EE417 Project Final Report: Drive a Simulated Car via Hand Tracking

Yunus Emre Danabaş
Mechatronics Eng.
Electronics Eng.
Sabanci University

Abstract

This report presents a real-time vision-based gesture teleoperation system for mobile robots that eliminates dependency on physical input devices. Addressing cost and accessibility barriers in traditional methods (joysticks, steering wheels), our solution uses standard webcams/RGB-D cameras to capture hybrid gestures: static poses (Stop, Speed Up/Down) classified by a lightweight MLP (99.65% accuracy) and dynamic trajectories (Turn Left/Right, Forward) recognized by an efficient LSTM (99.77% accuracy). The pipeline leverages MediaPipe for hand landmark extraction and TensorFlow Lite for deployment, achieving 13-25ms latency on consumer hardware. ROS-integrated gesture mapping enables real-time control in Gazebo simulations. Key contributions include: (1) Dual-branch architecture enabling proportional steering and discrete commands, (2) Containerized implementation supporting CPU/GPU environments, and (3) Open-source release with full reproducibility. Experimental validation confirms robust performance under real-time constraints, providing a practical alternative where conventional controllers are infeasible.
Keywords*: Gesture teleoperation, human-robot interaction, MediaPipe, ROS, Gazebo, real-time vision.*

1. Introduction

Robot teleoperation traditionally relies on physical input devices such as joysticks, keyboards, or specialized steering wheels (e.g., Logitech G29). While these devices offer reliable and responsive control, they create significant barriers related to cost, portability, and accessibility, especially in educational contexts, assistive robotics, or field-deployable systems. Gesture-based Human–Machine Interaction (HMI) offers a promising alternative, enabling intuitive, natural control of robotic systems without requiring specialized or costly hardware.

However, current gesture-based teleoperation solutions often either depend on expensive depth sensors or wearable devices, or lack robust continuous control capabilities, limiting their practical applicability and adoption [3, 6]. A low-cost, camera-only solution capable of real-time, proportional steering and discrete command recognition remains an open research challenge.

To address these limitations, this project introduces a real-time, vision-based gesture teleoperation pipeline for simulated mobile robots. Using a standard webcam or affordable RGB-D camera, the system captures visual hand gestures, supporting both discrete pose-based commands (e.g., *Stop*, *Speed Up*, *Speed Down*) and continuous steering gestures (e.g., *Turn Left*, *Turn Right*), forming a versatile, hybrid control interface (Figure 1).

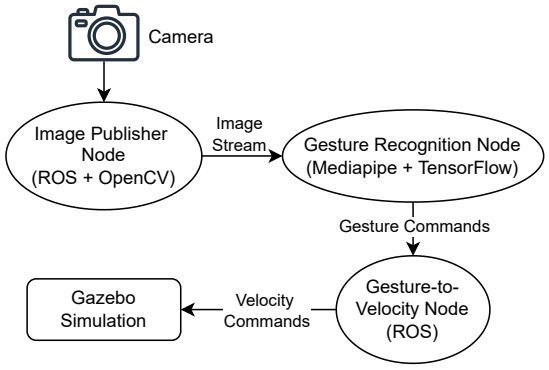


Figure 1. Simplified system diagram illustrating the flow from gesture capture to simulated robot control.

The pipeline leverages MediaPipe for robust extraction of 21 hand landmarks per frame, with gesture

classification achieved using lightweight neural models deployed through TensorFlow Lite. Individual static hand poses are classified via a compact neural network, while multi-frame gesture trajectories are processed by an efficient LSTM-based model (see Section 3 for implementation details). Recognized gestures are then translated into velocity commands using ROS (Robot Operating System) and executed within a Gazebo simulation environment.

The complete system is fully open-source and reproducible, with containerized deployments optimized for both CPU and GPU environments. It accommodates a variety of camera hardware, including standard webcams and Intel RealSense cameras. Experimental results demonstrate that our gesture classifiers achieve up to 99% accuracy and operate comfortably in real time, even on resource-constrained devices (Section 4).

In summary, this work presents a practical, lightweight, and extensible gesture-based teleoperation framework. It demonstrates robust real-time control capabilities, providing a compelling proof-of-concept for vision-driven robotic teleoperation in scenarios where traditional control hardware is infeasible or undesirable.

2. Problem Description

2.1. Context and Present Practice in Mobile–Robot Teleoperation

Most mobile robots in education, research, and industrial contexts rely heavily on traditional input devices such as joysticks, keyboards, or commercial-grade steering wheels (e.g., Logitech G29). While these devices are effective and reliable, they introduce significant barriers including increased costs, reduced portability, and limited accessibility—particularly for novice or mobility-impaired users. Vision-based gesture control methods have emerged as promising alternatives, offering comparable expressiveness with significantly reduced hardware demands.

2.2. Prior Gesture-Based Teleoperation Approaches

Existing gesture-based teleoperation systems primarily use one of three hardware classes: RGB-only cameras, depth or infrared (IR) sensors, and wearable inertial measurement unit (IMU)-based gloves. Each technology has specific strengths and limitations, as summarized in Table 1.

While RGB-only (camera-based) approaches are particularly appealing due to their affordability and simplicity, they typically suffer from lower robustness to environmental variability, limited continuous control capabilities, and inconsistent integration with standard robotics middleware like ROS. Depth-sensing and wearable systems address robustness concerns but do so at increased hardware complexity and user inconvenience.

Family	Hardware	Examples	Pros / Cons
Camera-only	Webcam	[6, 7]	+ Low cost – Lighting sensitive
Depth / IR	Kinect, Leap Motion	[1, 3]	+ Accurate 3-D – Higher cost
Wearable IMU	Data-glove, armband	[4, 5]	+ Robust – Requires to wear equipment

Table 1. Typical gesture teleoperation technologies for mobile robots.

