Rodent Deterrent and

Classification System

Final Report

May 2, 2024

Team 55

Jung Ki Lee

Mankeerat Sidhu

Rishab Vivekanandh

TA: Angquan Yu

Abstract

During the warmer seasons, a prevalent challenge arises across various outdoor environments, including residential backyards, golf courses, and open grass fields: the incessant disruption caused by rodents and avian species in their pursuit of subterranean prey such as earthworms and soil-dwelling insects. This phenomenon not only poses a significant nuisance to grass farming enterprises but also undermines the aesthetic appeal and horticultural efforts of individual homeowners. Regrettably, existing deterrent methodologies have proven inadequate in providing enduring solutions to this pervasive issue.

In response, our research endeavors sought to engineer an innovative solution by harnessing the synergy of established technologies with cutting-edge advancements. Our devised two-part system constitutes a pioneering approach aimed at effectively mitigating rodent intrusions. The initial component of our system revolves around detection, predominantly facilitated by a network of specialized sensors. Incorporating passive infrared (PIR) and ultrasonic sensors, this phase is engineered to discern motion patterns indicative of rodent activity, subsequently activating the ultrasonic component to spatially map the environment. Following successful detection, the secondary phase of our system is invoked, focusing on the active deterrence of rodent presence. Leveraging advanced object recognition models, exemplified by YOLOv8, in tandem with a camera apparatus, our solution adeptly identifies and tracks the targeted species. Upon accurate detection, the system orchestrates a dual-pronged deterrence mechanism, featuring luminous LED illumination in a strobe-like fashion, complemented by the emission of high-frequency sound waves via ultrasonic transducers.

The amalgamation of these multifaceted components culminates in a comprehensive deterrent system, poised to effectively address the persistent challenge of rodent and avian intrusion within diverse outdoor environments. This research not only signifies a pivotal advancement in pest management technology but also underscores the potential for interdisciplinary collaboration to engender innovative solutions to real-world challenges.

Table of Contents

1. Introduction

1.1 Problem

Every summer and fall, thousands of backyards, lawns, golf courses and open grass fields suffer from rodents and birds digging the ground searching for earthworms, soil-dwelling insects, and insect larvae. This leaves behind large patches of loose turf and ruins the grass. Not only is this a huge problem for the grass farming industry but is also a nuisance for every backyard owner, ruining the aesthetics and plants grown on the lawn. The current deterrent methods are technologically naive including just a motion sensor, lights and loud sounds which cause loud noises at night, fail to prevent lawn digging, and leave the user unaware of the type of rodent affecting their lawn.

1.2 Solution

We propose a rodent detection and deterrent system which comprises many parts. Using infrared and ultrasonic sensors on a rotating servo, we would detect any rodent outside of the usual landscape of the lawn the device is placed in. The PI camera system would simultaneously work to take a clean shot of the rodent/bird and store it in the file system. If recognized to be a ground digging rodent, for the actual deterrent, our colored lights and localized speaker beeps go in the direction of the rodent rather than in a single direction like previously commercialized methods. This ensures rodent deterrence and also informs the user the type of animals responsible for digging their lawn.

1.3 High Level Requirements

1. The system must be able to successfully and accurately detect rodents with > 90% success rate and also avoid false detections based on other movements in the environment (eg. person walking, dog running).
2. Components should have high durability and battery capacity to ensure a long lasting solution (battery life of up to a month).
3. Sensors should be capable of detecting at a relatively long range while also being able to scan a large field of view (360° field of view and 10m radius).

2. Functionality

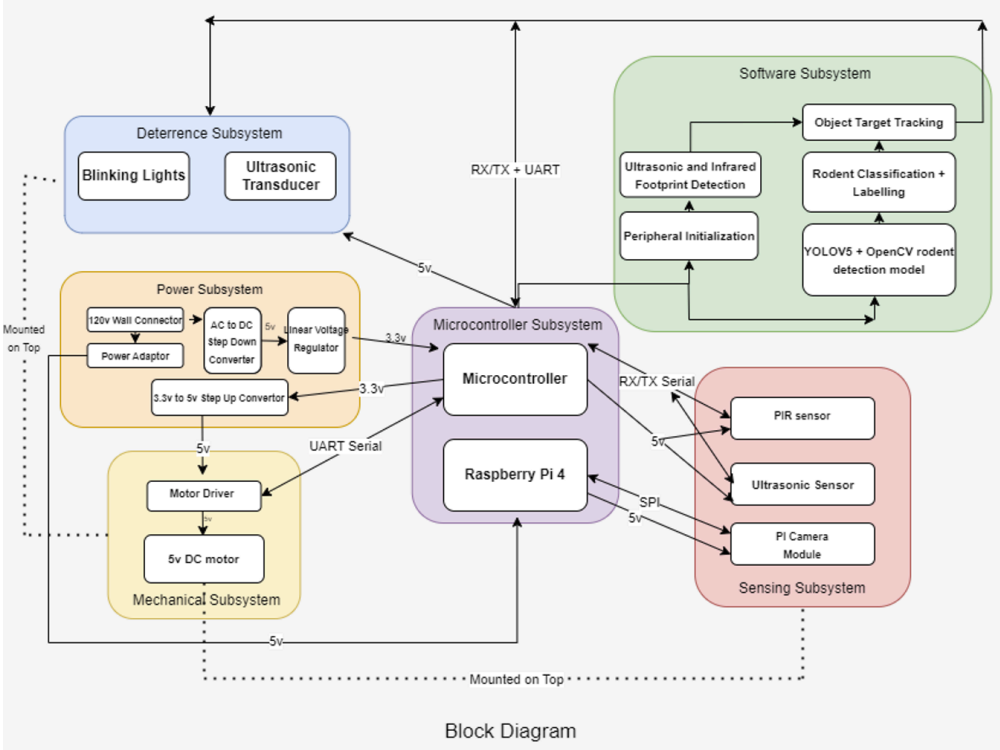

Figure 1: Block Diagram

In our block diagram (Figure 1), we see how multiple subsystems come together to create our intended functionality, but it is equally important to understand the functionality of this system. We created a 2-part system that, when combined, would effectively deter the rodents. The first step in the process of the system is the detection. For our project, this consisted of most of our sensors. The sensors we used for this project were PIR and ultrasonic sensors. The detection system first aims to detect motion of the object through the PIR sensors and then activate the ultrasonic. We used this methodology to ensure that we only utilize power when necessary. This is ensured since the PIR sensors stand still and cover a 360° field of view, and no other sensor will turn on until this occurs. The ultrasonic sensor's goal on the other hand is to map the environment and to identify where the target is in the environment. We showed this by also creating a helpful visual for the radar (Figure 2). After this phase, the second part of the system is activated and is involved with deterring the rodent. The system uses a camera and novel object detection models (YOLOv8) to identify the rodent, track it, and activate the deterrent mechanisms. Our system uses two main deterrence systems: bright LEDs and ultrasonic transducers. Once the camera correctly detects the animal, the LEDs are activated in a strobe-like manner and the buzzer is activated at ultrasonic frequencies. After activating these sequences of events, the rodent is highly likely to be deterred from the area.

