ECE 445

SENIOR DESIGN LABORATORY

FINAL REPORT

Actions to Mosquitoes

Team #4

XIANGMEI CHEN (xc47@illinois.edu)
PEIQI CAI (peiqic3@illinois.edu)
YANG DAI (yangdai2@illinois.edu)
LUMENG XU (lumengx2@illinois.edu)

TA: Guo Hao Thng
Sponsor: Said Mikki

May 9, 2024

Abstract

In this project, we design a machine that can detect the existence of mosquitoes by sound, and localize them using camera, then track and suck them by the moving of the car and the rotation of fan, thus complete our actions to mosquitoes.

The detection subsystem is crucial, using an microphone to record sounds and detect mosquito wingbeats from 300 to 600 Hz. It applies MFCC method and CNN to process the audio.

The localization subsystem is equipped with a high-resolution camera and employs the Roboflow Train 3.0 model for real-time mosquito detection and spatial localization. This subsystem ensures precise identification of mosquitoes, even at a distance of 1-2 meters, and with the time delay in 4 seconds.

The attack subsystem comprises a fan-based capture unit and a mechanical structure for precise positioning. The fan operates on powerful suction and is activated upon mosquito detection, ensuring efficient capture. The mechanical structure, featuring a chassis with wheels and a lead screw for 360-degree rotation, is designed for stability and maneuverability.

The power and control subsystem serves as the backbone, providing stable power supply and voltage regulation. It includes a Raspberry Pi-based control unit for processing sensory data and generating commands for the motor control system, which in turn manages the precise movements of the machine via PWM signals. Also a PCB board is applied to do the voltage switch from 12V to 5V to power the Raspberry Pi.

Overall, the project's design adheres to ethical standards, ensuring privacy in monitoring and humane mosquito elimination, with a strong emphasis on safety protocols throughout the development process.

Contents

1 Introduction **1**
- 1.1 Problem and Solution Overview . . . 1
- 1.2 Functionality . . . 1
  - 1.2.1 Detection Subsystem . . . 1
  - 1.2.2 Localization Subsystem . . . 2
  - 1.2.3 Attack Subsystem . . . 2
  - 1.2.4 Power and Control Subsystem . . . 2
- 1.3 Subsystem Overview . . . 3
  - 1.3.1 Detection Subsystem . . . 4
  - 1.3.2 Localization Subsystem . . . 4
  - 1.3.3 Attack Subsystem . . . 5
  - 1.3.4 Power and Control Subsystem . . . 6

2 Design **9**
- 2.1 Audio Detection Module . . . 9
  - 2.1.1 Design Ideas and Algorithm . . . 9
  - 2.1.2 Neural Network Description . . . 10
- 2.2 Computer Vision Module . . . 11
  - 2.2.1 Design Ideas . . . 11
  - 2.2.2 Data Augmentation . . . 12
  - 2.2.3 Inference Acceleration . . . 14
- 2.3 Movement and Rotation Module . . . 16
  - 2.3.1 Configuration Space Calculation . . . 16
  - 2.3.2 Finite State Machine Design . . . 19
  - 2.3.3 PWM Design . . . 20
- 2.4 Control and Integrated Module . . . 21
  - 2.4.1 Deployment of the audio detection to Raspberry Pi . . . 21
  - 2.4.2 Deployment object detection model to Raspberry Pi . . . 22
  - 2.4.3 Connection and Control of Motors . . . 23
- 2.5 Mechanical Structure . . . 26
  - 2.5.1 Design description and drawings . . . 26
  - 2.5.2 Simulation results . . . 28
  - 2.5.3 Design alternatives . . . 30

3 Cost and Schedule **34**
- 3.1 Cost Analysis . . . 34
  - 3.1.1 Cost of labor . . . 34
  - 3.1.2 Cost of parts . . . 34
  - 3.1.3 Sum of Costs . . . 35
- 3.2 Schedule . . . 35

4 Requirements and Verification **37**
- 4.1 Audio Detection Module . . . 37

4.1.1 Model Training Accuracy . . . 37
4.1.2 Audio Processing and Detection Latency . . . 37
4.2 Computer Vision Module . . . 38
4.2.1 Model Metrics . . . 38
4.2.2 FPS and Inference Latency . . . 39
4.3 Movement and Rotation Module . . . 40
4.3.1 Camera Activation Time Delay . . . 40
4.3.2 Servo Motor Response . . . 41
4.3.3 Car Rotation Sensitivity . . . 41
4.4 Control and Integrated Module . . . 41
4.4.1 Battery for Stable Power Supply and Switch . . . 41
4.4.2 Motor Control for Movement and Capture . . . 42