2.3. Key Gaps Identified

Analyzing existing approaches, we identify four significant gaps that our project aims to address:

G1 Cost-effectiveness: Many high-performance systems still require costly dedicated depth sensors or instrumented gloves, limiting widespread adoption.

G2 Continuous control capability: Few camera-only solutions offer proportional steering alongside discrete commands, constraining user control granularity.

G3 Turn-key reproducibility: Fully containerized, easy-to-deploy ROS and Gazebo pipelines with reliable performance on both CPU and GPU are rare, hindering educational and research use.

G4 Latency measurement: Existing literature rarely provides detailed per-frame latency analyses under realistic simulation loads, limiting confidence in real-world usability.

2.4. Research Question and Objectives

Motivated by these gaps, this project seeks to answer the following research question:

*Can an RGB-only, lightweight vision-based pipeline achieve real-time (**$<$30 ms**), robust (**$\geq$97%**) gesture teleoperation of mobile robots in ROS/Gazebo without specialized hardware?*

To systematically address this question, we establish the following measurable objectives:

- **Accuracy:** Achieve at least 99% accuracy for static gesture recognition and at least 99% accuracy for dynamic (steering) gestures, as validated in Section 4.
- **Responsiveness:** Maintain end-to-end latency around 25 ms on CPU and approximately 13 ms on GPU setups while actively running Gazebo simulations (see Section 3.6).

Study (Year)	Method/Device	Env./Application	Key Contributions & Accuracy
Raheja et al. (2015) [3]	Kinect RGB-D, DTW/HMM	Indoor HRI	96.9% static gesture accuracy; demonstrated RGB-D teleoperation feasibility
AutoNav Teleop (2024) [7]	Webcam + MediaPipe	Multi-robot Gazebo	50% task-time improvement using discrete gestures
Olikkal et al. (2024) [2]	MediaPipe + biomimetic DNN	Humanoid robot hand control	95.7% across 33 static poses
Trinh et al. (2023) [6]	Webcam + TFLite	ROS-based teleoperation	Open-source ROS integration; no continuous steering
Our Project (2025)	**Webcam + TFLite (MLP + LSTM) Docker (CPU/GPU)**	**ROS + Gazebo for mobile robots**	**99% static and steering accuracy; $\leq$13 ms latency on GPU; hybrid discrete/continuous control; fully reproducible**

Table 2. Summary and comparison of influential gesture-control studies with the proposed method.

- **Hybrid control:** Support at least four discrete commands and three proportional steering commands to enhance control expressiveness.
- **Reproducibility:** Provide a fully reproducible, single-command Dockerized deployment compatible with CPU and GPU hosts and capable of supporting commodity cameras such as standard webcams or Intel RealSense.

2.5. Comparison with Existing Methods

Table 2 situates this project's contributions within the landscape of influential recent gesture control studies. Our solution uniquely combines high accuracy, thoroughly measured real-time latency performance, and comprehensive reproducibility via a containerized ROS and Gazebo stack.

3. Methods

3.1. System Overview

Figure 2 presents the complete architecture of the developed gesture-based teleoperation system. A 30 FPS camera stream (webcam or Intel RealSense D435) is published by a ROS node. A dual-branch gesture recognition module classifies static and dynamic hand gestures, mapping these to semantic velocity commands. A dedicated ROS mapping node then converts gesture labels to `/cmd_vel` twist commands, driving a differential-drive robot simulated in Gazebo.

3.2. Hardware & Software Stack

Development initially targeted a Lenovo ThinkPad E14 (Intel i5-10210U, no GPU) and later migrated to a desktop with an NVIDIA RTX 4060 Ti GPU. Dockerized deployments ensure consistent and reproducible setups on Ubuntu hosts (20.04 to 24.04).

Component	Specification
Camera	RealSense D435 or V4L2 webcam (960×540 px, 30 FPS)
Vision middleware	MediaPipe Hands v0.10 (21 landmarks)
ML inference	TensorFlow Lite v2.13 (MLP/LSTM, FP16 quantized)
GPU delegate	CUDA 11.8 + cuDNN 8 (optional, ¡ 8 ms inference)
Robot middleware	ROS Noetic (Docker)
Simulation	Gazebo 11 (diff-drive controller)
Languages	Python 3.8 (Bash scripts)

Table 3. Summary of hardware and software components.

3.3. Data Acquisition Pipeline

Camera abstraction. The `camera_publisher_node.py` standardizes V4L2 webcam and RealSense D435 inputs into a single ROS topic (`/image_raw`). Decode latency is consistently below 0.6 ms on GPU hardware.

Steering-wheel rig. A matte-black steering wheel is mounted on a stable 3D-printed base, clamped to a hobby vise (Fig. 3). The Intel RealSense D435 camera is positioned approximately 25 cm behind the wheel (not fixed), directly facing the user's hands (Fig. 4). This setup ensures consistent MCP (Metacarpophalangeal) trajectory capture.

