ML-Based Trash Collector Bot

Hanson Li
lchanggle@berkeley.edu

Jiamu Chen
jimchen@berkeley.edu

Samyak Tiwari
samyak.tiwari1679@berkeley.edu

UC Berkeley

December 14, 2023

Abstract

The ML-Based Trash Collector Bot, developed for a Berkeley EECS course, employs a YOLO v5 neural network for object detection, utilizing Raspberry Pi and camera for real-time trash identification and collection. The project integrates line following and PID algorithms for navigation, with a focus on TinyML for onboard processing. Challenges included hardware assembly and tuning, and software issues with the Adeept Tank's control system. The bot's design incorporates various sensors and actuators, connecting wirelessly to networks and leveraging SSH for remote access. References cite resources from Ultralytics, Adeept, and OpenAI's ChatGPT.

Contents

1 Introduction

1.1 Project Overview

1. **Brief description of the ML-based trash collection robot:** The robot is engineered to navigate through an environment, identify trash using machine learning algorithms, and pick it up. It employs a Yolo v5 algorithm for object detection, which is a pre-trained computer vision neural network that stands for "You Only Look Once". This is a popular choice for real-time object recognition tasks.

2. **Purpose and significance of the project:** The project aims to showcase the practical application of machine learning and robotics in performing tasks that are repetitive and labor-intensive, like trash collection. This serves both an educational purpose for students and a potential model for automating waste management.

1.2 Project Goals and Objectives

1. **High-level goals of the robot (navigation, trash identification, pickup):**
Navigation: The robot is designed to follow lines and navigate through its environment using computer vision.
Trash Identification: It uses the Yolo v5 algorithm to detect and identify trash items, which is crucial for targeted waste collection.
Pickup: The robot is equipped with a mechanical arm and servo motors to pick up identified trash.

2. **Specific objectives related to the course topics To integrate various components like cameras, Raspberry Pi, and actuators (motors and arms) for a fully functional prototype:**
To implement wireless communication protocols for remote operation and data transmission.
To train the TinyML model with pre-labeled pictures of trash and deploy it on a Raspberry Pi, which serves as the robot's brain.
To compare the effectiveness of a finite state machine (FSM) approach versus a machine learning approach for the tasks of navigation and trash collection.

2 Project Components

2.1 Architecture Diagram

1. **Visualization of the overall system architecture:** The overall architecture of the system is shown in 1.

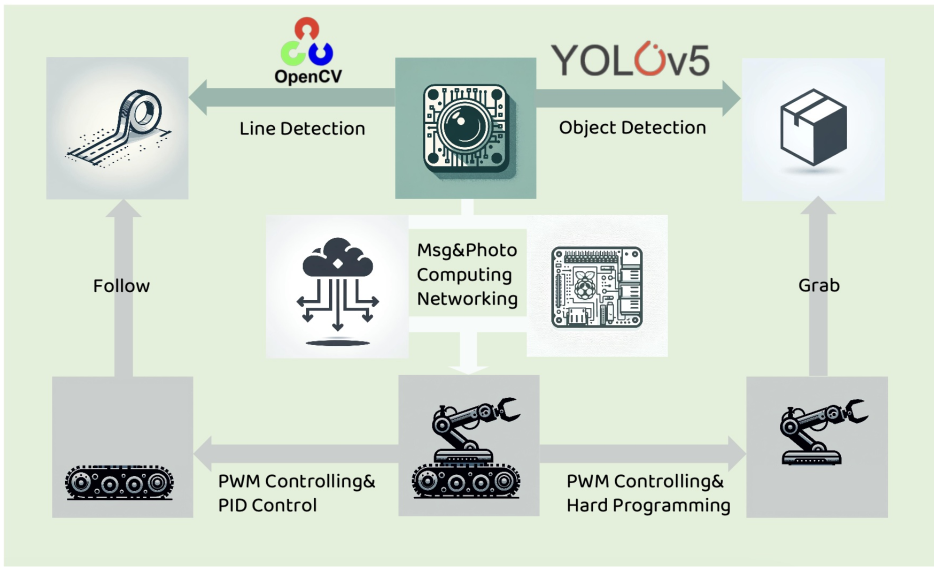

Figure 1: system architecture.

