Gesture-based Teleoperation System

Ian Bucu
NIE: X7770690W

May 22, 2025

Contents

1 Introduction

In this project, the objective is to develop a teleoperation system for a robot using hand gestures as control commands. By utilizing the ROS (Robot Operating System) framework and the MediaPipe library for gesture recognition, we aim to provide an intuitive and natural interface for controlling the robot.

The system consists of several interconnected modules: capturing the operator's images, processing and recognizing gestures, and generating Ackermann control commands for the robot. Additionally, a safety layer based on obstacle detection will be implemented, using data from a LIDAR sensor to avoid collisions and ensure the robot's integrity in its operational environment.

The development methodology includes configuring the ROS workspace with necessary dependencies, creating nodes for image capture and processing, and implementing gesture recognition algorithms using MediaPipe. The recognized gestures are translated into movement commands that are sent to the robot through ROS. For the safety layer, a node will be developed to continuously monitor LIDAR data and adjust movement commands to prevent collisions.

The Gazebo simulation environment will be used to develop and test the system in a controlled setting. This will allow validation of the system's functionality under various conditions and scenarios, ensuring its robustness and reliability before potential implementation on a real robot.

This project not only reinforces concepts in computer vision and robot control but also provides practical experience in integrating multiple technologies and tools into a cohesive project.

2 State of the Art

Gesture-based robot control is an active research area in robotics and human-robot interaction. This approach aims to create natural and accessible interfaces that enable users to control robots intuitively.

2.1 Gesture Recognition

The field of gesture recognition has undergone significant transformation alongside progress in computer vision and deep learning. Early systems depended on handcrafted feature extraction paired with classifiers like Support Vector Machines or Artificial Neural Networks. The introduction of deep learning has dramatically enhanced gesture recognition capabilities, with modern approaches using Convolutional Neural Networks and Transfer Learning models now able to identify and categorize gestures with remarkable precision [1].

2.2 MediaPipe

MediaPipe is an open-source library developed by Google that simplifies the implementation of media processing pipelines, including hand detection and gesture recognition. It employs optimized deep learning models capable of real-time operation, even on resource-constrained devices.

2.3 ROS (Robot Operating System)

ROS (Robot Operating System) is a flexible framework for robotics software development. It provides tools and libraries to help build robust, scalable robotic applications. ROS enables communication between different system modules through a publish-subscribe messaging system [2].

2.4 Obstacle Avoidance

Obstacle avoidance is a critical function in autonomous robot navigation. Sensors such as LIDAR are commonly employed to detect obstacles in the robot's environment.

2.5 Technology Integration

The integration of MediaPipe for gesture recognition with ROS for robot control and navigation represents an effective combination of advanced technologies.

3 Development

This section explains the various scripts developed to implement the gesture-based teleoperation system.

3.1 `capture_video.py`

This Python script, `video_publisher.py`, is responsible for capturing video frames from a webcam and publishing them as ROS `Image` messages to the `/operator/image` topic. This stream serves as the input for subsequent processing, such as gesture recognition, in other ROS nodes.

Initially, I overcomplicated the design by attempting to perform gesture classification directly within this `video_publisher.py` script. The commented-out lines involving `mediapipe as mp`, `hands.process(image)`, and `classify_gesture(hand_landmarks)` clearly illustrate this early approach. My intention was to process the image and identify gestures immediately after capturing the frame.

However, this turned out to be inefficient and went against the principle of separating concerns in a ROS-based system. The `video_publisher`'s sole responsibility should be to provide the raw video stream. Tasks like image flipping (which was necessary to correct the inverted webcam feed) and gesture recognition are better handled by a separate node, such as `pose_estimation.py`. This clear division of labor makes the system more modular, easier to debug, and more scalable. The current code correctly reflects this refined design, focusing exclusively on publishing the raw video data.

3.2 `show_camera_robot.py`

The `show_camera_robot.py` script serves as a vital visualization tool within the robot's teleoperation system. It implements a **ROS node that subscribes to the robot's camera feed** and displays this stream in a real-time OpenCV window. This provides the operator with immediate visual feedback of the robot's environment, crucial for effective gesture-based control. This component was straightforward to implement due to its focused role of simply displaying the video feed.

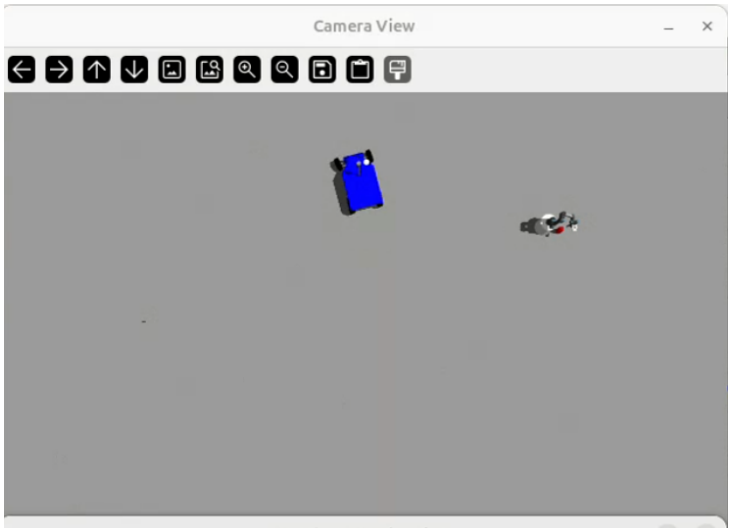
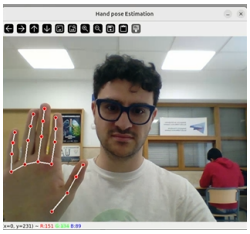


Figure 1: Real-time Camera Feed Visualization

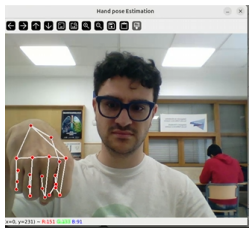
3.3 `pose_estimation.py`

The `pose_estimation.py` script implements the core gesture recognition system that translates hand poses into Ackermann steering commands for robot control. This node serves as the primary human-robot interface in the teleoperation system, combining computer vision and robotic control paradigms.