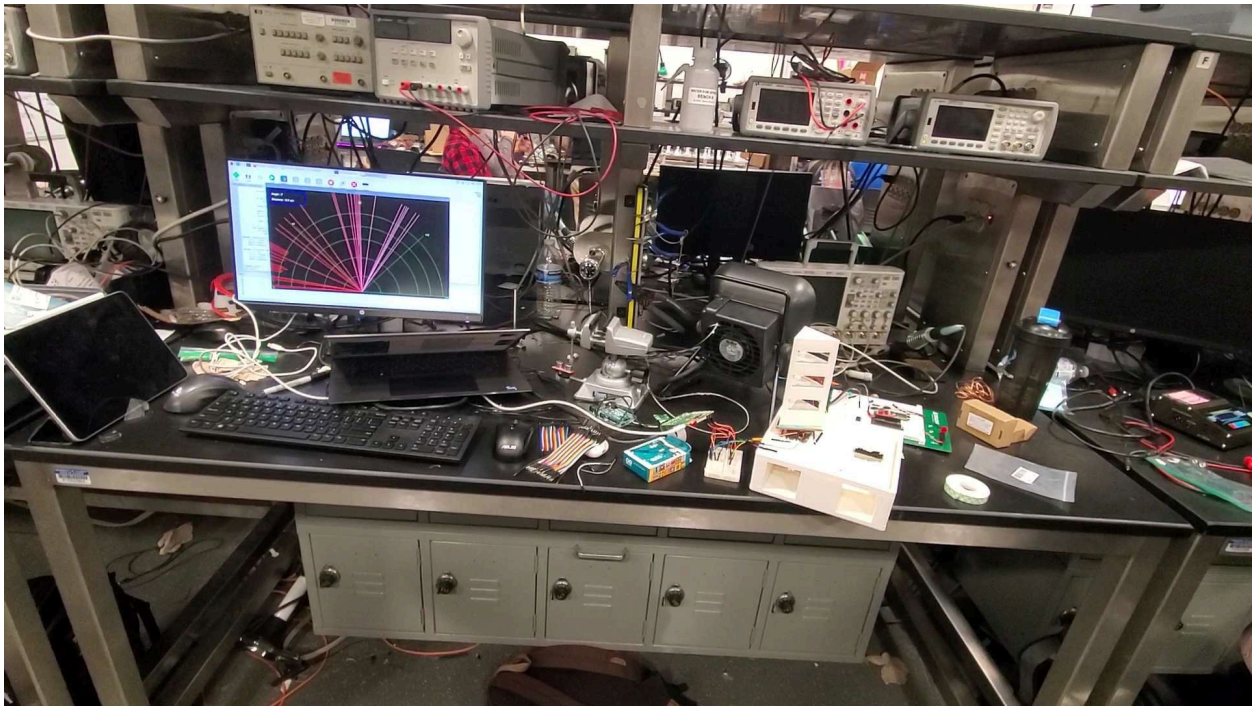

Figure 2: Radar Visual

3. Subsystem Overview

3.1 Subsystem Design

This project consists of several integral subsystems that our team needs to build to ensure the success of this project as displayed in the block diagram (Figure 1). These subsystems will need to work together to reach our goal of deterring rodents. Our system is comprised of six core subsystems:

1. Deterrent Subsystem
2. Microcontroller Subsystem
3. Software Subsystem
4. Power Subsystem
5. Mechanical Subsystem
6. Sensor Subsystem

For each of the subsystems, we will briefly discuss their importance within the whole scope of the project, while also discussing essential requirements we need from components that are core to the subsystems. Lastly, we discuss how each of the subsystem requirements are verified in a procedural manner.

3.1.1 Deterrent Subsystem

The deterrent subsystem receives signals from the sensor and microcontroller subsystem. When the sensor subsystem detects a rodent, the sensor subsystem and the microcontroller subsystem sends appropriate signals, activating the deterrent system. The deterrent subsystem is core to our entire system, and the success of this subsystem relies heavily on the success and reliability of the devices we are using. Therefore, this subsystem requires heavy attention and thoughtfulness in its design.

a. **Lights**
   i. When the deterrent system detects a rodent, the system should produce a lighting mechanism to startle the rodent.

#	Requirements	Verification	Verification Status
1	The lights should be blue and flashing in 0.2s intervals	● Set up the lights according to the specified configuration (blue color, flashing at 0.2s intervals). ● Use a stopwatch or timing device to measure the duration of each flash. ● Confirm that the lights flash at intervals of 0.2 seconds consistently by observing multiple cycles of flashing. ● Record the timing of each flash and calculate the average interval to ensure compliance with the requirement. ● Repeat the verification process multiple times to ensure consistency and reliability of the flashing pattern.	Yes
2	The lights should have a capacity to reach 500 nits	● Use a photometer or light meter capable of measuring luminance ● Initially position the sensor of the light meter within the industry standard of 10-50 cm from the lights. ● Activate the lights and measure the luminance emitted by the lights. ● Record the luminance value and compare ● Repeat the measurement process from steps 3 at varied distances to account for variations in light intensity.	Yes
3	The lights require 20mA current and should be gated to create flashing	● Measure the current flowing through the lights using a multimeter or current probe. ● Connect the lights to a power source capable of supplying the required current. ● Verify that the current drawn by the lights does not exceed 20mA during operation. ● Use an oscilloscope or logic analyzer to observe the gating signal applied to the lights for flashing. ● Confirm that the gating signal effectively controls the on-off cycling of the lights to create the	Yes

		flashing effect. ● Ensure that the gating signal maintains the specified flashing pattern (0.2s intervals) consistently. ● Validate the performance of the gating mechanism under different operating conditions and loads.	

b. **Speaker**

i. When the deterrent system detects a rodent, the speaker should produce sounds that repels the rodent. The sounds should be outside the human hearing scale such that the deterrent system won't be an annoyance to the users. Furthermore, the speakers should vary their frequency range such that the rodents don't become used to the sounds.

#	Requirements	Verification	Verification Status
1	The frequency at which the speaker sound should be at should above between 20kHz and 60kHz	● Use a frequency analyzer or spectrum analyzer capable of measuring ultrasonic frequencies. ● Power the speaker and play the sound ● Use the frequency analyzer to measure the output frequency of the speaker ● Verify that the measured frequencies fall within the specified range (20kHz to 60kHz). ● Repeat the frequency measurement to ensure consistency.	Yes
2	The speaker should be able to reach ranges up to 5m	● Set up the speaker and a sound level meter at a known distance (e.g., 5 meters) apart in an open space. ● Generate a sound signal with a constant intensity level through the speaker. ● Measure the sound pressure level (SPL) at the specified distance using the sound level meter. ● Verify that the measured SPL meets the specified threshold	Yes

		for audibility at the given distance.	
3	The speaker should have a minimum strength of 60 dB	• Use a sound level meter to measure the output strength (sound pressure level) of the speaker. • Generate a sound signal with a constant intensity level through the speaker. • Measure the SPL at a specified distance from the speaker. • Verify that the measured SPL exceeds the specified minimum threshold of 60 dB.	Yes
4	The speaker should have a current source that doesn't exceed 50 mA given a 5V power source	• Connect the speaker to a 5V power source. • Measure the current passing through the speaker using a multimeter. • Ensure that the measured current does not exceed 50 mA. • Document the results and compare them against the specified requirement.	No
5	The speaker should be randomly vary the timing of the bursts of ultrasonic sound waves	• Use an oscilloscope or logic analyzer to monitor the timing of the bursts of ultrasonic sound waves generated by the speaker. • Play a series of sound signals through the speaker and observe the timing of the bursts. • Verify that the timing of the bursts exhibits random variation, with no discernible pattern or repetition. • Analyze the waveform to confirm that the bursts occur at irregular intervals, as required by the specification.	No

