Human-Following Robot

Robotics Final Project Report

Pranav Srivastava[1], Wisal Alaz[1], Yousef Swed[1], Kevin Bretz[1], Laurens Grote[1]
[1]Leiden University

Emails:

May 27, 2025

Abstract

This technical report presents the design, implementation, and evaluation of an autonomous robot car tailored for object and person tracking. The design addresses the computational limitations of resource-constrained robotic systems, particularly the Raspberry Pi's inability to run complex deep learning models like under battery power. To overcome this, a distributed client-server architecture was developed, where the Raspberry Pi manages real-time motor control and live video streaming, while an off-board laptop performs the computationally intensive object detection. Empirical results demonstrate significant performance enhancement; the distributed system consistently achieved live video stream processing at 20 FPS, while the initial onboard processing attempts achieved frame rates of 5 FPS, only with lightweight models.

1 Introduction

The field of robotics is rapidly evolving, with applications stretching across various domains such as industrial automation and healthcare [1]. This project focuses on autonomous navigation of a robot car. Specifically, a system tailored to follow a given object or person. The primary motivation behind this work stems from the growing demand for intelligent mobile platforms that can enhance personal assistance, improve logistics efficiency, or provide new forms of human-robot interaction in dynamic environments.

Our objective is to develop an object-following capability for a robot car, with a priority for human detection. This report details the design, implementation, results, and evaluation of our results. Finally, we discuss the implications and significance of our results and challenges, and suggest future work.

2 Novelty

While the concept of an object-following robot using visual detection is well-researched [2], the novelty of our project lies in its practical implementation choices that addresses common challenges in low-powered robotic systems.

One key implementation choice is a distributed processing architecture. While the Raspberry Pi handles camera live streaming and motor control, we use a laptop to handle the computations for object detection and control logic. In order to separate the logic and send information from and to the Pi, we utilize MQTT-based communication. MQTT is a standards-based messaging protocol that is used for machine-to-machine communication [3]. It is often used in Internet of Things (IoT) applications, where devices typically transmit data over a resource-constrained network. It's implementation requires minimal resources on the Pi, while facilitating the necessary data communication. By dividing tasks between the laptop and the Pi, the more computationally intensive tasks can be performed on the laptop, while simpler tasks are performed on the Raspberry Pi.

3 Related Work

Object tracking and following in robotics has seen significant advancements, backed by the growing need for autonomous systems in applications such as surveillance, logistics, and assistive technologies [14].

One relevant study, "Object Tracker and Follower Robot using Raspberry Pi" [4], details a robot system designed for monitoring objects based on visual input. This system utilizes a Raspberry Pi to process images captured by an attached camera, allowing the robot to autonomously follow objects based on the captured photos. The implementation uses OpenCV for image processing, identifies objects by their shape or color, and then adjusts the robot's movement according to the position of the detected object. The authors highlight its cost-effectiveness and simplicity compared to other comparable robots. Other research has explored neural network controllers for visual-based object tracking [5], adaptive moving-target tracking using fuzzy neural networks for mobile robots [6], and person tracking using 2D laser scanners [7].

While these works demonstrate capabilities in object tracking, a common limitation in many traditional visual-based systems, including that of Sharma et al [4], is their reliance on processing photos, which can lead to latency in dynamic environments. Furthermore, systems that identify objects based on specific "shape or colour" may lack the flexibility to recognize a diverse range of objects without explicit re-programming. We overcome these limitations by detecting objects using a deep learning model, rather than relying on heuristics such as shape or colour. The implementation of the model is explained further in the following sections.

4 Design

This section outlines the architectural design of our object-following robot car system. We detail the hardware components selected, the software architecture, and the rationale behind key design decisions, particularly the distributed processing approach.