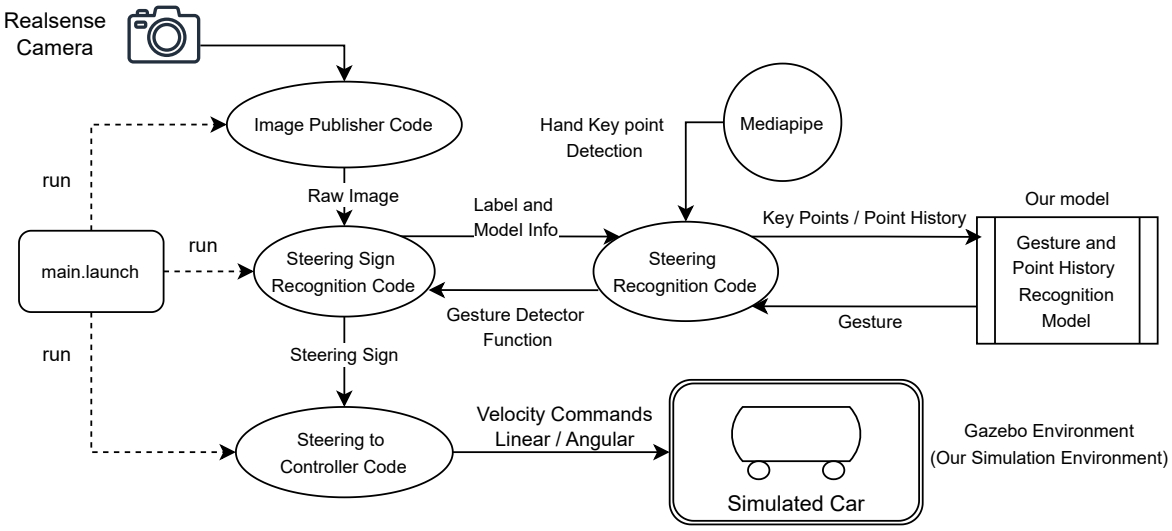


Figure 2. Detailed system diagram: camera ingestion, gesture inference (static and dynamic branches), ROS topic flow, and Gazebo actuation.

Figure 3. 3D-printed mount for stable steering wheel positioning.

Figure 4. Complete setup for capturing steering gestures.

Recorder GUI. A custom GUI visualizes the real-time MediaPipe landmark detections, allowing quick gesture labeling (via number keys). It records either: (i) a 42-dimensional vector (21 keypoints × 2D) for static gestures, or (ii) a 128-dimensional vector (16 consecutive frames × 4 MCP joints × 2D) for dynamic gestures. Live annotation significantly reduces labeling errors (Fig. 5).

Dataset composition. The complete dataset—collected from a single participant—was divided in two steps. First, 25% of the samples were set aside as an untouched *test* set. The remaining 75% were then split 80/20 into *train* and *validation* subsets, yielding per-class proportions of roughly 60% train, 15% val, and 25% test. Table 4 details the exact counts for each gesture.

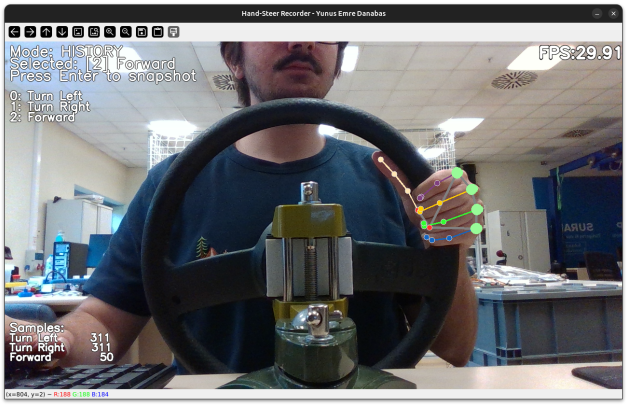


Figure 5. Recorder GUI showing real-time landmark detection and labeling interface.

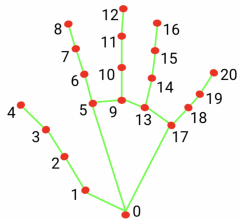
Branch	Label	Total	Train	Val	Test
Static (Keypoint)	Stop	1822	1089	269	464
	Holding Wheel	547	344	78	125
	Speed Up	1064	618	167	279
	Speed Down	1191	723	180	288
Dynamic (Point History)	Turn Left	512	301	82	129
	Turn Right	504	283	87	134
	Forward	697	443	88	166

Table 4. Dataset distribution across gesture types after the 60/15/25 train–validation–test split.

3.4. Gesture Recognition Pipeline

The pipeline has two branches:

Static branch. A small MLP ($20 \to 10 \to 4$; 1.1 k params) classifies four static gestures (*Stop*, *Holding Wheel*, *Speed Up*, and *Speed Down*) from wrist-normalized 21-keypoint vectors (Fig. 6).



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

Figure 6. MediaPipe hand landmark numbering used for gesture recognition.

Dynamic branch. An LSTM (32 units) followed by a dense layer (32 units) classifies dynamic gestures (*Turn Left*, *Turn Right*, *Forward*) from sequences of 16 frames of 4 MCP joints (128D input vector).
The 4 MCP joints correspond to indices 5 (Index_MCP), 9 (Middle_MCP), 13 (Ring_MCP), and 17 (Pinky_MCP) in the hand landmark set shown in Figure 6. FP16 quantization allows inference under 8.5 ms on GPU.

Branch fusion. Angular velocity changes (*Turn Left/Right*) require the *Holding Wheel* gesture. Linear velocity (*Speed Up/Down*) is adjusted without holding the wheel. Gesture outputs are majority-vote smooths over 16 frames to eliminate transient noise.

3.5. ROS–Gazebo Integration & Control Mapping

All ROS nodes run inside a single Docker container, communicating via ROS topics as depicted in Figure 2. Gesture labels map directly to velocities (Table 5).

