Autonomous Person-Following Robot

Jordan Cousineau
Boston University
44 Cummington Mall
jordache@bu.edu

Anthony Southmayd
Boston University
44 Cummington Mall
as0317@bu.edu

Abstract

This work presents an autonomous robot designed to follow a designated person using computer vision, with a simple obstacle avoidance mechanism that halts movement when a blockage is detected. The system integrates several components: the "You Only Look Once" (YOLO) object detection framework, Scale-Invariant Feature Transform (SIFT) for feature matching, a Raspberry Pi 3, a Raspberry Pi camera, an Arduino microcontroller, ultrasonic sensors, and drive motors. Given the computational limitations of the hardware, we hypothesized that using grayscale input instead of RGB would improve performance, and that applying SIFT to cropped regions rather than full frames would reduce processing time. To validate these assumptions, we trained and tested multiple configurations of our computer vision pipeline. This paper details the design, training, and evaluation process for deploying a lightweight, real-time person-following robot using embedded hardware.

1. Introduction

Real-time person-following robots are increasingly used in service, delivery, and assistive applications. However, deploying such systems on low-power, embedded platforms like the Raspberry Pi 3 presents several challenges, especially in terms of computer vision processing due to its computationally expensive process.

This paper addresses those challenges by presenting a cost-effective, lightweight robot that follows a designated human using the YOLOv8 object detection framework and SIFT feature matching. To enable deployment on constrained hardware, we explore performance trade-offs between grayscale and RGB image inputs, and evaluate the effects of full-frame versus cropped SIFT computation. Obstacle avoidance is implemented in a minimal form, where the robot halts when blocked rather than rerouting.

Our contributions include (1) an embedded vision-based tracking system using YOLOv8 and SIFT, (2) an empirical study on detection performance across color formats and image regions, and (3) an analysis of frame rate optimization strategies.

2. Hardware

2.1. Overview

The hardware used, as seen in Figure 1, includes a robotic car kit, an Arduino, a Raspberry Pi 3, and a camera module. The Arduino acts as a hardware controller for the low-level 5V portion of the system, which is composed of the motors and the ultrasonic sensor. The Pi acts as a high-level controller which is responsible for the camera module and the general control logic. The Pi sends control messages containing motor power values to the Arduino via a serial connection. The Arduino will send a signal to the Pi using a digital out pin when the ultrasonic distance is below a specified threshold. The decision to use a digital pin in addition to the serial connection resulted from the desire to have a quicker response time in case of obstacle collision. With the signal going over a digital pin, the Pi can be configured to throw an interrupt and immediately take action, instead of waiting for a buffer to be read with the serial protocol.

2.2. Control Logic

The control flow of the Arduino and Raspberry Pi can be seen in Figures 2 and 3, respectively. The Arduino code has a serial read stage, an ultrasonic stage, and a motor stage. The serial stage of the code checks whether the Pi has sent new motor values. If new values have been sent, the Arduino converts the three bytes sent into a right and left motor value; otherwise, the existing motor values will be used. Once any serial data has been read, the Arduino checks the distance of the ultrasonic sensor and compares it against a set threshold. When the distance is below the threshold, i.e. an obstacle is in front of the robot, then the motors will be turned off and the digital out pin will be turned on to inform the Pi. If there is no obstacle detected, the motors will be set to the values received in the serial stage.