3.1.2 Microcontroller + Communication Subsystem

The microcontroller will process all the information with lowest possible latency and integrate the sensors with the deterrence system. The success of the microcontroller is integral, as it will serve as the middle-man for passing critical information from the software subsystem, to the deterrence and power subsystems. Furthermore, the microcontroller is also essential for translating the data from our sensor subsystem, to the software subsystem, such that the software subsystem can perform at the highest level. Although the microcontrollers themselves that we are using for our design are off-the-shelf, we do include requirements that we need from our microcontrollers for this subsystem to successfully work.

#	Requirements	Verification	Verification Status
1	Microcontroller should have communication interfaces capable of at least 1 Mbps for UART, 10 Mbps for SPI, and 400kHz for I2C	● Connect the microcontroller to a compatible device for UART, SPI, and I2C communication. ● Transmit and receive data through each communication interface while measuring the transfer speed. ● Use a suitable measuring device or protocol analyzer to monitor the data transfer rate. ● Verify that the UART communication achieves a minimum speed of 1 Mbps, SPI achieves 10 Mbps, and I2C achieves 400 kHz.	No
2	Microcontroller should support USB 3.0 for high communication protocols for data transfers of at least 100 Mbps	● Connect the microcontroller to a USB 3.0 compatible device or host. ● Transfer data between the microcontroller and the USB host while measuring the data transfer rate. ● Use appropriate tools or software to monitor the USB data transfer speed. ● Verify that the microcontroller supports USB 3.0 and achieves a minimum data transfer rate of 100 Mbps. ● Ensure that the USB communication remains stable and reliable during the verification process.	No
3	Microcontroller should have processing speed of at least 100	● Execute real-time data processing tasks and	No

	MHz to handle real-time data processing and communication efficiency	communication protocols on the microcontroller. ● Measure the execution time of critical operations or algorithms using a stopwatch or timing device. ● Verify that the microcontroller can complete essential tasks within the specified time constraints. ● Monitor the microcontroller's clock frequency during operation to ensure it operates at or above 100 MHz.	
4	Microcontroller should have minimum RAM of 32 KB and minimum flash memory capacity of 256 KB to store program and code data	● Access the microcontroller's datasheet or specifications to confirm the RAM and flash memory capacity. ● Utilize debugging tools or software to retrieve information about the microcontroller's memory resources. ● Verify that the microcontroller's RAM capacity is at least 32 KB and flash memory capacity is at least 256 KB. ● Allocate and store program code and data on the microcontroller to ensure it fits within the available memory	No

3.1.3 Software Subsystem

The software subsystem's success is essential to our whole project succeeding. The software subsystem will work side by side with data provided from any sensor module to ensure that the tracking of the rodents is accurate and effective. The software will perform analysis on the live feed of the camera sensors as a means of detecting and tracking the rodent. The vision detection should occur with low latency, such that the analysis can essentially be done in real time. The goal in the end of this subsystem is to identify the rodent within the view accurately. Figure 3 shows a high level flow diagram of how the software subsystem will work. The model we will create will be trained on a robust dataset and be processed using open source databases and libraries like OpenCV and PyTorch. The model will be extensively tested using the dataset to tune and analyze its performance. In the end, the results of the software subsystem will provide signals to the other modules.

#	Requirements	Verification	Verification Status
1	Software should detect rodent with +90% accuracy	• Set up a testing environment with representative input data for rodent detection • Pass in random images from the rodent image dataset to the model • Measure the accuracy and F1 score for the run • Repeat the testing with different input images to ensure robustness and reliability • Collect data and see if the accuracy exceeds the specified amount	Yes
2	The detection time should be done in real time and should have latencies lower than 0.1s	• Set up a testing environment with representative input data for rodent detection. • Implement the rodent detection algorithm or model on the software subsystem. • Simulate the detection process with various inputs representing different scenarios and conditions. • Measure the time taken for the software to process each input and detect the presence of a rodent. • Verify that the detection latency is consistently lower than 0.1s for all test cases.	Yes
3	Require a database of at least 1000 images of rodent to train vision model on	• Collect or curate a dataset comprising at least 1000 images of rodents for training the vision model. • Ensure that the dataset includes diverse images representing various rodent species, poses, environments, and lighting conditions. • Verify the integrity and quality of the images in the dataset to ensure suitability for training. • Use data validation	Yes

		techniques to confirm that the dataset meets the specified quantity requirement. ● Perform statistical analysis to ensure the dataset's diversity and representativeness for effective model training. ● Document the sources of the images and any preprocessing steps applied to the dataset for transparency and reproducibility.	
4	Data should be constantly publishing information for other subsystems to use	● Implement data publishing functionality within the software subsystem to continuously transmit relevant information. ● Set up a data monitoring system to track the publication of information in real-time. ● Subscribe other subsystems or modules to receive the published data streams. ● Monitor the data flow between subsystems and verify that information is consistently published at regular intervals. ● Perform stress testing to evaluate the software's ability to maintain continuous data publishing under varying loads and conditions. ● Ensure that the published data is accurate, relevant, and up-to-date for consumption by other subsystems.	Yes

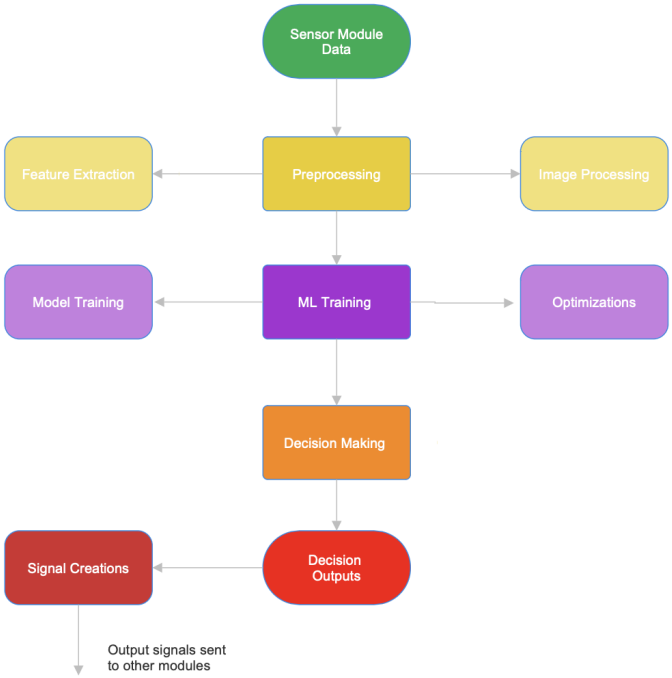

Figure 3: Software Detection High Level Overview

To meet the requirements of the software subsystem, we created a high level overview of how the object detection mechanism would work. Figure 3 describes the process of achieving this. Our object detection code was based on a model known as YOLO (You Only Look Once), specifically the YOLOv8n model, which is an extremely pruned down model from the normal mode. In the process of creating the object detection code for rodents, we realized that it was extremely difficult to meet the requirements. Achieving an accuracy of 90% is very difficult, because accuracy is highly dependent on the amount of training data.