Initial attempts to perform object-detection directly on the Raspberry Pi resulted in many performance limitations. While connected to a power source via cable, the object-detection code executed on the Pi was able to recognize the desired object. However, the performance was limited by low frame rates (1-2 FPS) and unresponsiveness. Crucially, the Pi consistently crashed and did not execute the code under battery power, due to overheating and insufficient computational resources. Consequently, the system was redesigned into a client-server architecture to offload computationally intensive tasks to an external laptop, enabling the Raspbery Pi to focus on real-time motor control and live stream data communication. This design allowed us to circumvent the performance limitations of the Pi, and successfully utilize the deep learning models by YOLO for object detection.

4.1 Hardware Design

The hardware foundation of our project is the Freenove 4WD Smart Car Kit (FNK0043)[8], chosen for its robust 4-wheel drive platform and compatibility with the Raspberry Pi. This kit serves as the **Robot Car Platform**, featuring for DC TT motors, a chassis, and an included motor driver board, powered by rechargeable batteries. A Raspberry Pi 5 functions as the **Primary Processing Unit (On-board)**, managing motion control, camera sensor data acquisition, and network communication. Simultaneously, a laptop acts as the **Secondary Processing Unit (Off-board)**, dedicated to real-time object detection. This allows the usage of without the computational or power constraints of the Raspberry Pi. For visual input, a **Raspberry Pi Camera V3 (Wide angle)** captures and streams live video to the laptop for processing. The Raspberry Pi 5 interfaces directly with the motor driver and the Pi Camera V3, while also managing communication with the laptop.

4.2 Software Architecture

The software architecture uses a modular and distributed design, separating perception and control functionalities. The **Video Streaming Module** on the Pi captures video frames from the Pi Camera V3 and streams them via Wi-Fi to the laptop. Implemented using Flask, the video resolution is down scaled to 320x240 pixels from 640x480 pixels to enhance streaming stability. The **Perception Module**, executed on the laptop, receives this live stream data. It uses the object detection model to identify the target object and determine its relative position within the camera frame. Based on the target's

bounding box coordinates, pre-defined left and right threshold values, and the camera frame center, as seen in Figure 1.

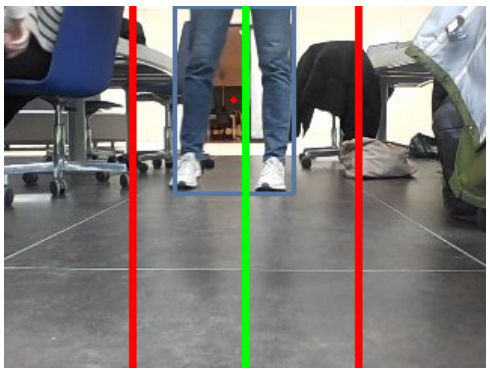

Figure 1: A screenshot demonstrating successful human detection by the model, showing the bounding box around the target.

the module calculates a directional command (forward, left, right, or stop) and sends the command to the Raspberry Pi via MQTT along with the bounding box data. The **Control Module**, running on the Pi, receives these directional commands and bounding box data from the laptop's Perception Module. This module integrates target following logic by interpreting commands from the laptop to actuate the robot's motors, guiding it toward the detected object. Finally, the **Communication Infrastructure** allows data exchange between the laptop and Pi using MQTT. The data flow involves the Pi streaming live video, the laptop processing it with and publishing detection results (bounding box, movement commands) to an MQTT topic, and the Pi subscribing to this topic to receive commands for determining motor actions.

This distributed software architecture enabled the use of advanced deep learning models, specifically YOLOv5, while ensuring reliable perception and efficient real-time robotic control and data communication.

5 Implementation

This section elaborates on the practical aspects of programming our project. The hardware was pre-built, hence, no assembly was required.

5.1 Software Development

The software is primarily developed in Python, using libraries such as RPi.GPIO, PyTorch [9], OpenCV [10], and paho-mqtt. Object detection is performed using the model. The camera frame is segmented into three logical zones (left, center, and right) to determine movement decisions. All project code is accessible in the attached zip file of our submission.

5.2 Software Setup and Installation

It is assumed that the Freenove tutorial and setup instructions are already applied [11]. Required python libraries on the Raspberry Pi are installed using apt to ensure compatibility. Key libraries include python3-flask, python3-gpiozero, python3-libcamera, python3-numpy, python3-opencv, python3-paho-mqtt, python3-picamera2, and python3-torch. The full library installation list can be found in the appendix section 9.2.