2. **Key components and their interactions:** So the main components are raspberry pi as the computing source, and linux and python as software platform, and plastic tank as hardware platform, and claws and tracks as actuators, and camera and light sensors as the sensors. The main interaction are using sensors to detect the environment, especially the tape lines and the trash object, and using actuators to follow the line and grab the trash, between which lies the fundamental PID and control coding for transferring the sensing information to control policy.

2.2 Software Components

1. **Detailed breakdown of the software modules:** The main python code consists of importing hardware package, establish threading and processing data from the sensors, planning the finite state machine, and outputing control siganls to the actuators. In the above, the sensing code includes yolov5 model, which is pre-trained on the outside computer. The raspberry pi only execute the model using the pt file. It also includes opencv library to detect the lines, but we later use light sensors to replace them.

2. **Functions and responsibilities of each component:** The importing part enables us to use the peripherals. The sensing part helps us to get environment data. The planning part acts as the computing center for deciding what the robot should do. The output part is responsible for controlling the actuators to realize the planning.

2.3 Hardware Components

1. **Overview of the robot's physical components:** The image is shown in 2

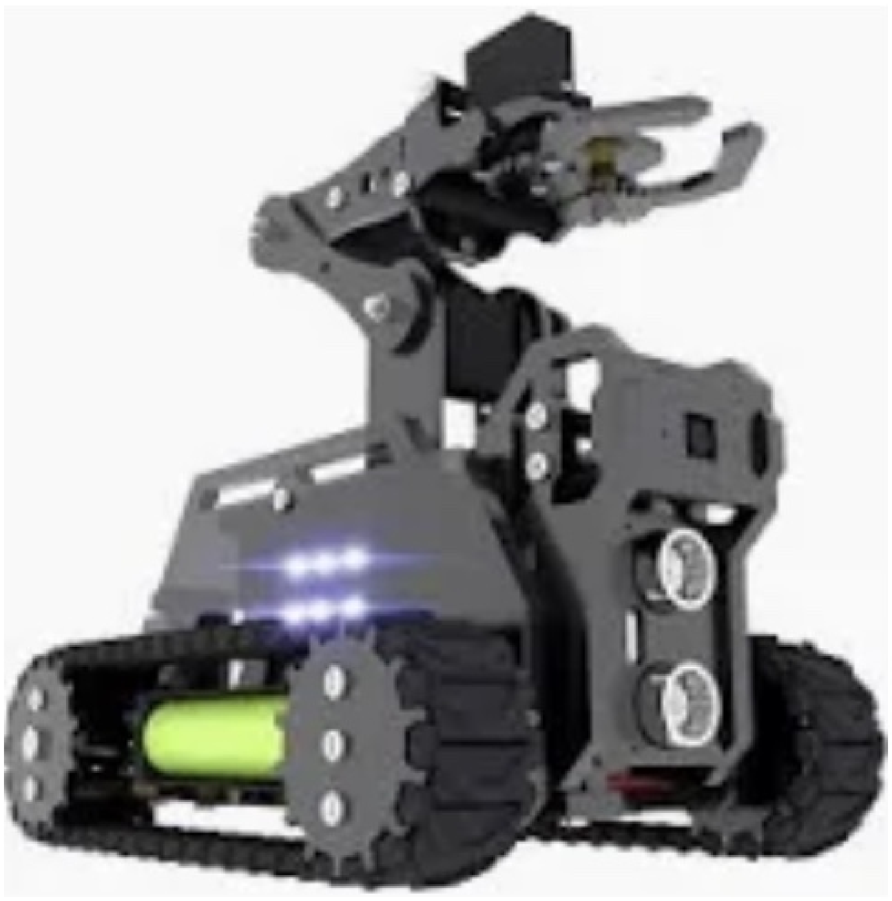

Figure 2: robot hardware overview.

2. **Connection between hardware and software components:** There is a power supply board attached to the raspberry pi. The battery is connected to the board to provide electricity. The camera, the light sensors, the motors and servos are attached to the power supply board. Also, as mentioned in the demo, the power supply seems to be a little bit unstable. The recommended voltage for raspberry pi is 5V. This is provided by the power supply board, which contains voltage stabilizing modules. The input voltage of the power supply board is 7V-15V. The two 18650 batteries provide 2*3.7=7.4V of voltage, which is a little bit inadequate. When using the adjustable power supply, setting it to 9V, the robot acts much better. But when encountering heavy loads of servos, the robot might shut down. This phenomeno needs further investigation. The information of the circuit of power supply board is shown in 3. The pdf file is also included in the github repository.