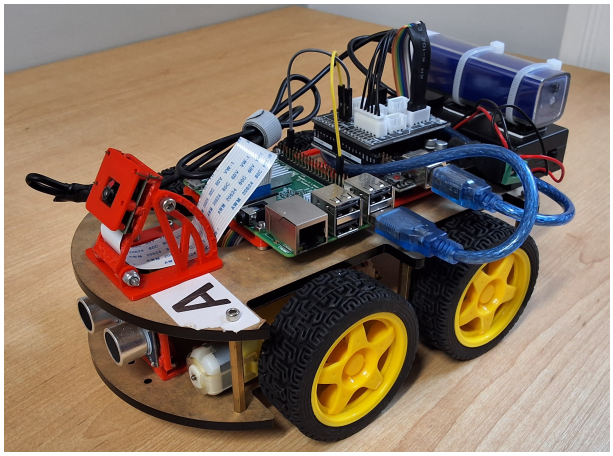

Figure 1. Robot Car Hardware

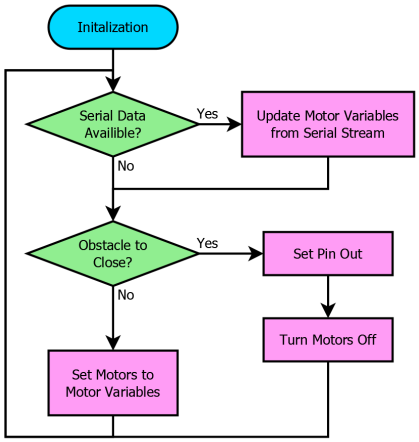

Figure 2. Arduino Logic Flow

The Pi control flow is more complex than the Arduino code, due to the Pi acting as a high-level controller. The Pi starts by retrieving a frame from the camera and applying preprocessing steps, such as resizing and shape reformatting, to prepare the image for the machine learning model. The ML model will then predict the bounding boxes of any people in the image. These predictions are then filtered to remove any duplicate boxes and any that fall below a confidence threshold. SIFT is then used on the filtered results to identify whether there is a target and the position of the target. The robot will try to maneuver so that the target individual, which is the first person seen based on saved SIFT features, is in the center of the camera. If no target is found, then either the robot hasn't seen a person yet or the person has moved out of frame. If a person hasn't been seen yet, the robot will remain stationary until a person is identified and their features saved as the target. If the person has moved out of frame, the robot will continue its path, assuming the person is out of frame in the direction the robot is moving. An additional portion of the logic, which is not present in Figure 3, will clear any saved person if they have been out of frame for too long. This ensured that the robot wouldn't get permanently stuck during testing. Finally, although the Arduino sends a signal indicating an obstacle to the Pi, the Pi doesn't currently apply any logic related to the signal. However, the system includes the capability for possible future applications.

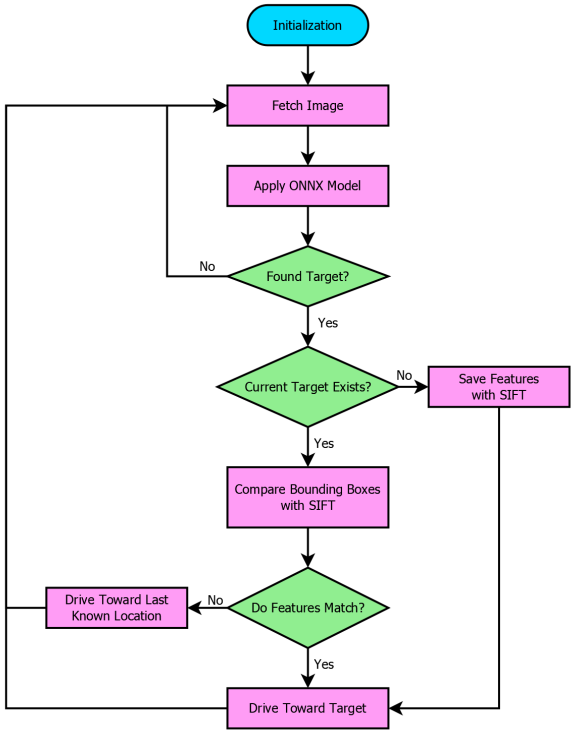

Figure 3. Raspberry Pi Logic Flow

2.3. Limitations

Over the course of this project we encountered several hardware limitations. The major limitation we ran into was the limited computing power of the Raspberry Pi 3. The Pi 3 is a couple generations old and limited to 1GB of RAM. We were initially concerned about RAM capacity, but we believe the CPU bandwidth ended up being more of a bottleneck and resulted in the Pi processing at approximately 1 FPS. This issue would most likely be greatly reduced with more current hardware, due to the current generation having a CPU that runs twice as fast with additional improvements. The other two semi-major limitations we ran into both involved the camera module. The module we are using is the original revision, released in 2013, of the Pi camera module. While the module was able to produce images with enough resolution, a newer revision would most likely have

more efficient hardware. The other camera limitation involves updates to the software stack. The Linux library that is used to capture images received a major revision change fairly recently and OpenCV, the library we are using for image processing, hasn't become fully compatible yet. This prevented us from testing the robot with a live camera view, which ultimately limited our testing capabilities.

3. Software

3.1. YOLO

YOLO was chosen because it is a "state of the art, real-time object detection" framework and is able to "achieve high speeds and accuracy" [5]. Its effectiveness in embedded real-time systems has also been demonstrated in applications such as autonomous vehicle steering using YOLOv8 for object tracking [4]. We ended up training two YOLOv8 nano models: one that was trained on an RGB dataset, and another that was trained on a grayscale dataset.

3.1.1. Training the YOLO Models

Both the RGB and grayscale model were trained on the 2017 COCO dataset [2], specifically a subset of the dataset. The subset contained roughly 30k images, where 85 percent of the images contained at least one human and the last 15 percent did not contain humans. This was done to improve the training time of the models, and to decrease any overhead that detecting other objects would present during live usage of the model.

3.1.2. Evaluation of the YOLO Models

To evaluate the YOLO models, we exported them to the Open Neural Network Exchange (ONNX) format, which supports efficient usage across a range of "hardware and software environments" [1]. The models were deployed and tested in a Python script running on a laptop. We measured performance using frames per second (FPS), calculated by timing the execution of both human detection and SIFT processing using timestamps captured before and after each iteration:

$$\frac{1}{endTime - startTime} \tag{1}$$

We recorded FPS values over a 10-second window during live detection and computed the average to assess each configuration's performance.

The results of this comparison revealed that the grayscale model performed approximately 2.4% better than the RGB model. However, this improvement was not considered significant enough to significantly improve performance on the Raspberry Pi.