5 Conclusion **43**
5.1 Accomplishments . . . 43
5.2 Uncertainties . . . 43
5.2.1 Uncertainty in Localization Subsystem . . . 43
5.3 Future Work / Alternatives . . . 43
5.4 Ethical Considerations . . . 44

References **45**

1 Introduction

1.1 Problem and Solution Overview

Mosquitoes are not just a source of irritation due to their itchy bites; they are also public health threats, as documented by the World Health Organization (WHO), which identifies them as vectors for diseases like malaria and dengue [1]. The challenge of controlling these agile insects is compounded by the limitations of current methods, which can be less effective and potentially harmful, as noted in studies on the environmental impact of mosquito control. To address these issues, we've developed an innovative device that actively captures mosquitoes. It operates by moving through the environment and swiftly sucking up mosquitoes upon detection, offering a more targeted and safer alternative to traditional repellents and swatters.

We intend to design our project by four subsystems: a detection subsystem, a localization subsystem, an attack subsystem, and a power and control subsystem. The detection subsystem serves as the trigger, using audio cues to activate the machine when mosquitoes are present. The localization subsystem employs a camera to locate the mosquito and provides real time location data to the attack subsystem, which then mobilizes to capture or eliminate the mosquitoes using a powerful suction device and CO2, heat, and motion-based lures. The power and control subsystem is strategically divided to supply continuous energy to the detection subsystem and activated power to the localization and attack subsystems, optimizing energy usage, and ensuring sustained operations.

Our design can be implemented equipped with some subsystem requirements. Firstly, the detection subsystem requires high sensitivity and accuracy, with a minimum detection accuracy of 90% and a false positive rate below 10%. Secondly, the localization subsystem demands a camera capable of identifying mosquitoes with at least 80% accuracy and providing real time data to the attack subsystem, which must possess precision mobility and an effective attractant mechanism for mosquito capture. What's more, the power and control subsystem is tasked with stable and efficient power delivery, featuring voltage regulation, surge protection, and a failsafe mechanism to ensure the seamless operation of the machine. Collectively, these subsystems form a comprehensive solution for mosquito detection, tracking, and elimination, emphasizing efficiency, accuracy, and safety.

1.2 Functionality

1.2.1 Detection Subsystem

The detection subsystem is the cornerstone of our mosquito eradication device, designed to identify the presence of mosquitoes through their unique wingbeat frequency. This subsystem operates using an acoustic sensor array that captures and processes sound waves within the 300 to 600 Hz range, which is specific to mosquitoes. By leveraging the Mel-Frequency Cepstral Coefficient (MFCC) and a machine learning model, the subsystem can discern mosquito sounds with a high degree of sensitivity and accuracy, ensuring that the device is only activated in the presence of the target species.

The detection subsystem's high sensitivity and accuracy are critical for the device's effectiveness. By accurately identifying mosquito presence, the device can operate efficiently, conserving energy and resources until needed, directly contributing to the project's goal of effective mosquito management.

1.2.2 Localization Subsystem

The localization subsystem builds upon the detection phase by visually identifying the mosquito's location using a high-resolution USB camera connected to a Raspberry Pi. The subsystem employs the Roboflow Train 3.0 model, a state-of-the-art algorithm for real-time object detection, to pinpoint the mosquito's position within the camera's field of view. This subsystem is capable of adjusting the attack subsystem's height to align with the mosquito's altitude, facilitating a precise and targeted approach.

The localization subsystem's precision is essential for the device's ability to navigate towards and capture mosquitoes. By accurately determining the mosquito's location, the device can effectively eliminate the target, contributing to the reduction of mosquito-borne diseases and aligning with the project's public health mission.