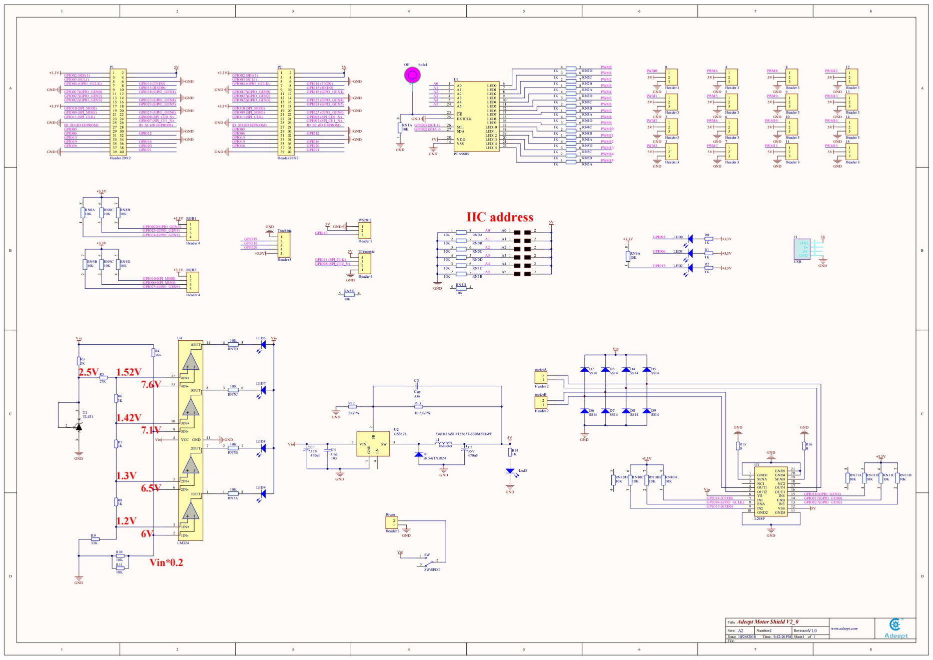

Figure 3: The power supply board circuit diagram.

3 Finite State Machines (FSM) and Course Topics

Finite State Machines (FSM) are crucial in controlling the behavior of automated systems like trash collection robots. In this project, the FSM guides the robot through various states for efficient navigation and trash collection. The image could be shown in **??** and 5

1. **FSM Utilization in the Project:** The FSM for the robot includes states such as Navigation, Trash Detection, Repositioning, and Picking Up. Transitions between these states are triggered by conditions like trash detection and successful collection. This structured approach allows the robot to systematically scan the environment, identify, and collect trash.

2. **Importance of FSM:** FSMs provide a predictable and organized method for the robot's operations. This systematic approach ensures that the robot performs tasks efficiently, adapting its behavior based on specific environmental cues. The FSM's structured nature allows for easier troubleshooting and modifications to the robot's behavior, enhancing its overall functionality and reliability.

3.1 TinyML in Trash Identification

1. **Integration of TinyML in the Trash Identification Process**

    - Pretraining Yolo-v5 on A100 GPU for efficient model training.