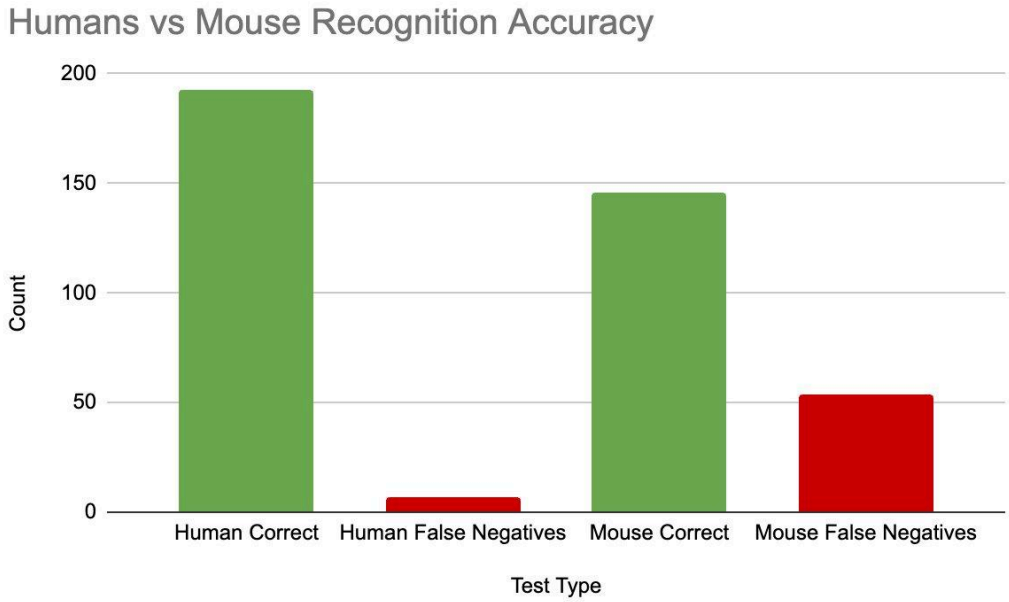

Figure 4: Human vs Mouse Recognition Accuracy

In Figure 4, we see that, when comparing the object detection accuracy with human and mouse, we see that accuracy takes a relatively hard hit. In our testing we saw an accuracy of around 80% for detection of rodents compared to around 95% for humans. The main reason for this is that there is extensive data and annotated images for humans, and significantly less for rodents such as mice and racoons.

The average frame rate that we saw, when running the model on the RPi4 was around 14 FPS. When running the model on a higher performance computer, it reached 60 FPS. The frame rate significantly took a hit on the latency of the deterrence system being activated. Since the frames were low on the RPi4 while the object detection algorithm was acting, we did see instant detection with a latency of lower than 0.1 ms. However, because the frames were extremely low on the RPi, the accuracy of the detection took hits as some frames appeared to be blurry due to the low frame rate.

Nevertheless, the remainder of the requirements within the software subsystem were met. In total, we collected around 1,200 images to train the model and the object detection code was consistent sending information throughout the system to activate and deactivate certain systems. On a side note, we should have used more training images to achieve higher accuracy, but this was not feasible due to the time constraints of this project.

3.1.4 Power Subsystem

The power subsystem is responsible for generating, storing, regulating, distributing, and managing electrical power to ensure the proper functioning of onboard systems and instruments. This system includes using a 120v wall connector, multiple power adaptors and voltage regulators.

#	Requirements	Verification	Verification Status
1	The power is set to ensure overcurrent protection, overvoltage protection, and thermal management to prevent damage to electrical components and ensure safe operation.	● Review the design and specifications of the power subsystem to ensure it includes overcurrent protection mechanisms such as fuses or circuit breakers. ● Verify the presence of overvoltage protection components such as voltage regulators or surge protectors in the power supply circuitry. ● Test the thermal management system under various operating conditions to ensure it effectively dissipates heat and prevents components from overheating. ● Conduct stress tests and fault simulations to confirm that the protection mechanisms trigger appropriately in case of	Yes

		overcurrent, overvoltage, or thermal issues.	
2	Using power from a wall connector ensures continuous power to all the subsystems and rotation of the sensors mounted on top of servos.	• Test the protection circuits for handling overcurrent and overvoltage conditions. • Monitor the temperature of the regulators during operation to ensure they do not exceed thermal limits. • Perform tests to confirm that the step-up and step-down regulators maintain a stable output voltage under varying load conditions.	Yes

Although we use a wall connector for our project, we run experiments with a 10,000 mAh battery pack to test the ability to be used and placed in the lawn for extended amounts of time. The results are shown in the graph below as the system was tested extensively without staying in idle (PIR motion) setting for too long. This test solidified the battery life of around 12 hours with a small phone charging battery pack (Figure 4). Thus further attempts can be made to install solar powered batteries to the system in future work.