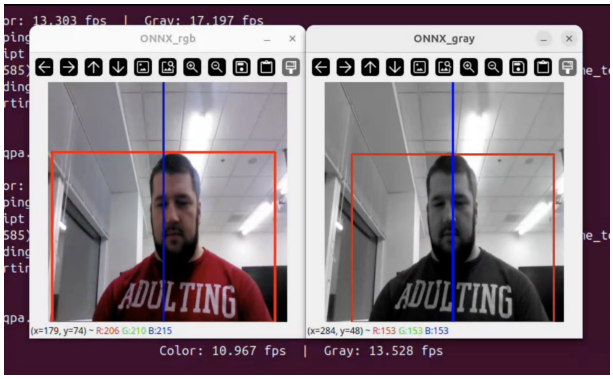

Figure 4. Live Model detecting human (Red Border) with SIFT Detection (Blue Line) showcasing FPS

Trial	RGB	Grayscale
1	29.656	31.031
2	31.375	31.14189723
3	21.13589979	24.560131
4	34.74301284	30.89222775
5	28.7724622	31.40157409
Average	29.13665959	29.80552804

Table 1. Results of the FPS Comparison.

3.2. SIFT

SIFT is an algorithm that detects local features in images by extracting keypoints and comparing them using Euclidean distance [3]. While the original method was designed for matching features across static images, we apply SIFT to each frame of a live video feed to track features in real time. In our implementation, we use a distance threshold of 0.8 and require a minimum of 20 matched features. When these conditions are met, we save the new keypoints to enable tracking under dynamic conditions such as turning, lighting changes, and background shifts. To evaluate the effectiveness of SIFT we tested its performance when applied to entire frames versus a cropped version of each frame.

3.2.1. Evaluation of SIFT

Similar to our model evaluations, we use frames per second (FPS) as the primary performance metric, use the same FPS calculation, and take the average of the first 10 seconds worth of FPS values. In this experiment, we compare the application of SIFT to the full video frame versus a cropped region from the bounding box of human detection. This comparison is conducted for both the RGB model and grayscale model to assess how input and frame size affect SIFT's real-time performance.

The RGB full-frame configuration outperformed the

RGB cropped configuration by approximately 47%, while the grayscale cropped configuration performed about 0.24% better than its full-frame counterpart. These RGB results were unexpected; we initially hypothesized that cropping would improve performance by reducing the number of pixels processed. This discrepancy is notable, as the results in Table 1 show significantly higher performance, whereas Table 2 reports values consistently around 20 FPS. Interestingly, this trend does not hold for the grayscale model, where cropping did lead to a slight performance improvement.

Trial	RGB Full	RGB Cropped	Grayscale Full	Grayscale Cropped
1	26.507	20.542	29.115	31.246
2	28.049	19.323	30.350	25.672
3	28.799	18.656	30.685	28.705
4	28.836	18.812	29.680	35.772
5	29.087	18.765	29.923	28.724
Average	28.255	19.220	29.951	30.024

Table 2. FPS comparison between full-frame and cropped SIFT on RGB and Grayscale inputs.

4. Conclusion

This project aimed to build a lightweight, real-time person-following robot using computer vision techniques deployable on resource-constrained hardware. By integrating YOLO object detection with SIFT feature matching, we created a functional human-following robot with minimal obstacle avoidance. We believe that this robot could be relevant in the real world assisting people in carrying or transporting items such as groceries, tools and equipment, or luggage.

4.1. Limitations

One of the most prominent limitation within this paper was the hardware used: a Raspberry Pi 3 with one gigabyte of RAM. This limited us in testing since we were unable to get a live video feed, only getting relevant info using print statements. Another limitation was the Raspberry Pi's inability to control the motors which added another component in the robot car flow, forcing data to be transferred between the Raspberry Pi and Arduino.

4.2. Future Work

For future work we believe the obstacle avoidance component could be further improved upon. It currently acts as an interrupt, successfully preventing the robot from collision; however, it is unable to make decisions that could prevent the robot from collision and continue to follow the person. Another potential branch for future work would be a more mission-specific chassis since ours provided a solid base for testing and usage, but could be much better equipped.

References

[1] Desen Bu, Bei Sun, Xiaoyong Sun, and Runze Guo. Research on yolov8 uav ground target detection based on rk3588. In *2024 2nd International Conference on Computer, Vision and Intelligent Technology (ICCVIT)*, pages 1–5, 2024. 3

[2] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. *European Conference on Computer Vision (ECCV)*, pages 740–755, 2014. 3

[3] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60:91–110, 2004. 3

[4] Joseph M. Phillips, Sam Shue, and James M. Conrad. A design architecture for steering an autonomous all-terrain vehicle using object tracking via computer vision and yolov8. In *2024 IEEE 21st International Conference on Smart Communities: Improving Quality of Life using AI, Robotics and IoT (HONET)*, pages 242–247, 2024. 3

[5] Faris Syed, Mohamed Al Musleh, and Nidhal Abdulaziz. Advancing autonomous vehicle perception: Yolo algorithms and custom dataset integration for the uae environment. In *2024 9th International Conference on Robotics and Automation Engineering (ICRAE)*, pages 18–22, 2024. 3