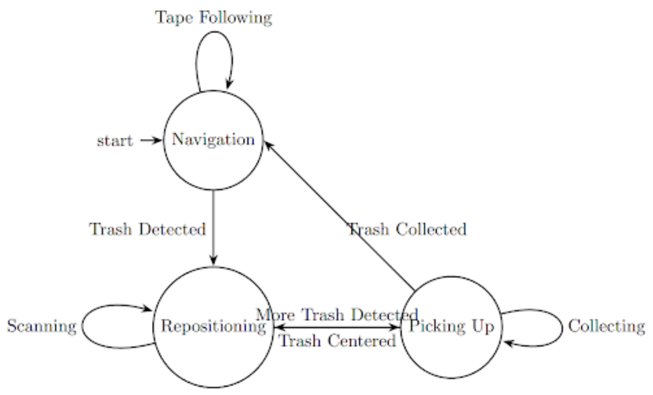

Figure 4: Finite State Machine of the System

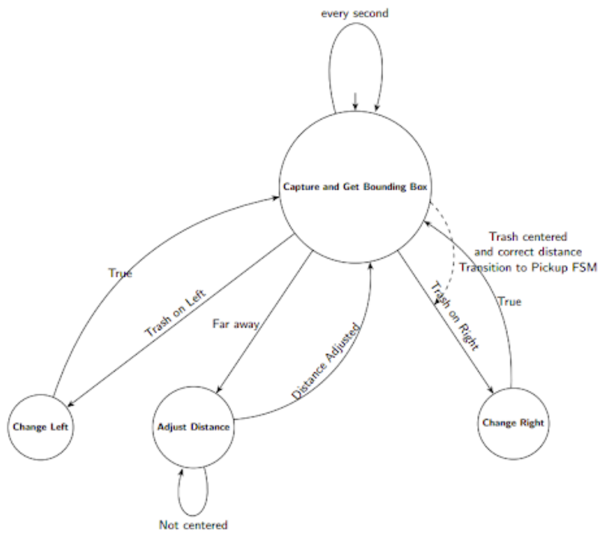

Figure 5: Finite State Machine of Repositioning

- Deploying the pretrained model on Raspberry Pi, a resource-constrained device.
- Customizing weights for optimal performance on limited computing power.

2. **Benefits and Limitations of Using TinyML on Resource-Constrained Devices**
   - Benefits:
     (a) Low power consumption.
     (b) Real-time data processing.
     (c) Cost-effective solution.
   - Limitations:
     (a) Potential reduction in accuracy.
     (b) Limited complexity of manageable tasks.
     (c) Challenges in model development and training.

3.2 Sensors and Actuators

1. **Sensors:** We used a light sensor and a rasberry pi camera for our main sensors in this project.
   (a) The light sensor was used for navigation purposes. We ended up not using the middle light sensor as the thickness of the tape caused sensors too close to each other to contradict each other, confusing the software that was driving our bot.
   (b) The rasberry pi camera sensor was used for trash identification and repositioning.
2. **Actuators:** The primary actuators we used were the servo motors which drove our robot arm and the tank wheel motors which drove the wheel itself. Both gave us different troubles.
   (a) There were two wheel motors, one on the back of each side of the tank. The front two wheels did not have motors. The trouble we faced was aligning the back wheels with the front. This required us to make frequent adjustments to the back wheels, and even led us to eventually sodering the back motor wires to the motor because the frequent adjustments caused them to unattach.
   (b) The servo motors which we used on the arm were frequently malfunctional, thankfully we ordered spares early on.

3.3 Wireless Control

1. **Explanation of the wireless control system for the robot:** The main idea is to use webhook to retrieve the ip address and use ssh to connect to it. This process requires wifi connection, where we use the eduroam to achieve that. Thus, all the coding and execution are performed through ssh.
2. **Communication protocols and technologies used:** We also use the local https to store the image, which is easy to get for other applications.

4 Accomplishments

The goal of this project was originally to construct a robot that executes line-following navigation, interrupts the process when it spots a piece of trash, repositions itself towards the trash, picks it up, and continues navigation thereafter. We were able to implement the navigation, a computer vision algorithm that identifies trash, a repositioning algorithm, and picking up the trash separately. However, we have yet to successfully integrate these components and implement necessary interrupts.

4.1 Line Following

4.1.1 Our initial approach.

Our initial approach to the line-following implementation involved the use of OpenCV. The library essentially isolates a specific color, in our software's case, red, and outlines it with a virtual box. The navigation idea would be to steer towards the center of the box at a rate that's proportional to the distance between the tank's midline and the box's midline. This is the 'P' component of the commonly implemented PID control theory. The practice of line detection can be shown in 6.