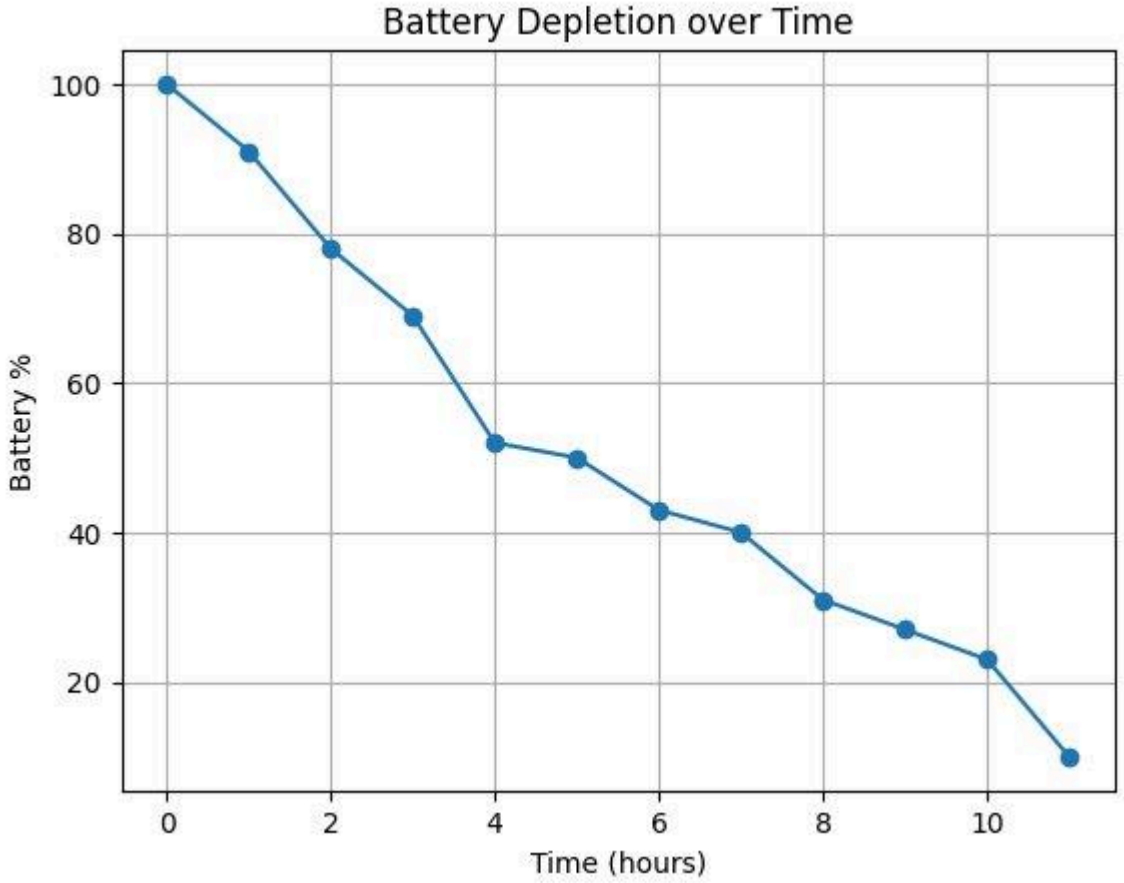

Figure 4: Battery Depletion

3.1.5 Mechanical Subsystem

The mechanical subsystem involves any of our parts that requires any movement. In the scope of this project, this includes all the servos that will move our device so that it is able to scan the entire area. Ensuring these mechanical components work are important, as we want to ensure we can scan the entire field of view.

#	Requirements	Verification	Verification Status
1	The rotor should have a minimal RPM of 10 RPM to allow for continual monitoring of the environment	• Install the rotor in the intended environment and measure its rotational speed using appropriate sensors or instrumentation. • Validate that the rotor consistently maintains a minimum RPM of 10 RPM under various operating conditions.	Yes
2	Servos should have a MTBF (mean time between failures) of 10,000 hours to ensure the servos are capable of constant activation and maintaining performance	• Gather reliability data for the servos or reference similar models with known MTBF values. • Conduct accelerated life tests on a sample of servos to estimate their failure rate over time. • Analyze the test results to calculate the MTBF and compare it against the specified requirement of 10,000 hours.	No
3	The platform, which consists of the necessary systems (sensors, camera etc.), should be able to withstand at least 10 N of force such as to prevent dislodgement.	• Apply a controlled force of at least 10 N to the platform in different directions to simulate potential dislodgement scenarios. • Verify that the platform remains securely attached and functional after exposure to the specified force levels.	Yes

3.1.6 Sensor Subsystem

The sensor subsystem is essential to our project. It will feature an array of sensors that we will congregate to ensure that we can accurately track the rodent. Furthermore, the need for multiple sensors acts as a

failsafe, to ensure that we can still perform the task should any of the other sensors be inhibited in doing their job. The data from these sensors will interact directly with the microcontrollers of our system, which will pass the data onto our software subsystem. The parts within the sensor subsystem feature off-the-shelf items, so we will not mention any requirements and verifications.

4. Design

4.1 Initial Design

Figure 1 shows our initial block diagram for the design of completing this system. While we followed this diagram rather religiously, we realized that, along the way, there were changes that needed to be made to make these subsystems work together more efficiently.

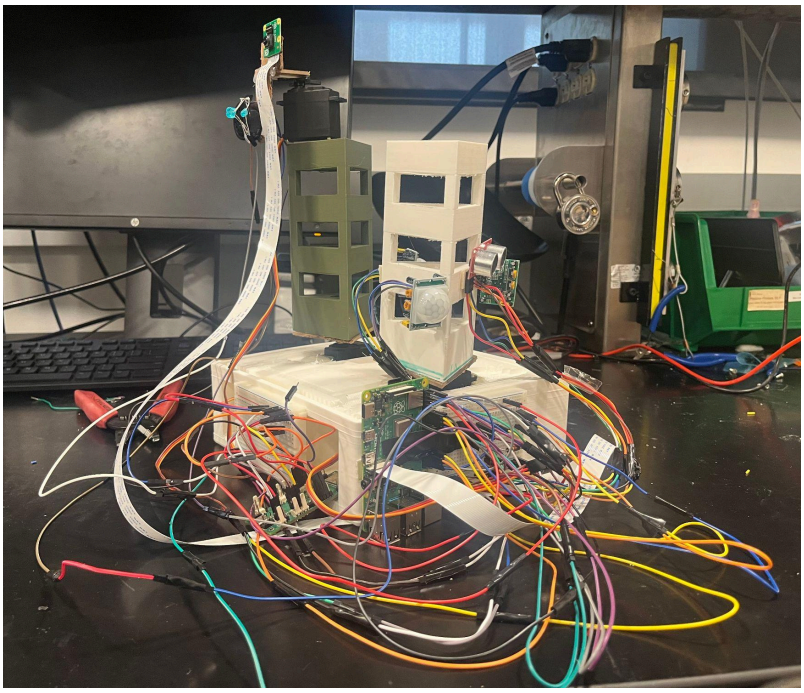

Figure 5: Final Product

Before explaining design changes that were made, it is essential to describe the final design. Figure 5 shows our final design. The two towers and the base of our design are custom 3D printed designs that we made. Again, this kept in touch with our initial idea to create a 2 part solution where one tower would be dedicated to sensing, and the other to deterring. Figure 6 shows a bare bones model of our 3D printed structures. The base structure of our system was designed to house the PCB, servos, the Raspberry Pi 4 and any additional breadboards that we would need for wiring. The white tower, which sits atop one of the servos, is the sensing system within our mechanism. The PIR sensors, and ultrasonic sensors reside on this tower. This tower remains still until the PIR sensor detects movement, after which the tower rotates and maps the environment using the ultrasonic sensor aboard. The green tower in Figure 5 is our main deterrent tower. On this tower sits our LEDs, our ultrasonic transducer and our camera. When the

deterrent system is activated, LEDs start flashing at 10 Hz. The ultrasonic transducer also produces ultrasonic sounds of frequency 20kHz as such sounds are known to be effective at deterring rodents [1]. This tower also contains an extra servo, to which the camera is attached to, as a means of covering the 360° field of view, which is not possible given one servo.

Figure 6: 3D Printed Structure

4.2 Design Changes

In the process to create this final design, we had several design changes that we had to make due to unexpected obstacles created from our initial design.

For one, we noticed that our initial design clashed with our high level requirement to conserve power. By keeping the camera on the sensing tower, we actively need to power it to ensure it stays on. Furthermore, the camera needs to be able to cover the 360° view, which would mean that additional power would be spent on rotating the servos. This initial design was inefficient and failed to meet the goals that we had set with our high level requirement. As such, we converted our design, moving the camera to the deterrence tower. With this design change, it allowed us to keep the sensing tower still for the most part, allowing it to only be activated once the PIR sensor detects any motion. Furthermore, it would preserve power, as only when the PIR sensor detects motion, will the camera receive power from the Raspberry Pi. As such we were able to see longer battery life when connecting our system to a battery pack, compared to our initial design.