Gesture	Linear velocity (m/s)	Angular velocity (rad/s)
Stop	0.00	0.00
Speed Up	$+0.08$	No change
Speed Down	-0.08	No change
Turn Left	No change	$+0.05$
Turn Right	No change	-0.05
Forward	No change	No change

Table 5. Gesture-to-velocity mapping. Angular commands require *Holding Wheel* to be active.

3.6. Performance Benchmarking

Mean latency measurements over 1000 frames confirm real-time capability, strongly recommending GPU deployment for smooth simulation performance (see Table 8).

3.7. Reproducibility & Release

The complete pipeline is publicly accessible via GitHub: `yunusdanabas/hand_steer_sim`, including Docker images, pre-trained models, and scripts, ensuring rapid and hassle-free reproduction.

4. Results

This section presents the experimental evaluation of our gesture-based control pipeline. Our experiments are designed to answer the following questions:

1. Can static and dynamic gesture classifiers achieve high accuracy under limited model size constraints?

2. Can the system maintain real-time performance across CPU and GPU setups, even during active Gazebo simulation?
3. Does the complete ROS-integrated control loop produce smooth, intuitive robot behavior in simulation?
4. Are certain gestures, such as *Forward*, more challenging to reliably detect or maintain in practice?

4.1. Model Performance: Steering Gestures

Two distinct models were trained and evaluated for gesture recognition:

- A **Keypoint MLP model** for static poses (*Stop*, *Holding Wheel*, *Speed Up*, *Speed Down*), using 42-D MediaPipe hand landmarks.
- A **Point History LSTM model** for dynamic gestures (*Turn Left*, *Turn Right*, *Forward*), based on 128-D MCP joint trajectories over 16 frames.

Training was performed using the Adam optimizer with a learning rate of 0.001 for up to 1000 epochs. Early stopping monitored validation loss, halting training at approximately epoch 100 for the MLP model and epoch 36 for the LSTM. The data were split into 60% train, 15% validation, and 25% test (see Section 3.3 and Table 4). All reported metrics reflect final performance on the held-out test set; no cross-validation or multiple random seeds were used. Detailed model architectures and training hyperparameters are provided in Appendix E.

The MLP classifier achieved 99.65% accuracy, while the LSTM achieved 99.77% macro F1-score. Confusion matrices for both classifiers are shown in Figures 7 and 8, with clear class separation.

Training and validation curves show stable convergence. The LSTM model's validation loss fell below 0.02 early in training.

Gesture	Precision	Recall	F1-Score	Support
Stop (0)	1.00	1.00	1.00	464
Holding Wheel (1)	0.99	1.00	1.00	125
Speed Up (2)	1.00	1.00	1.00	279
Speed Down (3)	1.00	0.99	0.99	288
Macro Avg	1.00	1.00	1.00	1156
Weighted Avg	1.00	1.00	1.00	1156
Accuracy			**0.9965**	1156

Table 6. Classification report – Static Keypoint Classifier (MLP).

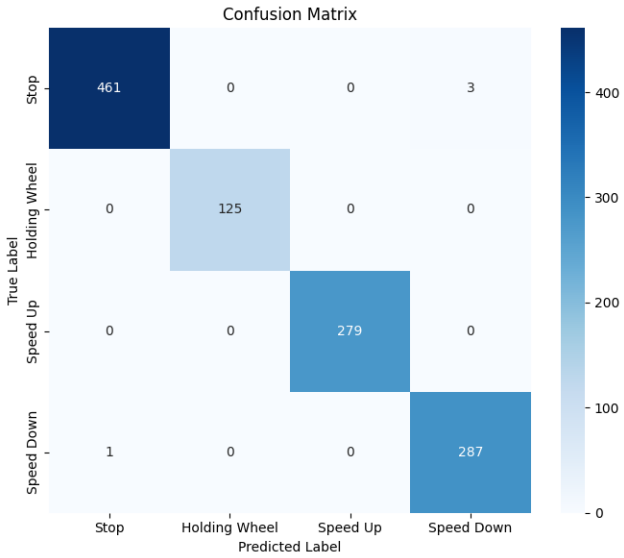


Figure 7. Confusion matrix – Static Keypoint Classifier (MLP). Axes denote true (Y) vs. predicted (X) labels.

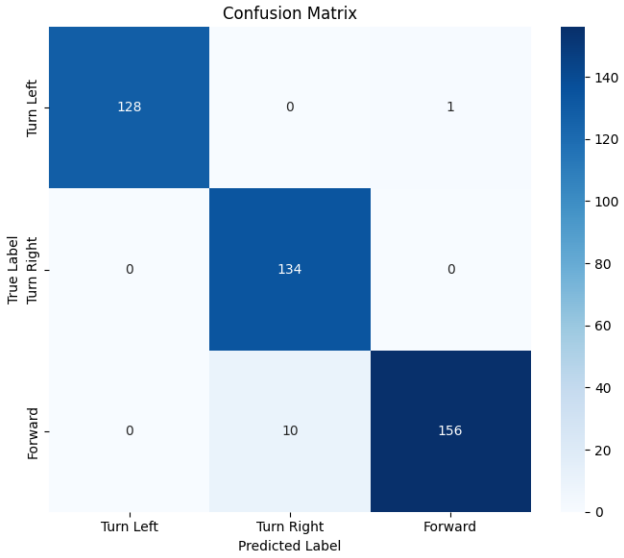


Figure 8. Confusion matrix – Dynamic Point History Classifier (LSTM). Axes denote true (Y) vs. predicted (X) labels.

4.2. Latency Analysis

End-to-end latency was measured across three different deployment configurations: a Lenovo ThinkPad (with and without Gazebo), and a desktop with NVIDIA RTX 4060 Ti. The "Inference" time includes MediaPipe detection and MLP/LSTM classification. Display and publishing overheads were also measured independently.

