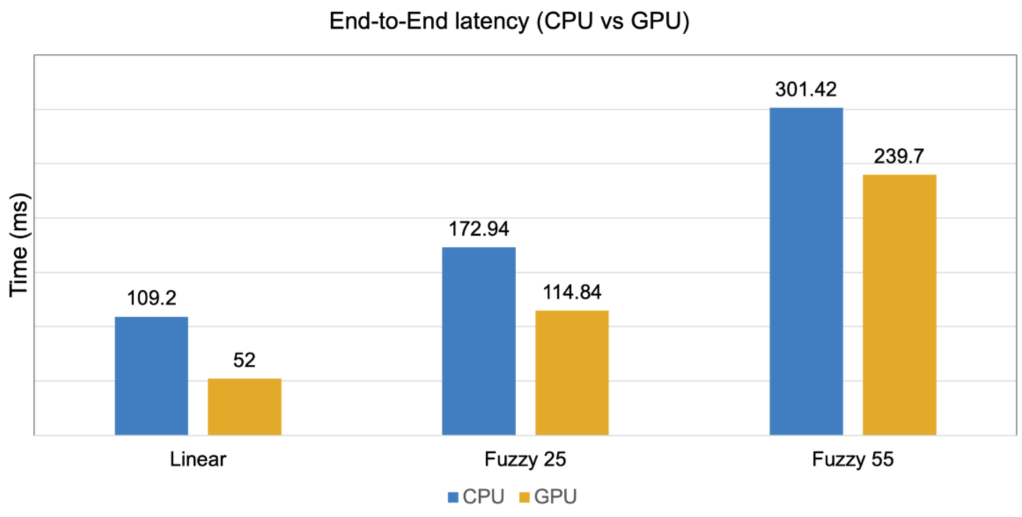


Figure 13: The latency of end-to-end model

the CPU for every control model, approximately 2.1 times faster with linear control, 1.5 times faster with fuzzy control using 25 rules, and 1.3 times faster with fuzzy control using 55 rules. Finally, on both CPU and GPU, all fuzzy controllers ran slower than the linear controller, because the latter only involves simple linear calculations, whereas fuzzy control requires more complex nonlinear operations, and adding more rules further reduces its computational efficiency.

3.2.2. Energy

For energy consumption, we measured the power used by the entire model when deployed on the CPU versus the GPU. As shown in Figure 14, the CPU consumed over eight times more energy than the GPU, so

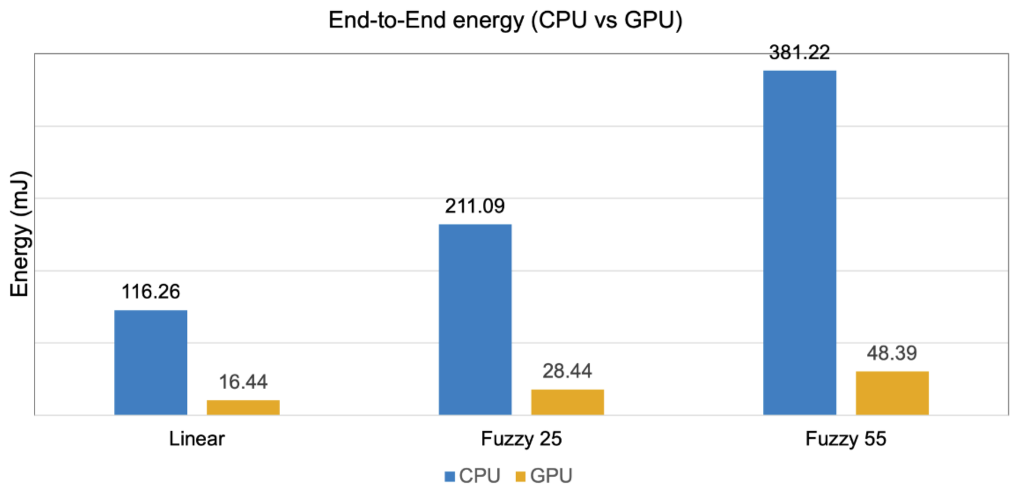


Figure 14: The energy of end-to-end model

running on the GPU significantly reduces the system's power usage. On the same device, linear control used less energy than fuzzy control, and adding more fuzzy rules increased the energy consumed by the fuzzy controller. In other words, as the model's computations become more complex and involve more operations, its energy consumption also rises.