Another design change we had was with the use of the ESP32. This microcontroller was chosen to be the main hub for our code (Figure 7). However, during the process of coding the ESP32 we realized a few issues with its role as the main microcontroller. Firstly, we realized that the microcontroller had not enough near compute power to load and run our object detection model. In addition to this, we also realized that, unless communicated to serially, the data transfers were too slow between devices. Although we were unable to change our PCB design to address these changes, we decided to host most of the software materials on the Raspberry Pi, and utilized our PCB as a power source for the systems.

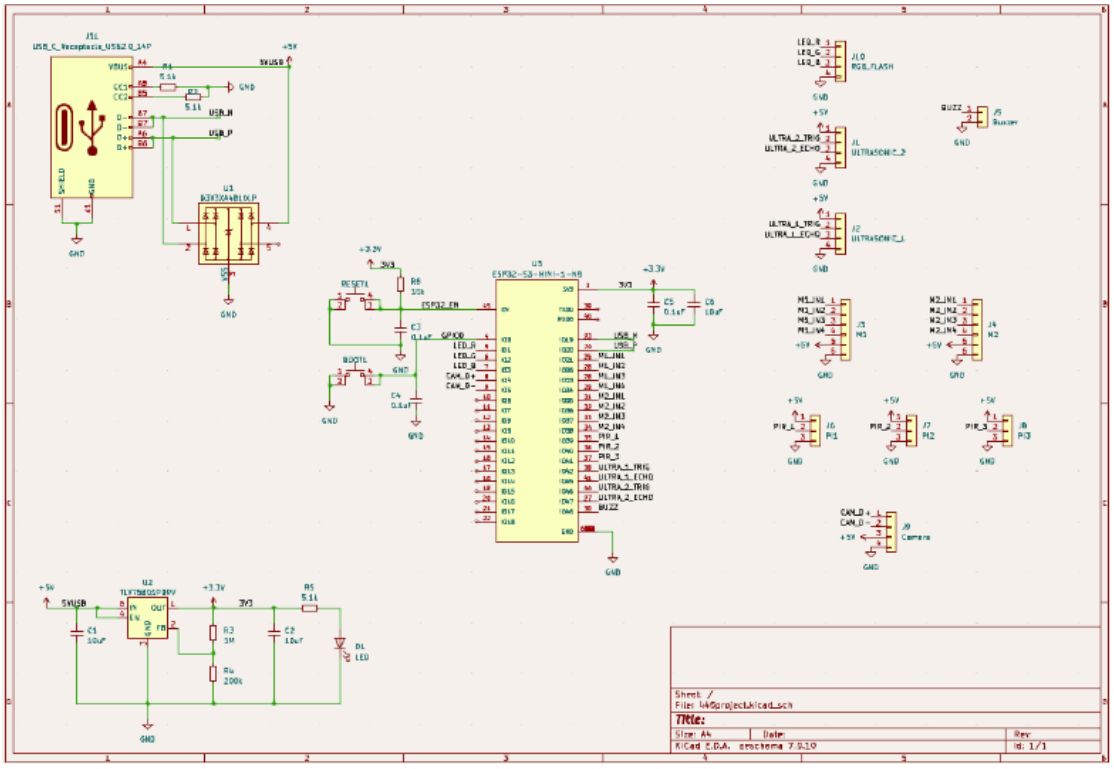

Figure 7: Initial PCB Design

5. Cost Analysis

5.1 Labor Costs

The average ECE graduate student earns around $45-$50 an hour. Our team is committed to working a minimum of 10 hours a week henceforth. Committing this time will be necessary to complete the several requirements along the way. Given the calendar, we will have around 8 weeks to complete this project. With these numbers, we are approximating the lower bound of the labor costs to be around $4,000 per member. Considering our group has 3 members, we are looking at labor costs of around $12,000 for the entire team.

5.2 Parts Cost

Description	Part Name	Quantity	Cost
Raspberry Pi 4	Raspberry Pi 4	1	$ 75.00
ESP-32E Microcontroller	ESP32-S3-MINI-1U-N8	1	$ 3.87
PI Camera Module	Raspberry Pi Camera Module 2	1	$ 18.00

PIR sensor	HC-SR501	3	$ 3.00
Ultrasonic Sensor/transducer	HC-SR04	2	ECE store
Blue LED	Blue LED	3	ECE store
Servo Motor	MG996R	2	$ 18.00
Linear Voltage Regulator	TLV75801PDRVR	1	$ 0.36
TVS Diodes	D3V3XA4B10LP-7	1	$ 0.37
5V Power Adaptor	5V Power Adaptor	1	Free
3D Printer Filament	3D Printer Filament	1	$ 30.00

5.3 Grand Total Cost

The total labor cost for this project was approximated to be $12,000 USD and the cost of parts that are essential to our project is $148.60. Therefore, the grand total cost of this project is $12,148.60 USD.

6. Schedule

Week of	Task	Person(s)
2/26	Design Review	All
	Order parts for prototyping	All
	Develop program for detection	Rishab
	Visit machine shop if needed	All
	Begin designing PCB	Jung Ki / Mankeerat
3/4	Complete and test software subsystem	Rishab
	Finish initial design of PCB, review it and order if possible	Jung Ki / Mankeerat
3/11	Spring Break	All
3/18	Soldering of PCB parts	All
	Ordered parts come in, test software subsystem on Arduino/Rpi	Mankeerat, Rishab

3/25	Integrate the sensors onto the PCB and test	Rishab
	Finish building mechanical subsystem	Mankeerat, Jung ki
4/1	Combine all subsystems	All
	Test for minor bugs	
4/8	Run tests for deterrence	All
4/15	Mock Demo	All
4/22	Final Demo	All
	Mock Presentation	All

7. Ethics and Safety

Ethics:
It's crucial that we adhere to ethical guidelines throughout the duration of this project. One particularly prominent ethical consideration is the potential for causing harm [2]. Within the scope of our project, a primary concern is the welfare of the animals we're trying to prevent entering areas where our system is placed. Our group is committed to upholding this code by prioritizing the well-being of these animals above all else.

As a team, we've recognized that our approach involves influencing the behavior of these species to help them recognize restricted areas. For this reason, to prevent any ethical breaches, we are using indirect methods, such as audio and visual cues, as a means of solving the core problem. We believe this strategy not only safeguards the animals but also mitigates the necessity for harsher measures like pest control, particularly in situations where it is unwarranted.

Safety:
Safety is of the utmost concern to our group. We want to ensure that the application of our device will be safe for both the users and the animals we are targeting with our device. Our first safety concern is the user. With the use of ultrasonic sound, we want to ensure that it is being utilized at a low intensity, since ultrasonic sounds can be damaging to humans even if they are not within the human hearing frequency range. Our second safety concern is with regards to the animals and is related to the use of lights and sounds within our system. We want to ensure that this deterrent system is effective, but also not harmful to them. Therefore, our light system will be implemented such that the intensity will not be damaging to the animal, and the sound system will be run at high frequencies, but not high enough that it will cause damage to the ears of the animals.