Both GPU and CPU-only configurations (without Gazebo) achieved real-time performance well below the 33 ms threshold for 30 FPS interaction.

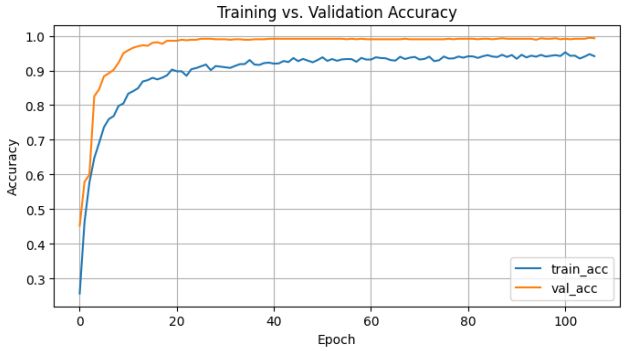


Figure 9. Training and validation accuracy – Keypoint MLP model.

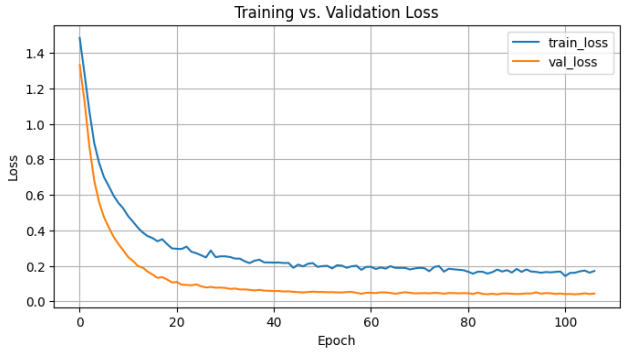


Figure 10. Training and validation loss – Keypoint MLP model.

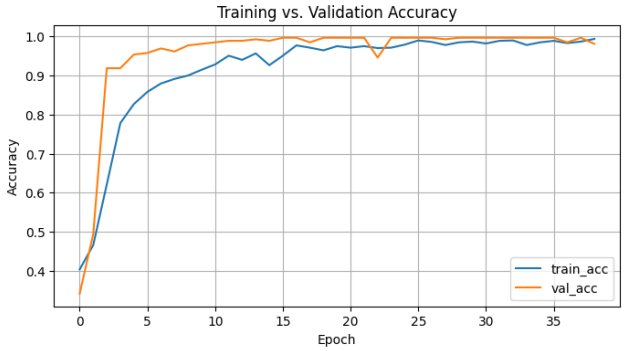


Figure 11. Training and validation accuracy – Point History LSTM model.

Gesture	Precision	Recall	F1-Score	Support
Turn Left (0)	1.00	0.99	1.00	128
Turn Right (1)	1.00	1.00	1.00	126
Forward (2)	0.99	1.00	1.00	175
Macro Avg	1.00	1.00	1.00	429
Weighted Avg	1.00	1.00	1.00	429
Accuracy			**0.9977**	429

Table 7. Classification report – Dynamic Point History Classifier (LSTM).

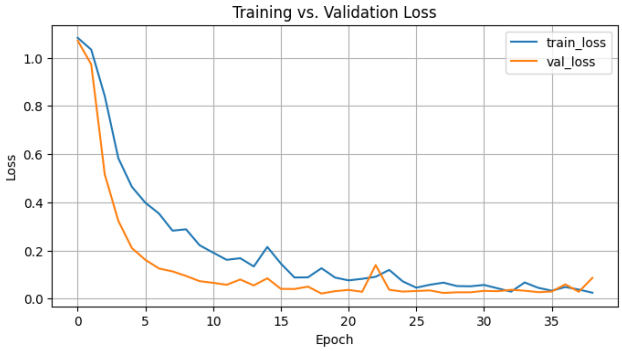


Figure 12. Training and validation loss – Point History LSTM model.

Metric (ms)	ThinkPad (no Gazebo)	ThinkPad (+Gazebo)	GPU (4060 Ti)
Decode	0.47	4.69	0.53
Inference	20.10	94.10	8.56
Display	4.26	10.72	4.28
Total	25.03	109.62	13.19
Callback FPS	39.53	9.68	75.69

Table 8. Latency per stage and resulting FPS on different hardware platforms.

4.3. Simulation Trials

Live gesture-based teleoperation was tested in Gazebo. The robot performed basic motion commands — forward driving and turning left/right — with minimal visible lag. Transitions between gestures were smooth for turns, though occasional flickering was observed when sustaining the *Forward* gesture.

The `Forward` gesture maintains constant angular velocity as long as the detected class remains unchanged. Videos of the experiments are linked below:

- Inference Overlay Demo
- Driving Demonstration in Gazebo

4.4. Model Size and Deployment Efficiency

Both models are compact and designed for low-latency execution:

- **Keypoint MLP:** 1.1k parameters, ∼4.4 kB, 3 ms inference on CPU.
- **Point History LSTM:** 6.4k parameters, ∼25 kB (quantized), 8.5 ms inference on GPU.

The models were quantized to FP16 and exported as `.tflite` files using TensorFlow Lite. Deployment was validated on desktop and laptop setups, without observable accuracy degradation due to quantization.

4.5. Forward-Gesture Challenge

While *Forward* achieved excellent quantitative results during training, real-time operation revealed it to be

harder to maintain than turning gestures. Its subtle, symmetric nature lacks the lateral displacement seen in *Turn Left* or *Turn Right*, causing occasional misclassification.

This limitation is discussed further in Section 5, along with potential solutions such as gesture redesign or semantic refinement.