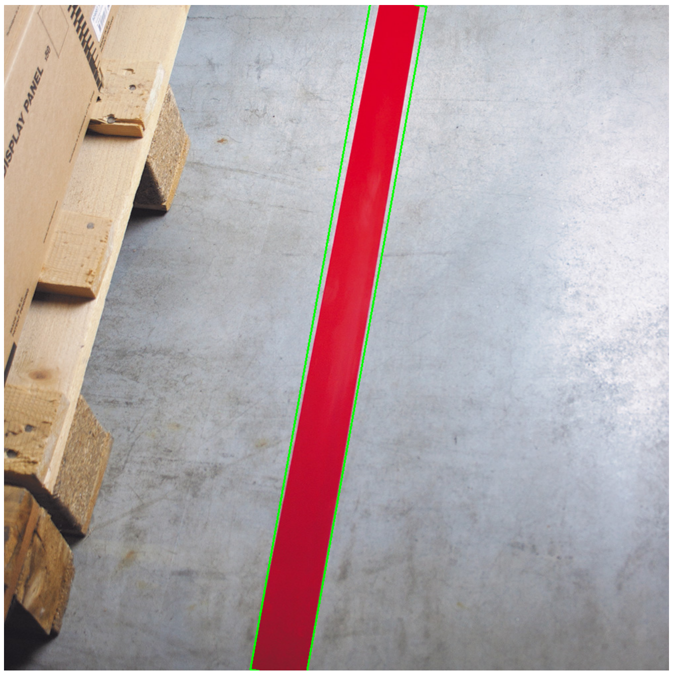

Figure 6: A box generated by our OpenCV detectline.py algorithm.

4.1.2 The approach that we ended up using.

Ultimately, we chose to execute line navigation by making use of the light sensors at the bottom of our Adeept Tank vehicle. Adeept provided pre-written software to do this, however, some combination of inconsistency in assembly and code bugs made it unusable. Therefore, we ultimately had to rewrite the software. This was done in two parts.

1. Writing move.py function. The key here was to implement the logic to trigger motor actions in order to obtain a desired movement. Below is shown the code segment for our left-side wheels.

```
        def motor_left(status, direction, speed):#Motor 2 positive and negative rotation
if status == 0: # stop
GPIO.output(Motor_B_Pin1, GPIO.LOW)
GPIO.output(Motor_B_Pin2, GPIO.LOW)
GPIO.output(Motor_B_EN, GPIO.LOW)
else:
if direction == Dir_backward:
GPIO.output(Motor_B_Pin1, GPIO.HIGH)
GPIO.output(Motor_B_Pin2, GPIO.LOW)
pwm_B.start(100)
pwm_B.ChangeDutyCycle(speed)
elif direction == Dir_forward:
GPIO.output(Motor_B_Pin1, GPIO.LOW)
GPIO.output(Motor_B_Pin2, GPIO.HIGH)
```

```
pwm_B.start(0)
pwm_B.ChangeDutyCycle(speed)
```

In particular, we produced a move() function, which input a speed, two directions (forward/backward and left/right), and a turn radius (0 as a stationary turn, 1 as no turn).

2. Next, we wrote a run.py algorithm, which used the light sensors to move. We played around with several implementations, but what ultimately worked was using activations of the left and right sensors (sensors were activated when the light they produced was absorbed by black tape instead of being reflected back) to turn right or left, respectively, and steadily moving forward when neither sensor was active.

```
        status_left == 1:
        move.move(100, 'forward', 'right', 0.6)
    elif status_right == 1:
        move.move(100, 'forward', 'left', 0.6)
    else:
        move.move(100, 'backward', 'no', 1)
```

Ideally, we would have used the initial computer vision approach as a backup for the light sensing. Although the light sensing method is more logical when the tank is right above navigation tape - more accurate sensing, not confusing other slight black streaks on the ground, less computationally expensive - using computer vision would have allowed us to look ahead for navigation purposes, rather than just immediately below. By having this capability, our bot would be able to re-navigate towards the tape track in the case that it went off track (eg. in order to pick up trash).

4.2 Trash Identification via Computer Vision

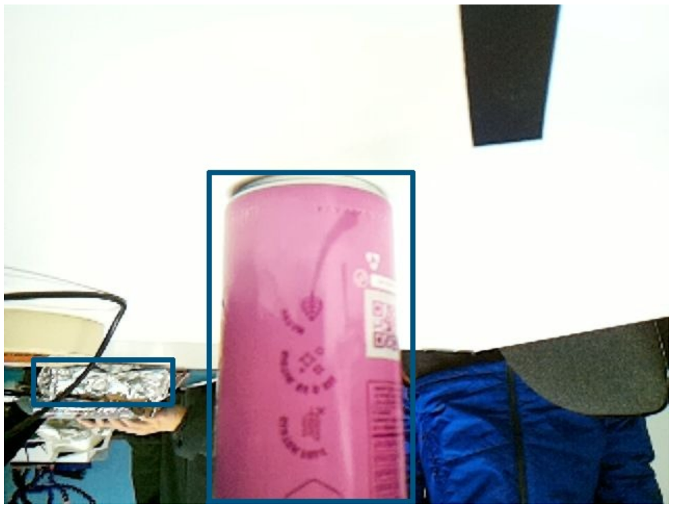

Figure 7: Image Recognition Example

The cornerstone of the robotic trash identification system is a robust computer vision model, specifically engineered to recognize and localize tin cans within its operational environment. Leveraging the power of the Ultralytics YOLO-v8 architecture, the team has meticulously pre-trained the model on a dataset of pre-labeled tin can images sourced from OpenImages—a comprehensive collection of annotated photographs. The practice of detecting tins can be shown in 7.