1.2.3 Attack Subsystem

The attack subsystem is tasked with the physical capture and elimination of mosquitoes. It is equipped with a fan capture unit that generates an airflow to draw in mosquitoes and a mechanical structure that allows for precise positioning. The subsystem's mechanical structure, including the chassis, wheels, and lead screw, enables 360-degree rotation and accurate movement towards the mosquito's location. The servo motors and control unit work in tandem to ensure the device can maneuver effectively for capture. Relation to Overall Purpose: The attack subsystem's efficiency and precision are paramount to the device's eradication capabilities. By effectively capturing and eliminating mosquitoes, the subsystem plays a direct role in reducing the mosquito population, thereby addressing the project's aim of improving public health and safety.

1.2.4 Power and Control Subsystem

The power and control subsystem is the central hub of the device, ensuring that all other subsystems function harmoniously. It comprises a control unit based on the Raspberry Pi, which processes sensory data and formulates commands for the motor control system. The motor control system, in turn, generates PWM signals to manage the motors, enabling the precise movement and positioning of the device. Additionally, the power supply unit provides a stable and regulated power source, ensuring uninterrupted operation.

The power and control subsystem is indispensable for the device's autonomous functionality. It ensures that the device can process sensory data, make informed decisions, and execute actions to eradicate mosquitoes effectively. This subsystem's stability and

efficiency are foundational to the project's success in creating a reliable and effective mosquito control solution.

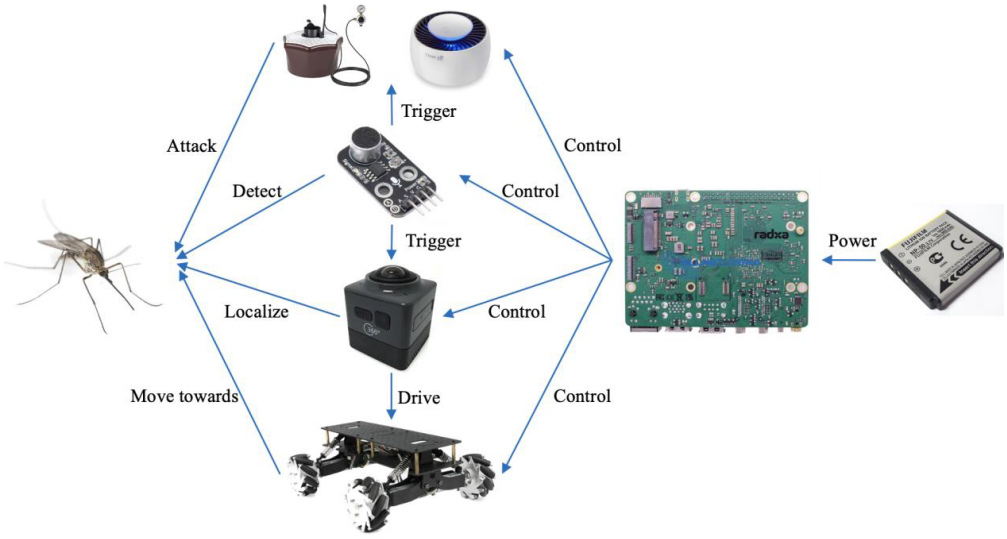


Figure 1: The overall visual graph of the design: One motor chip powers and controls all other parts, the microphone works as a trigger to enable the camera and the attacker, then the main part starts to localize and move to attack the mosquitoes.

1.3 Subsystem Overview

1. The microphone in the detection subsystem must be directly connected to the Raspberry Pi, sensitive and efficient to mosquitoes' noise, and can also record the sound in the environment, which means it should trigger the machine only if there is noise caused by mosquitoes in its working area, and it should distinguish the noise of mosquitoes from other noises.

2. It is significant for the camera to determine direction of the mosquito once it catches mosquito in its vision, so that the machine can adjust its moving according to the action of mosquitoes.

3. We design the attack subsystem only for mosquitoes, it should contain some materials that can attract them, as well as sucking mosquitoes accurately into itself to make the work efficiently.

This mosquito eradication machine is designed to detect, locate, attack, and eliminate mosquitoes autonomously. It comprises four main subsystems, each playing a crucial role in the machine's operation and interacting seamlessly with one another to achieve the goal of mosquito eradication.

1.3.1 Detection Subsystem

This subsystem, ensuring continuous surveillance, is equipped with an acoustic sensor array that captures sound waves and processes them to determine if mosquito activity is detected. The acoustic sensor is capable of distinguishing the unique wingbeat frequency of mosquitoes, which is typically between 300 to 600 Hz for most species [2]. Upon detecting a mosquito's presence, this subsystem initiate the machine's response cycle. This ensures energy efficiency by only activating the more power intensive components when necessary.