As systemd service is configured on the Raspberry Pi (e.g., at /etc/systemd/system/camera_stream.service) to ensure continuous operation of the camera stream as seen in appendix section 9.3.

This service, configured to run Gunicorn with the stream_camera.py application, is activated via sudo systemct1 enable camera_stream ––now. An MQTT broker (Mosquitto)[12] is deployed on the Raspberry Pi to facilitate inter-device communication. The main control program on the Pi is executed via python pi_control.py.

For the laptop setup, dependencies are managed using a requirements.txt file, installed in a virtual environment via the pip install –r laptop/requirements.txt command. The GUI, as seen in Figure 2,

can then be launched with python ui.py, which allows for the selection of 82 different detectable objects. Notice that this was only tested on linux-based systems, in particular Arch-Linux and Ubuntu. The used python version was 3.13.

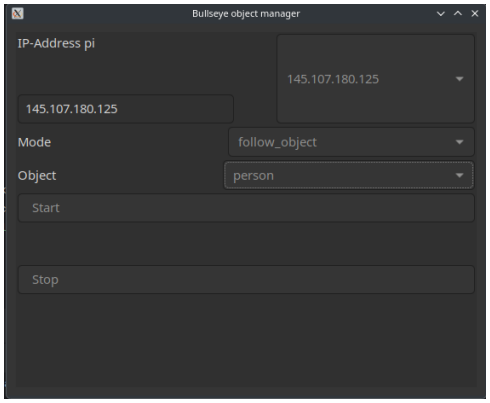

Figure 2: Object detector UI

5.2.1 Interface Specification

Communication between the Pi and the laptop is defined by specific MQTT topics. The laptop publishes messages on two topics: /pi/control/command and /pi/object/info, in a predefined JSON format. The /pi/control/command topic sends control instructions from the laptop to the Raspberry Pi as seen in appendix section 9.4.1.

The /pi/object/info topic provides object detection results from the laptop to the Pi, specifically, bounding box coordinates for detected objects as seen in appendix section 9.4.2.

5.3 Project Structure and Extensibility

The project's modular directory structure allows for maintainability and scalability. The common_objects directory contains data transfer objects (DTOs) used by both the Pi and laptop modules. The laptop directory encapsulates all laptop-specific code, including the YOLO detection logic, an MQTT handler (mqtt_handler) for outgoing messages, and human_follow_tracking.py for human tracking. The pi directory contains all the car control functions, including app.py for processing laptop commands and creating controller instances, and the car_control subdirectory for the direct car control logic.

A particular challenge involved ensuring that the video stream provided the most recent frame rather than a buffered sequence, which caused significant delays. This was addressed by implementing a separate Pi to continuously poll the network stream and provide the application with the latest frame, which helped optimize the responsiveness of the robot.

6 Results

This section presents the results obtained from testing and evaluating our robotic system.

6.1 Experimental Setup

To evaluate the functionality of the object-following robotic system, tests were conducted in classrooms and hallways as they provided a controlled, indoor environment. A human target or other standing objects was placed in front of the robot for each experiment. They included stationary objects like cups or balls as well as a moving individual moving at different angles and speeds. The objective was to test the robot's ability to track, detect, and follow the target while moving straight ahead or changing course as necessary.

6.2 Performance Evaluation

The empirical results demonstrate a significant performance enhancement achieved by the distributed architecture. The live video stream, processed by the Perception Module on the laptop, consistently

yielded frame rates between 20-30 FPS. This represents a noticeable improvement compared to the initial monolithic approach, where hosting and executing all modules directly on the Pi resulted in severely limited frame rates of 1-2 FPS. The difference in frame rates indicates that the distributed approach worked successfully, by off-loading the computationally intensive tasks to the laptop.