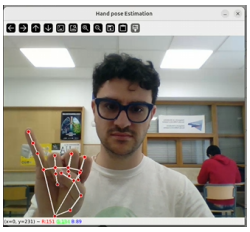
During the implementation of this module, an initial hurdle was encountered with image orientation; the input image required a horizontal flip to correct its inverted appearance. Once this issue was resolved, the development proceeded without significant complications. To enhance code modularity and maintain a clean callback function, the gesture recognition logic was abstracted into a separate `classify_gesture()` function.



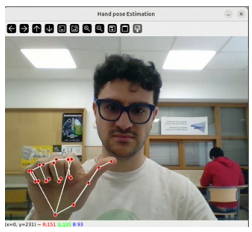
(a) Forward



(b) Stop



(c) Turn Left



(d) Turn Right

Figure 2: Individual recognized control gestures.

Architecture Overview

The system implements a multi-stage processing pipeline:

- **Vision Pipeline**: ROS image subscriber → OpenCV conversion → MediaPipe processing
- **Gesture Logic**: Landmark analysis → Geometric classification → Command generation
- **Control Interface**: Ackermann command publisher → Vehicle control topic

Gesture Classification Logic

The geometric-based classifier analyzes relative finger positions using these rules:

- **Forward**: All fingertips above wrist (y-axis)
- **Left**: Thumb dominant above wrist
- **Right**: Pinky dominant above wrist
- **Stop**: All fingertips below wrist

Control Mapping

Gesture	Speed (m/s)	Angle (deg)
Forward	1.0	0
Left	0.5	-90
Right	0.5	90
Stop	0.0	0

Table 1: Control Mapping for Robot Gestures

3.4 `obstacle_avoidance.py` and `obstacle_detection.py`

The `obstacle_callback` method processes the incoming point cloud. It calls `check_obstacles` to determine if any obstacles are within a predefined safety zone (currently defined as an x-coordinate less than 1.5 meters and an absolute y-coordinate less than 1.0 meter). If obstacles are detected, the robot's speed is set to 0.0, effectively bringing it to a halt. The `ackermann_callback` method simply stores the

most recent Ackermann command, which is then used as the base for any modifications by the obstacle avoidance logic.

This module proved relatively straightforward to implement. To maintain code cleanliness and promote modularity, a dedicated function was introduced to handle obstacle detection, separating this specific logic from the main processing loop.

The `obstacle_detection.py` script, while developed during a previous phase (Practice 1) and thus not central to this report's development narrative, plays a crucial role in the overall system. It acts as the primary component for identifying environmental obstacles by processing point cloud data from the LIDAR sensor. Its functionality is indispensable for the safety layer implemented in `obstacle_avoidance.py`, warranting this honorable mention.

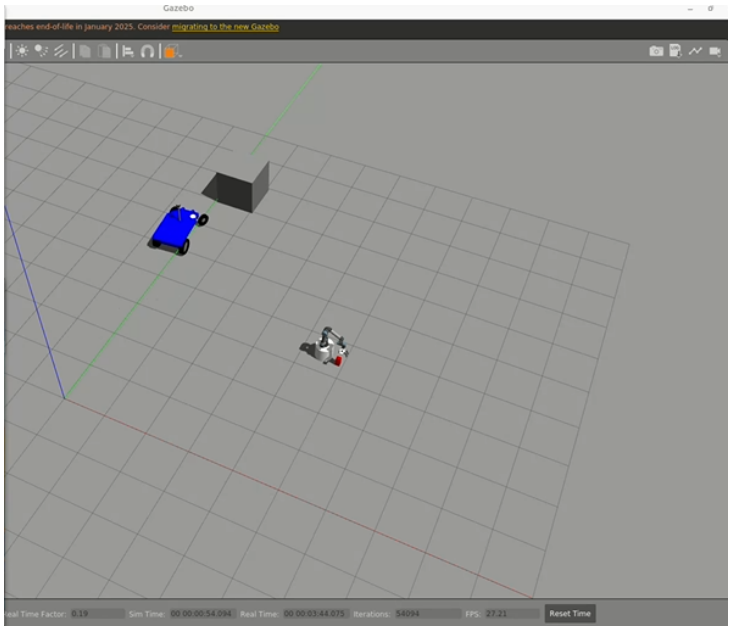


Figure 3: Robot stopping in front of obstacle

4 Conclusions

In this project, we developed a gesture-controlled teleoperation system for a robot, integrating **ROS** and **MediaPipe** for gesture recognition and obstacle detection. Our implementation covered image capture and processing, gesture classification, and the generation of control commands, ensuring safe robot navigation.

This project not only solidified our understanding of **computer vision** and **robot control** but also allowed us to effectively integrate multiple technologies into a cohesive system. The theoretical underpinnings of gesture recognition and the practical application of the Robot Operating System were crucial throughout this development [1, 2].

A demonstration video of the system's functionality is available at: `https://drive.google.com/file/d/1dVIHroy_99ZIbRTR-o4twTjRCaoyshxM/view?usp=sharing`

References

[1] P. Molchanov, S. Gupta, K. Kim, and J. Kautz, "Hand gesture recognition with 3d convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2015, pp. 1-7.

[2] A. Koubaa et al., *Robot Operating System (ROS)*. Springer, 2017, vol. 1.