5. Discussion

This section reflects on system behavior, gesture design, control responsiveness, and real-world usability based on empirical observations and qualitative feedback from simulation trials.

5.1. Forward Gesture Challenges

Despite achieving high accuracy during offline testing, the *Forward* gesture showed some practical limitations in simulation. Particularly at the edges of turning maneuvers, minor hand deviations were often misinterpreted as steering commands, causing unintended drift. This issue arises because the current model interprets gestures based on relative hand movements rather than absolute poses, making it sensitive to small, inadvertent motions. Introducing a dead-zone filter in the `wheel_to_twist` node could mitigate these false activations, enhancing robustness and preventing unintended steering at neutral hand positions.

5.2. Operator Feedback and Intuitiveness

The overall system demonstrated responsiveness and consistent performance, with negligible latency across both GPU-accelerated and CPU-only setups. However, the metaphor of using a steering wheel for controlling a differential-drive robot created an ergonomic mismatch between user expectations and actual robot behavior. Although differential-drive was selected primarily due to simulation simplicity and project time constraints, transitioning to an Ackermann-steered simulated vehicle would likely provide a more intuitive interaction experience.

Despite this metaphorical mismatch, the implemented gesture vocabulary was straightforward and easily maintained without noticeable user fatigue or awkwardness during extended sessions. All core gestures (*Stop*, *Holding Wheel*, *Speed Up*, *Speed Down*, *Turn Left*, *Turn Right*, and *Forward*) were reliably recognized and consistently executed.

5.3. Gesture Set Design Considerations

The gesture set was deliberately minimized to ensure clear class separation, reduce labeling complexity, and expedite training. Although effective for the project's scope, future deployments could benefit from an expanded gesture vocabulary. For instance, incorporating additional static or dynamic gestures—such as finger-based signaling (e.g., turn indicators, lights control)—while maintaining a grip on the wheel could substantially enhance command expressiveness. Such expansions might require more advanced recognition methods, possibly integrating static classifiers, temporal analysis, or ensemble-based decision fusion.

5.4. Single-Hand vs. Two-Hand Interaction

The current implementation was optimized for single-hand interaction to prioritize simplicity and latency. However, two-handed gestures present a valuable avenue for enhancement, potentially enabling more precise movement detection and additional intuitive commands, such as emergency stops or auxiliary vehicle controls (e.g., activating lights or signals). Crucially, integrating these two-handed gestures can be modularly layered atop the existing architecture without extensive reengineering.

5.5. Control Mapping and Real-Time Feedback

The present control approach relies on discrete velocity increments, simplifying initial testing but limiting fluidity compared to analog controls. A future improvement could involve implementing smoother velocity scaling through more nuanced gesture analysis, potentially leveraging metrics such as gesture trajectory or displacement magnitude. This enhancement would likely yield more natural and responsive robot teleoperation.

Moreover, the lack of real-time visual feedback (e.g., a GUI overlay indicating speed or direction) occasionally left operators uncertain about the robot's velocity state, especially during subtle adjustments. Integrating an intuitive GUI-based speed indicator would significantly improve user experience and control precision.

5.6. Deployment Feasibility and Limitations

The developed pipeline demonstrates strong potential for deployment due to its minimal hardware requirements (standard webcams or RealSense cameras), compact models, and efficient execution on low-cost hardware. The pipeline remained robust even when camera positions varied significantly, and its performance was consistent across cluttered backgrounds and varying lighting conditions.

Nonetheless, several aspects would require refinement for practical field use:

- Expansion of the gesture set, validated across diverse users to ensure consistent and intuitive command recognition.
- Incorporation of clear, real-time feedback mechanisms to inform operators about active commands and system state.
- Recalibration of the steering control logic and velocity mapping for physical robot dynamics and real-world vehicle constraints.

5.7. Dual-Branch Fusion Design

The dual-branch design—employing static pose classification to gate dynamic gesture commands—proved instrumental in system stability. The *Holding Wheel* condition effectively minimized unintended gesture activations, ensuring that dynamic steering commands were recognized only during deliberate user engagement. However, a few isolated misclassifications occurred, particularly during extensive hand rotations, occasionally interpreting the gesture as *Stop*. Nonetheless, this fusion architecture provided a substantial reliability advantage over simpler, unified model alternatives, which might otherwise blur the boundary between intentional motion and idle states.

5.8. Generalizability and Robustness

The chosen MediaPipe Hands framework demonstrated exceptional robustness against lighting variations, skin-tone differences, and minor camera movements during training and simulation. Although these preliminary results are promising, broader validation is necessary. Future deployments should incorporate larger-scale user studies to rigorously evaluate inter-user variability, gesture intuitiveness, long-term fatigue, and generalization performance, thereby ensuring broader applicability and reliability.

6. Conclusions

This project demonstrated the feasibility of real-time, camera-based gesture control for mobile robots, even on low-end hardware (see Appendix B for detailed specifications). Using TensorFlow Lite, MediaPipe Hands, and ROS within a Gazebo simulation pipeline, we implemented a dual-branch gesture recognition system—combining static (hand pose) and dynamic (hand trajectory) classifiers—that achieved minimal latency (13 ms GPU, 25 ms CPU) and high classification accuracy (99.8% macro-F1).

Despite excellent offline performance metrics, real-world simulation trials highlighted certain usability challenges. Notably, the *Forward* gesture exhibited instability in practical conditions, triggering unintended drift as discussed in Section 5. This underscores a crucial insight: high offline test accuracy does not always translate to optimal real-world usability.