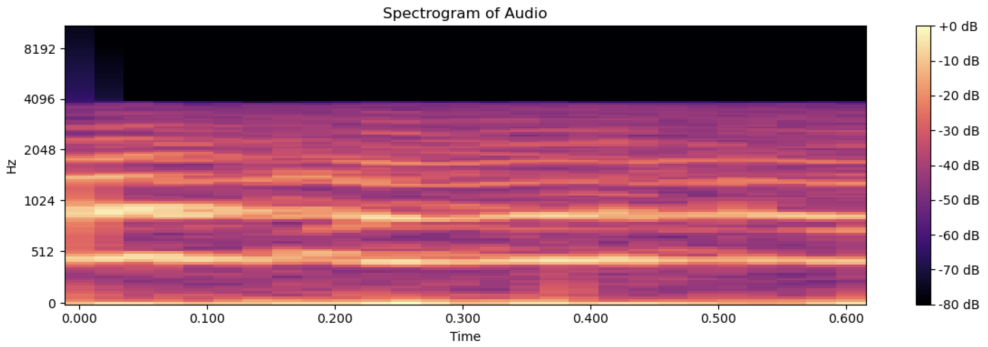


Figure 2: The frequency plot of some mosquitoes wingbeats noise.

For the microphone, we had bought a microphone for Raspberry Pi 4B, which is designed for our Raspberry Pi to use. As for the algorithm to distinguish the noise of mosquitoes, we plan to use the Mel-Frequency Cepstral Coefficient (MFCC) combined with machine learning model to train. MFCC is a feature extraction method used in audio processing to represent the short-term power spectrum of a sound, which can also be used to classify sounds according to the difference in frequency.

1.3.2 Localization Subsystem

For this subsystem, we plan to use the USB camera connected to the Raspberry Pi. This subsystem integrates advanced image processing algorithms to analyze the captured images. The camera's high resolution ensures that even small targets like mosquitoes can be clearly detected. These images are then processed to identify the mosquito's location in the space.

As for the algorithm, We will adopt the existing Roboflow Train 3.0 model for real-time coordinate detection and recognition of mosquitoes [3]. Comparing to other computer vision model architectures, the model has accuracy on par as well as faster training and inference speed.

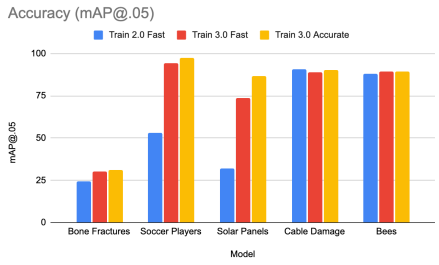


Figure 3: Accuracy of Roboflow 3.0.

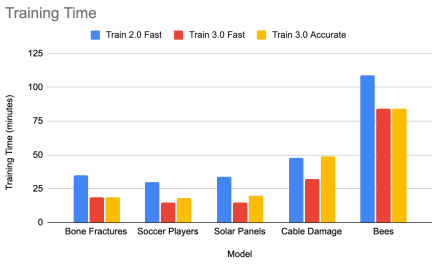


Figure 4: Training time of Roboflow 3.0.

The integrated dataset comes from public datasets online, and we will fine-tune it with data collected by ourselves. This subsystem's feedback loop with the attack subsystem allows for dynamic adjustment of the machine's position and orientation, optimizing the capture process.

Taking into account the actual environmental conditions, we will adjust the model's confidence threshold to ensure that while reducing the false positives of mosquito entities, we simultaneously increase the accuracy of mosquito detection.

1.3.3 Attack Subsystem

1.3.3.1 Overview

The control system is the central nervous system of the mosquito eradication machine, orchestrating the interplay between the power subsystem and the operational components. It comprises a control unit integrated within the Raspberry Pi, which processes input from the microphone and camera, and a motor control system that executes the commands to maneuver the machine. It is crucial for the machine's primary function of mosquito eradication. It ensures that once a mosquito is detected, the machine can effectively capture and eliminate it, contributing directly to the reduction of mosquito-borne diseases.

1.3.3.2 Fan Capture Unit

Description: The fan subsystem is a central component of the attack system, designed to capture mosquitoes. The fan operates to create a airflow that sucks in mosquitoes, facilitating their capture by the machine.