Training the Model: For the training process, Google Colab's formidable A100 GPUs were employed, providing the necessary computational resources to train the model over 200 epochs. This extensive training regimen was designed to fine-tune the model's parameters, ensuring high accuracy and swift recognition of tin cans within the robot's field of vision.

Code Integration: The integration of the model within the robot's control system involved setting up the necessary dependencies and adjusting the datasets for compatibility with the YOLO-v8 framework. Here is an excerpt from the codebase illustrating the preparation of the dataset for training:

```
!cd OIDv4_ToolKit && pip install -r requirements.txt
!cd yolov5 && pip install -r requirements.txt

# Downloading and preparing the tin can dataset
!yes| python3 OIDv4_ToolKit/main.py downloader --classes "Tin_can" --type_csv train
# ... Additional dataset preparation steps are omitted for brevity.
```

Dataset Structure and Training: The dataset's structure is meticulously organized into corresponding image and label directories for training, testing, and validation purposes. Following the data preparation, the training script is executed, which initiates the model's learning phase:

```
!cd yolov5 && python train.py --img 640 --batch 16 --epochs 200 \
--data /content/dataset.yaml --weights yolov5s.pt --cache
```

Training Output: Upon completion of the training, the model's performance metrics—such as precision, recall, and mean Average Precision (mAP)—are evaluated to ensure the model's efficacy. The training process generates output that includes these metrics for each epoch, providing insights into the model's improvement over time.

Challenges and Achievements: The task of identifying tin cans in various contexts and lighting conditions posed significant challenges. Nonetheless, the trained model demonstrated proficiency in detecting tin cans with commendable accuracy, a testament to the team's diligent training and optimization efforts. Such achievements in computer vision have profound implications for the advancement of autonomous robotic systems in waste management and beyond.

4.3 Repositioning Mechanism Detailing

Utilizing the Yolo-v5 object detection framework, our team has developed a sophisticated repositioning algorithm for the trash-collecting robot. The core functionality of this algorithm is to enable the robot to autonomously navigate its surroundings while actively scanning for waste objects, specifically focusing on tin cans as a target for this iteration.

During its patrol, the robot performs a continuous scan every second, processing the visual field through its onboard camera system. The Yolo-v5 algorithm analyzes the captured frames in real-time, searching for the predefined object classes—tin cans, in this case—within the environment. Once an object of interest is detected, the algorithm calculates the spatial location of the object within the camera's field of view, represented as bounding box coordinates. These coordinates are expressed as percentages relative to the screen's dimensions, indicating the object's position in terms of left, center, and right regions of the image.

The robot's logic controller interprets these bounding box coordinates to make dynamic repositioning decisions. For instance, if a tin can appears between 30 to 55 percent from the left of the screen, this indicates that the object is located to the left side of the robot's current path. The controller then initiates a corrective maneuver, adjusting the wheel actuators to steer the robot towards the right, thereby centering the object in its navigational path. Conversely, if the object appears on the right side of the visual field, the robot will make an equivalent adjustment to the left.

A critical aspect of the repositioning algorithm is the handling of objects detected in the central region of the camera's view. When the bounding box coordinates average within the 45 to 55 percent range horizontally, it signifies that the tin can is directly ahead of the robot. In this scenario, the robot proceeds to move forward, aligning itself for optimal interaction with the object—either for further inspection or for the actual collection process.

The repositioning strategy not only enhances the robot's navigational precision but also optimizes the efficiency of object collection by reducing unnecessary movements and ensuring a direct path to the target objects. This implementation underscores the synergy between advanced computer vision techniques and robotic mobility, pushing the boundaries of autonomous environmental service robots.