Our design prioritized rapid prototyping and practical deployment considerations. The resulting system performs reliably across CPU- and GPU-based setups, with the complete pipeline—including training scripts, ROS nodes, and Gazebo environment—publicly available for reproducibility (Section 3.7). Simplicity and modularity were central design principles, and the minimal dependencies (TensorFlow Lite, MediaPipe, ROS Noetic) enable easy adaptation by robotics learners, hobbyists, or HRI designers. Containerization with Docker streamlined setup and ensured consistent performance across environments, though initial integration required significant effort.

The dual-branch fusion approach was particularly effective, as dynamic steering gestures were reliably gated by the static “Holding Wheel” detection, significantly reducing false positives and improving overall control stability.

Future work could enhance this system substantially. Expanding the dataset with users from various backgrounds and more varied gestures, such as pointing a finger to signal a turn or raising two fingers to toggle lights, would enrich the interaction vocabulary. Implementing continuous proportional control instead of fixed velocity steps would improve steering fluidity and user experience. Two-handed gestures, tested initially on CPU and found computationally intensive, could be revisited with GPU acceleration without requiring extensive retraining, thereby adding expressiveness and intuitive control options. Additionally, migrating from a differential-drive to an Ackermann-steered vehicle, initially avoided due to time and complexity constraints, would align the gesture control interface more naturally with user expectations.

Ultimately, this project confirms that careful design choices can make vision-based teleoperation accessible, responsive, and practical—even without specialized hardware such as depth sensors or wearable devices—demonstrating significant promise for broad, cost-effective deployment.

References

[1] Jiahao Chen. Gesturemoro: Gesture-based mobile robot teleoperation with leap motion, 2024. Accessed: 2024-05-30. 2

[2] Parthan Olikkal, Dingyi Pei, Bharat Kashyap Karri, Ashwin Satyanarayana, Nayan M Kakoty, and Ramana Vinjamuri. Biomimetic learning of hand gestures in a humanoid robot. *Frontiers in Human Neuroscience*, 18: 1391531, 2024. 3

[3] JL Raheja, M Minhas, D Prashanth, T Shah, and A Chaudhary. Robust gesture recognition using kinect: A comparison between dtw and hmm. *Optik*, 126(11-12): 1098–1104, 2015. 1, 2, 3

[4] OSU Robotics. Turtlebot imu glove controller, 2021. Accessed: 2024-05-30. 2

[5] ROMR Team. Imu-based teleoperation for ros mobile robots, 2022. Accessed: 2024-05-30. 2

[6] Huy Trinh. Hand gesture teleoperation with ros and mediapipe, 2023. Accessed: 2024-05-30. 1, 2, 3

[7] Lucas Alexandre Zick, Dieisson Martinelli, André Schneider de Oliveira, and Vivian Cremer Kalempa. Teleoperation system for multiple robots with intuitive hand recognition interface. *Scientific Reports*, 14(1):1–11, 2024. 2, 3

A. Physical Setup CAD and Images

This appendix presents the physical hardware setup used for collecting steering gesture data. It includes 3D CAD views of both the wheel interface and the base mounting structure. The physical rig was 3D-printed and clamped to a stable vise to ensure consistent positioning.

Steering Wheel Interface

The steering wheel used in the capture setup is designed with ergonomic grips and a central hub. Figures 13 and 14 illustrate the isometric and side views.

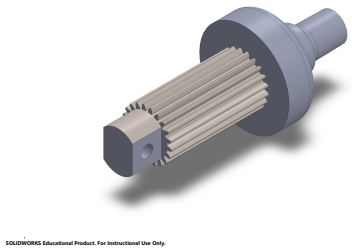


Figure 13. Steering wheel – Isometric view.

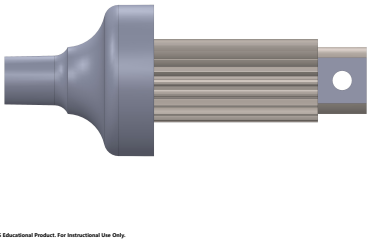


Figure 14. Steering wheel – Side view.

The physical prototype went through an iteration step from raw 3D print to a finalized painted version. Figures 15 and 16 illustrate this transition.

Figure 15. Unpainted steering wheel

Figure 16. Final painted version used during data collection.

Mounting Base

To maintain a consistent camera–hand distance and angle, the wheel was attached to a base rig clamped in a bench vise. Figures 17–19 show the CAD drawings.

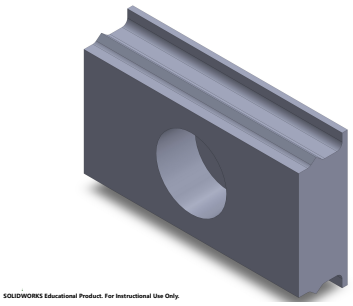


Figure 17. Mounting base – Isometric view.

Figure 18. Mounting base – Side view.

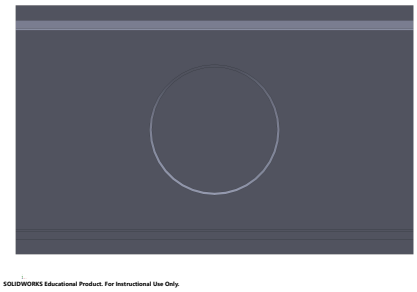


Figure 19. Mounting base – Front view.