Qualitatively, the robot was able to follow the target consistently. Specifically, in the human detection trial, the robot detected the target human with little waiting time from the time the target detection script was started. Furthermore, the robot successfully navigated left and right, following the human as they moved side to side.

A benefit of using a pre-trained deep learning model form YOLO, was that it enabled robust object detection. In environments with shadows, and reflections on walls, the robot still managed to consistently detect the human and follow without deviating towards other objects.

One limitation with this approach, is that the human was limited in how fast they could move. If the human moved too quickly to the left or right of the robot, the robot would sometimes struggle to detect the human in the corner view.

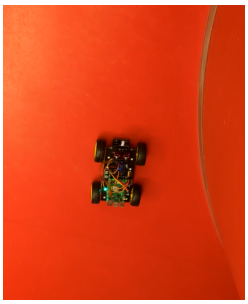

Figure 3: The robot car during human following, demonstrating its physical setup and operational context.

A short demo video of our project can be found here: https://www.youtube.com/shorts/Kq2Soc3KUVA

7 Discussion and Future Work

While the current implementation successfully achieves robust object detection and target following, certain limitations motivate the need for future improvements. Firstly, the system's performance is reliant on a stable WiFi network to ensure smooth live streaming of the the Pi's camera, and the availability of an off-board laptop. These two constraints limit the environments in which such a robot could be used. Consequently, alternatives could be further researched, such as optimizing and fine-tuning existing lightweight models to detect very specific objects, which would keep computational complexity low, and potentially allow a monolithic approach with all modules executed solely on the Pi.

Secondly, an object avoidance algorithm could be implemented to ensure that the robot can overcome obstacles while still following the specified target. This would open up possibilities for practical applications, such as search and rescue operations in rubbles caused by building collapses.

Finally, a crucial improvement that could be implemented with minimal additional work, is to continue the object-tracking even when the object is out frame. One approach for this implementation could be to continue turning in the direction that the object was last seen in frame. For example, if the object was last seen on the left side of the frame, the robot would continue to move leftward to detect the object again.

8 Conclusion

This project successfully designed and implemented a robot car with object follow-up, with human follow-up as the priority. It addressed power limitation issues by using a distributed module approach, where computationally heavy tasks were off-loaded to a laptop, allowing the Pi to focus on live streaming and motor control. This approach yielded smooth frame rates of up to 30 FPS, resulting in reliable target tracking and motor control. The developed robotic system demonstrates a practical solution for integrating advanced deep learning capabilities into resource-constrained robotic systems.

References

[1] Periyannan Ramamoorthy, S. (2024). Role of AI, Automation & Robotics in Pharmaceutical Industry. *Journal of Next-Generation Research 5.0*, *1*(1). [Online]. Available: https://doi.org/10.70792/jngr5.0.v1i1.45.

[2] Ambikapathy, A., Sandilya, J., Tiwari, A., Singh, G., & Varshney, L. (2021). Analysis of Object Following Robot Module Using Android, Arduino and Open CV, Raspberry Pi with OpenCV and Color Based Vision Recognition. In: Priyadarshi, N., Padmanaban, S., Ghadai, R.K., Panda, A.R., & Patel, R. (eds), *Advances in Power Systems and Energy Management* (Lecture Notes in Electrical Engineering, vol 690). Springer, Singapore. [Online]. Available: https://doi.org/10.1007/978-981-15-7504-4_35.

[3] Amazon Web Services. (2025). What is MQTT?. [Online]. Available: https://aws.amazon.com/what-is/mqtt/.

[4] Sharma, S., Abrol, A., Khajuria, S., Sharma, S., & Gupta, M. (2023). Object Tracker and Follower Robot using Raspberry Pi. *International Journal of Creative Research Thoughts (IJCRT)*, *11*(1). [Online]. Available: https://www.ijcrt.org/.

[5] Risma, P., Dewi, T., Oktarina, Y., & Wijanarko, Y. (2019). Neural Network Controller Application on a Visual based Object Tracking and Following Robot. *Computer Engineering and Applications Journal*, *8*(1). [Online]. Available: https://www.researchgate.net/publication/330849802.