4.4 Robot Arm Operation/Trash Pickup

1. **Overview of the robot arm's functionality:** The robotic arm is a sophisticated mechanical appendage designed with four servo motors, each responsible for a specific movement. The first servo controls the base of the arm, allowing for rotational movement. The second and third servos actuate the upper and middle sections of the arm, enabling it to extend or retract. The fourth servo operates the claw, providing the gripping action necessary for trash pickup. Additionally, a servo dedicated to rotating the claw facilitates precise orientation of the gripper. By programming specific degree ranges (from 0 to 400, varying for each servo), the team can finely control the arm's movements to perform complex tasks such as grasping and lifting objects.

2. **Challenges:** During the project, the team faced several setbacks. A notable mechanical issue occurred when one of the motors malfunctioned, prompting an urgent replacement. Hanson took charge of procuring a new motor, but this led to several days of delay as the team awaited its arrival. In another instance, attempts to control servo 13 via SSH resulted in connectivity losses, rendering the controlling computer inoperative—a significant hardware limitation that went beyond the team's immediate capacity to resolve. Additionally, the robot's design encountered limitations when attempting to lift heavy or bulky objects like tin cans, which exceeded the arm's carrying capacity.

3. **Achievements in the trash pickup process:** Despite the challenges, the team successfully implemented a basic trash pickup mechanism. Utilizing servo 12 for the upper arm and servo 15 for the claw, the robot demonstrated its capability to pick up lighter and more manageable objects such as paper napkins and compressed tin cans. This achievement signifies a critical step forward in the project, showcasing the potential for further development and optimization of the trash collection process.

4. **Code Snippet:** A sample code excerpt showcases the command structure for operating the robot's arm using the Adafruit PCA9685 library:

```
# Open the claw to prepare for trash pickup.
pwm.set_pwm(15, 0, 100)
# Sequentially move the arm down to reach the trash.
pwm.set_pwm(12, 0, 200)
# ... Additional steps have been omitted for brevity.
```

   This code snippet illustrates the direct control over the arm's servos, where specific pulse width modulation (PWM) signals are sent to execute movements like opening the claw or lowering the arm to approach and secure the trash.

5 Conclusion

5.1 Summary of Achievements

1. The team successfully assembled and configured the necessary hardware, and integrated a machine learning algorithm (Yolo v5) for object recognition with the robotic navigation system.

2. The project contributed to the field of ML-based robotics by demonstrating a practical application in environmental sustainability and providing an educational model that merges theoretical learning with practical application.

5.2 Lessons Learned

1. The project underscored the importance of seamless integration between hardware and software and highlighted the real-world challenges of machine learning in robotics.

2. The experiences from the project emphasize the need for extensive testing and an iterative development approach, and they offer insights into the scalability and potential real-world deployment of robotic systems.

6 Future Work

6.1 Potential Enhancements

1. Improvement in the robustness and accuracy of trash detection, enhancement in power management for increased autonomy, and refinement in the mechanical design for better manipulation and collection of various types of waste.

2. Future iterations could explore advanced navigation algorithms, incorporate more sophisticated machine learning models for object discrimination, and potentially scale up the design for larger-scale waste management applications.

7 Acknowledgments

7.1 Recognition of Contributions

1. The contributions of team members Samyak Tiwari, Jiamu Chen, and Hanson Lee, as well as the guidance of course instructors and the assistance from external contributors, are gratefully acknowledged.

2. The team extends their gratitude for the support and resources provided by the educational institution, any sponsoring entities, and the broader open-source community that contributed to the project's success.

8 Video Demos

8.1 Line Navigation

`https://drive.google.com/file/d/1AFCU05UVLMZm8HqpRZhMUvClkzjyBkll/view`

8.2 Trash Pickup Demo

`https://drive.google.com/file/d/1tIZOx-o6P_PddoKbytw9d3rS4AFtwXJw/view`

8.3 Repositioning Demo

`https://drive.google.com/file/d/1V5gEMWmX1bPAbiodPFkdTfXMa2dar5fG/view`

9 Repositories and References

9.1 GitHub Repositories

- Project Repository: `https://github.com/jimchen2/EECS-149-Final-Project`

9.2 References

For the development of this project, the following resources and references were utilized:

1. Ultralytics GitHub Repository: `https://github.com/ultralytics/ultralytics`

2. Ultralytics Raspberry Pi Guide: `https://docs.ultralytics.com/guides/raspberry-pi/`

3. Adeept RaspTank GitHub Repository: `https://github.com/adeept/Adeept_RaspTank`

4. OpenAI ChatGPT: `https://chat.openai.com/`