4. Deployment

Our robot is the Yahboom ROSMaster X3, which is equipped with a Jetson Nano running the Robot Operating System (ROS) on Ubuntu 18.04. During deployment, all computations, including MediaPipe and the control model, are performed on the robot itself. As shown in Figure 1, data can be fed into the robot in two ways: using a pre-recorded video or streaming real-time camera data. In both cases, the data serve as input to node 1, where MediaPipe performs gesture recognition. The recognized gestures are then passed to node 2, the control model, to produce the final forward speed and steering angular velocity. Those two outputs are sent to the robot's motor control node to drive the corresponding motion commands.

We first tried loading a recorded video onto the robot and having it execute the sequence of gestures in the video. The robot did move, but its actual motion deviated from our expectations—this was mainly because we hadn't mapped the controller's outputs correctly to the robot's allowed speed and steering ranges. For the second method, we set up a client–server system: a local PC connected to the camera acted as the client, sending camera data, while the robot acted as the server, receiving that data and performing the computations. That approach failed due to firewall restrictions on the robot and network latency so that we couldn't establish communication. However, we were able to achieve remote control in simulation by replacing the robot with another local PC to receive the camera data. In that simulated environment, we successfully controlled the robot's movements.

5. Limitation

In this project, we evaluated the performance of our design and its efficiency on the robot, but several limitations were identified during testing. First, although a fuzzy controller was employed to smooth the outputs, finger shake noise still caused instability in the control signals. To address this, better filtering techniques should be considered in the future to reduce small hand tremors more effectively. Second, due to limited time, we were unable to fully resolve the network issues encountered during real-time robot remoting. While real-time communication was successfully tested between two computers, this setup failed when deployed on the actual robot. We suspect that network-related problems, such as unstable connections or firewall restrictions may have contributed to this failure. Third, we observed that the speed and steering behavior in the simulation did not fully correspond to that of the real robot. In the simulation, the vehicle responded appropriately. However, on the physical robot, mismatches in speed scaling led to problems. Specifically, if the steering rate was too high compared to the velocity, the robot tended to spin in place instead of following a wide turning trajectory as intended. Lastly, the fuzzy controller sometimes responded too slowly, occasionally causing the robot to crash. Future work should focus on optimizing the fuzzy control system to enhance responsiveness and overall stability.

6. Conclusion

We designed a gesture-controlled robot system using hand recognition and a set of controllers, including a linear controller, a fuzzy controller with 25 rules, a fuzzy controller with 55 rules, and a fuzzy controller combined with a filter, all deployed on the ROSMASTER X3 platform. The system supports multiple control modes: camera-based real-time control and predefined route control. In the camera-based mode, the robot is controlled dynamically by interpreting live hand gestures captured by a camera. In the predefined route mode, we use pre-recorded gesture sequences to guide the robot along specific paths. To evaluate the system's performance, we compared the behavior of the linear and fuzzy controllers within a simulated environment. Specifically, we analyzed the robot's performance under three different scenarios, which represent distinct operating cases to assess the adaptability of the control methods. We also assessed the latency of both the MediaPipe hand tracking module and the end-to-end system when deployed on both GPU and CPU platforms. Furthermore, we evaluated the energy consumption of the complete system for both the fuzzy and linear controllers under CPU and GPU execution, providing insight into the computational and power efficiency of different control strategies.

References

1. M Libina, R Dhevendhiran, and B Praveenkumar. Iot based night patrol robot leveraging voice technology for superior surveillance. In *2025 International Conference on Multi-Agent Systems for Collaborative Intelligence (ICMSCI)*, pages 355–360. IEEE, 2025.

2. Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, et al. Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172*, 2019.
3. Inderjeet Singh, Vipin Kumar, T Aarthy, T Akilan, Vishnu Sharma, et al. Real-time monitoring and surveillance in hazardous environments using iot-enabled robotics. In *2024 1st International Conference on Advances in Computing, Communication and Networking (ICAC2N)*, pages 1093–1097. IEEE, 2024.