Our second safety concern will be the moving parts of our invention. The device will be moving in a 360° fashion, and this will require several moving parts. Therefore, we want to ensure that our design is safe and make sure nothing can get clamped and affect these moving parts. Lastly, is the safety regarding lithium batteries the project will require. Lithium batteries are notorious for igniting, especially in heated settings. Given that our device will be placed outside, we want to make sure that our batteries are sealed in a safe encasing, such that it does not have direct exposure to sunlight.

8. Conclusion

We are extremely proud at how far we have come from the initial planning of this project. Throughout the development of this project we had several obstacles that we needed to overcome, alongside several last minute design changes to make this project a reality. Furthermore, we believe that we can continue this project, and that this initial design is merely a prototype that we can expand on to make it a reality.

8.1 Accomplishments

The project successfully achieved the final functionality we wanted. The code was able to successfully detect, when in vision, multiple different breeds of rodents and animals. Furthermore, once detected, the system was successfully able to follow and apply the deterrence system. With regards to many of the other functionalities we wanted, we were successful in developing a system that could cover a 360° field of view. This was achieved by the adapted design of using two high torque servos alongside one another, since each servo only had the capability to cover a 180° angle. Our deterrence system was heavily reliant on the ultrasonic transducer utilized. While testing, we observed that the ultrasonic transducer, in tandem with flashing LEDs, was successful, and, even in real life scenarios, provided the most effect during the deterrence process.

8.2 Uncertainties

When the deterrent system is activated, both the lights and ultrasonic transducer turn on to repel the rodent. When we tested this outside into the real world environment, we observed that the rodent was being repelled by our device. Since both methods are known to scare rodents away, it is uncertain as to which mechanism was actually responsible for the deterrence. We were not able to individually test each of the deterrence mechanisms due to difficulties in debugging when disabling one of the systems, so we cannot be certain that both the lights and the ultrasonic sounds were scaring the squirrels away.

8.3 Future Work

This project has a lot of potential, and the current implementation we had finished was merely a prototype of something we can easily expand upon and that can be flushed out to be a good product. To reach this stage however, we believe that there are a few improvements that we should make.

Firstly, we believe that we need to improve either our algorithms or compute power. As mentioned before, the current implementation of the system using RPi 4 does not satisfy the computing requirements to perform novel object detection and this is an avenue that we would want to test out more. Additionally, we would like to work on handling multiple objects that are in the view of the camera. Currently, the object detection code has difficulty deciding which object to follow if there are multiple objects in the view. Expanding our project to cover this will improve our device into a more effective one. Lastly, we would like to make the product completely autonomous and improve the packaging. Currently, the system works with a wall connector and is therefore not completely autonomous, due to the need of needing some sort of plug. We would like to adapt this project to utilize a chargeable battery that can be charged using solar panels. Ideally, we would like to compile all of this into a compact and neat packaging. With this, we believe that the future work will allow us to complete robust and usable products for users who struggle with this issue of rodents destroying lawns.

References

[1] M. Bomford and P. H. O'Brien, "Sonic Deterrents in Animal Damage Control: A Review of Device Tests and Effectiveness," Wildlife Society Bulletin (1973-2006), vol. 18, no. 4, pp. 411–422, 1990, Accessed: May 02, 2024.
[2] Association for Computing Machinery, "ACM Code of Ethics and Professional Conduct," Association for Computing Machinery, Jun. 22, 2018. https://www.acm.org/code-of-ethics