[6] Wai, R.-J., & Lin, Y.-W. (2013). Adaptive Moving-Target Tracking Control of a Vision-Based Mobile Robot via a Dynamic Petri Recurrent Fuzzy Neural Network. *IEEE Transactions on Fuzzy Systems*, *21*(4). [Online]. Available: https://doi.org/10.1109/TFUZZ.2012.2227974.

[7] Leigh, A., Pineau, J., Olmedo, N., & Zhang, H. (2015). Person tracking and following with 2D laser scanners. In *2015 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 726-733). [Online]. Available: https://doi.org/10.1109/ICRA.2015.7139259.

[8] Freenove. (2025). Freenove 4WD Smart Car Kit for Raspberry Pi (FNK0043). Freenove Store. https://store.freenove.com/products/fnk0043.

[9] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems, 32.

[10] Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.

[11] Freenove. (2025). Freenove 4WD Smart Car Kit for Raspberry Pi Software. GitHub. https://github.com/Freenove/Freenove_4WD_Smart_Car_Kit_for_Raspberry_Pi.

[12] Random Tutorial Nerds (2000). Install Mosquitto MQTT Broker on Raspberry Pi. https://randomnerdtutorials.com/how-to-install-mosquitto-broker-on-raspberry-pi/

[13] OASIS. (2019). MQTT Version 5.0. OASIS Standard. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html.

[14] Adib Bin Rashid. (2024). AI revolutionizing industries worldwide: A comprehensive overview of its diverse applications. [Online]. Available: https://www.sciencedirect.com/science/article/pii/X24001386?via%3Dihub

9 Appendix

9.1 IMPORTANT: Hardware Failure

We were unable to retrieve proper quantitative metrics and detailed demo videos for our evaluation due to our robot not working anymore. We have communicated this with Mr. Bakker, and he has allowed us to include detailed instructions on how to run the code, in place of the demo videos. We have included some demo videos nevertheless, which we thankfully took before the live demos last week.

9.2 Full Python Library Installation List

The following libraries need to be installed on the pi, in addition the one of the freenove setup.py:

```
sudo apt install python3-click
sudo apt install python3-flask-cors
sudo apt install python3-flask-socketio
sudo apt install python3-flask
sudo apt install python3-gi-cairo
sudo apt install python3-gi
sudo apt install python3-git
sudo apt install python3-gpiozero
sudo apt install python3-gunicorn
sudo apt install python3-kms++
sudo apt install python3-libcamera
sudo apt install python3-libgpiod
sudo apt install python3-matplotlib
sudo apt install python3-numpy
sudo apt install python3-opencv
sudo apt install python3-paho-mqtt
sudo apt install python3-pgzero
sudo apt install python3-picamera2
sudo apt install python3-pigpio
sudo apt install python3-pyqt5
sudo apt install python3-rpi-lgpio
sudo apt install python3-serial
sudo apt install python3-smbus2
sudo apt install python3-spidev
sudo apt install python3-tk
sudo apt install python3-torch
```

9.3 Systemd Service Configuration

```
[Unit]
Description=Flask Camera Network Share
After=network.target

[Service]
User=bullseye
WorkingDirectory=/home/bullseye/final_deployment/src/pi
ExecStart=/usr/bin/python3 -m gunicorn --timeout 0 -w 1 -b 0.0.0.0:5000 stream_camera:ap
Restart=always

[Install]
WantedBy=multi-user.target
```

9.4 MQTT Topic Format

9.4.1 Control Command

```
{
    "mode": "follow_position" | "move_head" | "find_object",
    "recommended_command": stop| forward | left | right
    "command_intensity": 0-100: 0: low; 100: high
    "objectSize": 32,
}
```

9.4.2 Object Information Command

```
{
    "objects": [
        "ObjectName1": {
            "leftUpCorner": {
                "x": 342,
                "y": 234
                },
            "rightDownCorner": {
                "x": 342,
                "y": 234
                },
            "ObjectName2": { ... }
    ]
}
```