B. System Specifications

Desktop Environment (GPU Host)

- **Model:** Micro-Star International Co., Ltd. MS-7D67
- **Processor:** AMD Ryzen™ 9 7900X (24 threads)
- **Memory:** 64 GiB
- **Graphics:** NVIDIA GeForce RTX 4060 Ti
- **Storage:** 4 TB SSD

- **OS:** Ubuntu 24.04.2 LTS (64-bit)
- **Kernel:** Linux 6.11.0-26-generic
- **Display Server:** Wayland, GNOME 46

Lenovo ThinkPad E14 (CPU Host)

- **Model:** Lenovo ThinkPad E14 Gen 2
- **Processor:** Intel Core i5-10210U (4 cores, 8 threads)
- **Memory:** 16 GiB DDR4
- **Graphics:** Intel UHD Graphics
- **Display:** 14" FHD IPS (1920×1080)
- **Storage:** M.2 NVMe SSD
- **OS:** Ubuntu 20.04.6 LTS

C. Previous Static Gesture Model – Accuracy, Loss, and Confusion Matrix

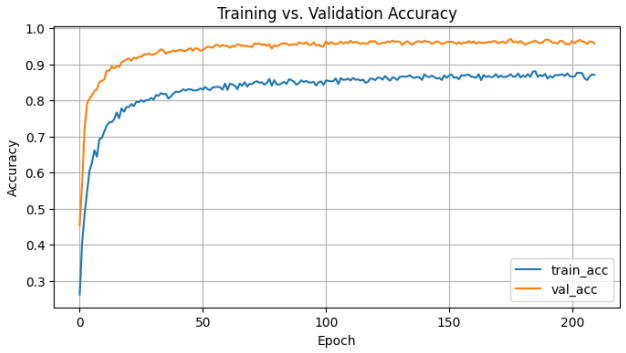


Figure 20. Training and validation accuracy – Static gesture model (Keypoint MLP).

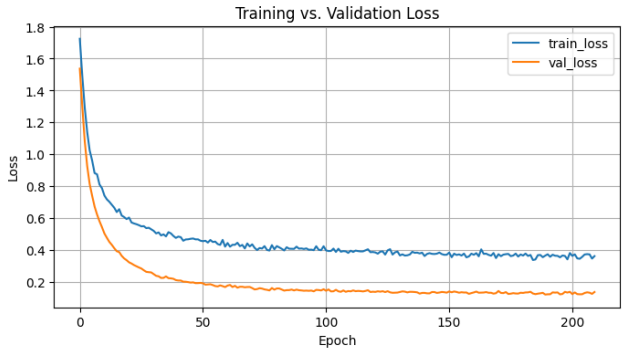


Figure 21. Training and validation loss – Static gesture model (Keypoint MLP).

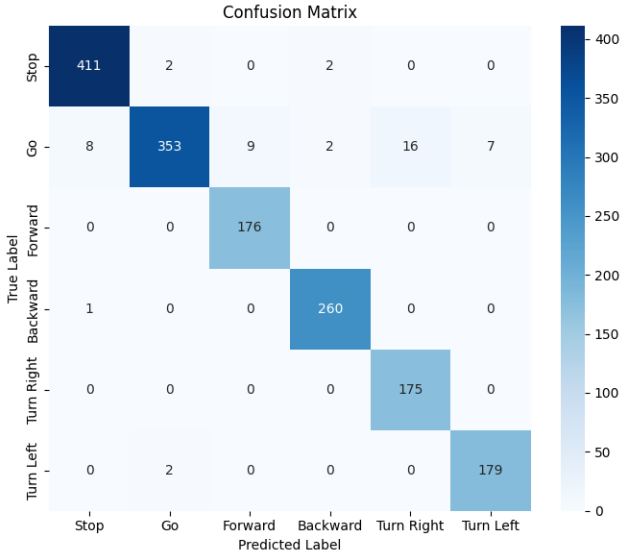


Figure 22. Confusion matrix – Static gesture model (Keypoint MLP).

D. Full Classification Reports

Static Gesture Classifier (Keypoint MLP)

Gesture	Precision	Recall	F1-Score	Support
Stop	0.99	0.98	0.99	415
Go	0.99	0.92	0.95	395
Forward	0.97	1.00	0.98	176
Backward	0.98	1.00	0.99	261
Turn Right	0.92	1.00	0.96	175
Turn Left	0.96	0.99	0.97	181
Macro Avg	0.97	0.98	0.97	1603
Weighted Avg	0.98	0.97	0.97	1603
Accuracy			**0.97**	1603

Table 9. Classification report – Static gestures (Keypoint MLP model).

Dynamic Gesture Classifier (Point History LSTM)

Gesture	Precision	Recall	F1-Score	Support
Turn Left	0.99	0.99	0.99	195
Turn Right	0.99	1.00	0.99	178
Forward	0.97	0.99	0.98	172
Macro Avg	0.98	0.99	0.98	545
Weighted Avg	0.98	0.99	0.98	545
Accuracy			**0.98**	545

Table 10. Classification report – Dynamic gestures (Point History LSTM model).

E. Model Architectures

Static Gesture Classifier { Keypoint MLP

Layer (type)	Output Shape	Param #
Dropout	(None, 42)	0
Dense (20 units)	(None, 20)	860
Dropout	(None, 20)	0
Dense (10 units)	(None, 10)	210
Dense (4 classes)	(None, 4)	44
Total		**1114** (4.35 kB)

Table 11. Model architecture – Static Keypoint MLP.

Dynamic Gesture Classifier { Point History LSTM

Layer (type)	Output Shape	Param #
Reshape	(None, 16, 8)	0
Dropout	(None, 16, 8)	0
LSTM (32 units)	(None, 32)	5248
Dropout	(None, 32)	0
Dense (32 units)	(None, 32)	1056
Dropout	(None, 32)	0
Dense (3 classes)	(None, 3)	99
Total		**6403** (25.01 kB)

Table 12. Model architecture – Dynamic Point-History LSTM.