Appendix A - Object Detection Code

```
import cv2
from picamera2 import Picamera2
import numpy as np
from led_test import *
from buzzer_test import *
import RPi.GPIO as GPIO
import time


classNames = []
net = None
picam2 = None
lock = False
servo_one = None
servo_two = None

def setup_servo():
   global servo_one, servo_two
   # Servo setup
   servoPinOne = 32 # adjust value
   servoPinTwo = 33
   GPIO.setmode(GPIO.BOARD)

   if servo_one is not None:
       print ("servo one clean")
       servo_one.stop()
       GPIO.cleanup(servoPinOne)
   if servo_two is not None:
       print("servo two clean")
       servo_two.stop()
       GPIO.cleanup(servoPinTwo)


   print("Skipped clcean, we clean either way")
   GPIO.cleanup(servoPinOne)
   GPIO.cleanup(servoPinTwo)

   GPIO.setup(servoPinOne, GPIO.OUT)
   GPIO.setup(servoPinTwo, GPIO.OUT)

   servo_one = GPIO.PWM(servoPinOne, 50)
   servo_two = GPIO.PWM(servoPinTwo, 50)
   servo_one.start(7) # adjust servo values
   servo_two.start(7) # adjust servo values

def set_servo_angle(angle, number):
   global servo_one, servo_two

   if number == 0:
       duty_cycle = 1.0 / 18.0 * angle + 2
       servo_one.ChangeDutyCycle(duty_cycle)
       time.sleep(0.001)
   else:
       duty_cycle = 1.0 / 18.0 * angle + 2
       servo_two.ChangeDutyCycle(duty_cycle)
       time.sleep(0.001)



def getObjects(img, thres, nms, draw=True, objects=['mouse', 'racoon']):
   classIds, confs, bbox = net.detect(img, confThreshold=thres, nmsThreshold=nms)
   objectInfo = []
   if len(classIds) != 0:
       for classId, confidence, box in zip(classIds.flatten(), confs.flatten(), bbox):
           className = classNames[classId - 1]
           if className in objects:
               flash_led()
               flash_buzzer()
```

```
                objectInfo.append([box, classNames[classId - 1], confidence])
                if draw:
                    cv2.rectangle(img, box, color=(0, 255, 0), thickness=2)
                    cv2.putText(img, classNames[classId - 1].upper(), (box[0] + 10, box[1] +
30),
                                cv2.FONT_HERSHEY_COMPLEX, 1, (0, 255, 0), 2)
                    cv2.putText(img, str(round(confidence * 100, 2)), (box[0] + 200, box[1] +
30),
                                cv2.FONT_HERSHEY_COMPLEX, 1, (0, 255, 0), 2)
            else:
                turn_off_led()
                turn_off_buzzer()


    if len(objectInfo) > 0:
        box, _, _ = objectInfo[0]
        center_x = (box[0] + box[2]) // 2
        center_y = (box[1] + box[3]) // 2
        return img, objectInfo, (center_x, center_y)
    else:
        return img, objectInfo, None

def cameraOn():
    global picam2, classNames, net, servo_one, servo_two
    picam2 = Picamera2()
    picam2.configure(picam2.create_preview_configuration())
    picam2.start()

    classFile = "/home/a123/Desktop/Object_Detection_Files/coco.names"
    with open(classFile, "rt") as f:
        classNames = f.read().rstrip("\n").split("\n")

    configPath =
"/home/a123/Desktop/Object_Detection_Files/ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt"
    weightsPath = "/home/a123/Desktop/Object_Detection_Files/frozen_inference_graph.pb"

    net = cv2.dnn_DetectionModel(weightsPath, configPath)
    net.setInputSize(320, 320)
    net.setInputScale(1.0 / 127.5)
    net.setInputMean((127.5, 127.5, 127.5))
    net.setInputSwapRB(True)

    angle = 0

    set_servo_angle(0, 1)
    set_servo_angle(0,0)

    rotation_and_tracking()

def rotation_and_tracking():
    global lock
    angle = 0
    while True:
        # Rotate from 0 to 180 degrees
        for angle in range(0, 361, 10):
            print("Curr angle")
            print(angle)


            if angle > 180:
                send_angle = angle%180
                set_servo_angle(send_angle, 1)
            else:
                set_servo_angle(angle, 0)


            frame = picam2.capture_array()
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            img = cv2.resize(frame, (640, 480))
            result, objectInfo, center = getObjects(img, 0.45, 0.2)
```

```
            print(len(objectInfo))
            if len(objectInfo) > 0:

                lock = True
                print(lock)
                break

            cv2.imshow("Output", result)
            if cv2.waitKey(1) == ord('q'):
                break

        if lock:
            # Track the person
            while lock:
                frame = picam2.capture_array()
                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                img = cv2.resize(frame, (640, 480))
                result, objectInfo, center = getObjects(img, 0.45, 0.2)

                if center is not None:
                    center_x, center_y = center
                    img_width = img.shape[1]
                    img_height = img.shape[0]

                    if center_x < img_width // 4 and angle > 0:
                        angle -= 10
                    elif center_x > img_width * 3 // 4 and angle < 180:
                        angle += 10


                    if angle > 180:
                        send_angle = angle%180
                        set_servo_angle(send_angle, 1)
                    else:
                        set_servo_angle(angle, 0)

                    print(angle)
                cv2.imshow("Output", result)
                if cv2.waitKey(1) == ord('q'):
                    break

                if len(objectInfo) == 0:
                    lock = False


        angle = 360
        # Rotate from 180 to 0 degrees
        for angle in range(360, -1, -10):

            if angle > 180:
                send_angle = angle%180
                set_servo_angle(send_angle, 1)
            else:
                set_servo_angle(angle, 0)


            print("Else")
            print(angle)
            frame = picam2.capture_array()
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            img = cv2.resize(frame, (640, 480))
            result, objectInfo, center = getObjects(img, 0.45, 0.2)

            if len(objectInfo) > 0:
                lock = True
                break

            cv2.imshow("Output", result)
            if cv2.waitKey(1) == ord('q'):
                break
```

```
        if lock:
            # Track the person
            while lock:
                frame = picam2.capture_array()
                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                img = cv2.resize(frame, (640, 480))
                result, objectInfo, center = getObjects(img, 0.45, 0.2)

                if center is not None:
                    center_x, center_y = center
                    img_width = img.shape[1]
                    img_height = img.shape[0]

                    if center_x < img_width // 4:
                        angle -= 10
                    elif center_x > img_width * 3 // 4:
                        angle += 10

                    if angle > 180:
                        send_angle = angle%180
                        set_servo_angle(send_angle, 1)
                    else:
                        set_servo_angle(angle, 0)
                    print(angle)
                cv2.imshow("Output", result)
                if cv2.waitKey(1) == ord('q'):
                    break

                if len(objectInfo) == 0:
                    lock = False


        break

    stop_camera()

def stop_camera():
    global picam2, servo_one, servo_two

    print("Stop cam")
    if picam2 is not None:
        picam2.stop()
        picam2 = None
    cv2.destroyAllWindows()


    if servo_one is not None:
        servo_one.stop()
        GPIO.cleanup(32)

    if servo_two is not None:
        servo_two.stop()
        GPIO.cleanup(33)

    GPIO.cleanup()




import RPi.GPIO as GPIO
import time
import cv2
from picamera2 import Picamera2
import numpy as np
import math
import sys
import threading
from radar import *
from detect import *

# change all these numbers to their respective GPIO pins
```

```
pir_1_in = 11
pir_2_in = 13
pir_3_in = 15

led_out = 37
buzzer_out = 35

# radar code sets up servo for pir/UR tower so we dont worry about this



def main():

   #Initialize all sensors/servos
   setup_servo()
    while True:



       GPIO.setmode(GPIO.BOARD)
       GPIO.setwarnings(False) # Ignore warning for now

       #Inputs
       GPIO.setup(pir_1_in,GPIO.IN)
       GPIO.setup(pir_2_in,GPIO.IN)
       GPIO.setup(pir_3_in,GPIO.IN)

       #Outputs
       GPIO.setup(led_out, GPIO.OUT)
       GPIO.setup(buzzer_out, GPIO.OUT)


       #Check if PIR detects anything

       while True:

           print (GPIO.input(pir_1_in))
           print (GPIO.input(pir_2_in))
           print (GPIO.input(pir_3_in))
           if GPIO.input(pir_1_in) == 1 or GPIO.input(pir_2_in) == 1 or GPIO.input(pir_3_in)
== 1:
               print("Motion detected")
               break

       #If does start radar
       radar_loop()

       #Once radar stops start camera
       cameraOn()


       print("Rodent deterred")


       GPIO.cleanup()

main()




import RPi.GPIO as GPIO
import time
import threading


# Set the GPIO mode to BCM (Broadcom SOC channel numbering)
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False) # Ignore warning for now
# Set the GPIO pin for the LED (you can change this to any available GPIO pin)
led_pin = 7 #change later
```

```
led_flashing = False
frequency = 1
GPIO.setup(led_pin, GPIO.OUT)
GPIO.output(led_pin, GPIO.LOW)


def flash_led():
   global led_flashing
   led_flashing = True

   def flash_thread():
       while led_flashing:
           GPIO.output(led_pin, GPIO.HIGH)
           time.sleep(0.15)
           GPIO.output(led_pin, GPIO.LOW)
           time.sleep(0.15)

   flash_thread = threading.Thread(target=flash_thread)
   flash_thread.start()

# Define a function to turn off the LED
def turn_off_led():
   global led_flashing
   led_flashing = False
   GPIO.output(led_pin, GPIO.LOW)


#turn_off_led()
import RPi.GPIO as GPIO
import time

# Set the GPIO mode
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False) # Ignore warning for now

# Set the GPIO pin for the ultrasonic transducer
TRIG_PIN = 32

# Set the PWM frequency and duty cycle
PWM_FREQ = 10000  # 12 kHz
DUTY_CYCLE = 50   # 50% duty cycle

# Set up the GPIO pin as an output
GPIO.setup(TRIG_PIN, GPIO.OUT)

# Create a PWM object
pwm = GPIO.PWM(TRIG_PIN, PWM_FREQ)

# Start the PWM signal
pwm.start(DUTY_CYCLE)

# Set the total duration for the loop
TOTAL_DURATION = 5  # 5 seconds

# Get the start time
start_time = time.time()

# Generate bursts of pulses until the total duration is reached
while time.time() - start_time < TOTAL_DURATION:
   # Generate a burst of pulses for a specific duration
   BURST_DURATION = 0.0025  # 250 microseconds
   pwm.ChangeFrequency(PWM_FREQ)
   time.sleep(BURST_DURATION)

# Stop the PWM signal
pwm.stop()

# Clean up the GPIO pins
GPIO.cleanup()
```