Interfaces: The fan is connected to the servo motor and is controlled by the control unit to activate when a mosquito is detected and ready to be captured.

Contribution to Overall Design: The fan's role in capturing mosquitoes is essential for the machine's efficacy. It ensures that once a mosquito is within range, it is effectively captured, preventing escape and enabling elimination.

1.3.3.3 Mechanical Structure and Positioning Unit

Description: The mechanical structure subsystem includes the chassis, wheels, and lead

screw that support 360-degree rotation. This subsystem, in conjunction with servo motors, allows the machine to position itself accurately for optimal mosquito capture

Interfaces: The mechanical structure interfaces with the servo motors through the motor control system, which receives commands from the control unit via GPIO.

Contribution to Overall Design: The mechanical structure provides the necessary mobility and precise positioning, ensuring that the machine can effectively capture mosquitoes from any detected location within its environment.

1.3.3.4 Overall process

Upon startup, the machine system activates its microphone, which immediately begins to function. Once the microphone detects the sound of a mosquito, the system automatically triggers the camera and the servo motor that controls the rotation of the lead screw. The purpose of the servo motor is to rotate the lead screw, with one end connected to the camera and the other to the mobility wheels. Through the rotation of the lead screw, the camera is capable of a 360-degree panoramic scan, ensuring that the direction of the wheels aligns with the camera's field of view.

When the camera captures an image of a mosquito, the servo motor connected to the wheels lowers them to the ground. Subsequently, the motor or electric motor associated with the fan and wheels activates, propelling the wheels towards the target direction that the camera is focused on, with the fan responsible for capturing the mosquito.

To achieve precise tracking of the mosquito, the two servo motors that control the rotation of the lead screw and the lifting and lowering of the wheels will receive PWM signals from the Raspberry Pi based on the mosquito's position coordinates in the camera's view, making corresponding adjustments to ensure that the mosquito remains centered in the camera's field of view at all times.

1.3.4 Power and Control Subsystem

1.3.4.1 Overview

The control system is the central nervous system of the mosquito eradication machine, orchestrating the interplay between the power subsystem and the operational components. It comprises a control unit integrated within the Raspberry Pi, which processes input from the microphone and camera, and a motor control system that executes the commands to maneuver the machine.

1.3.4.2 Control Unit

Description: The control unit, based on the Raspberry Pi, is responsible for real-time data processing from the microphone and camera. It runs advanced algorithms to distinguish mosquito sounds and identify mosquito positions. Based on this processed data, the control unit formulates commands for the motor control system.

Interfaces: USB interfaces for connecting the microphone and camera. Serial or I2C communication links with the motor control system for command transmission.

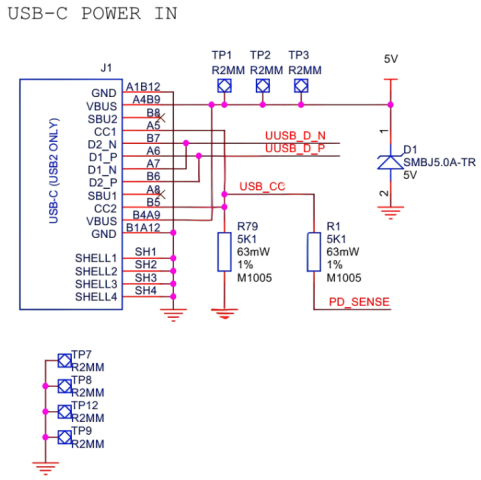


Figure 5: USB circuit of Raspberry Pi.

Contribution to Overall Design: The control system is essential for the machine's autonomous functionality. It processes sensory data, makes informed decisions, and coordinates the machine's movements and operational mechanisms, ensuring effective mosquito eradication.

1.3.4.3 Motor Control System

Description: The motor control system interprets commands from the control unit and generates appropriate PWM signals to manage the motors. It is responsible for the precise movement and positioning of the machine, including the rotation of the camera and the movement of the capture mechanism.

Contribution to Overall Design: The motor control system interprets commands from the control unit and generates appropriate PWM signals to manage the motors. It is responsible for the precise movement and positioning of the machine, including the rotation of the camera and the movement of the capture mechanism.

Interfaces: GPIO pins on the Raspberry Pi are used to generate PWM signals for motor control.

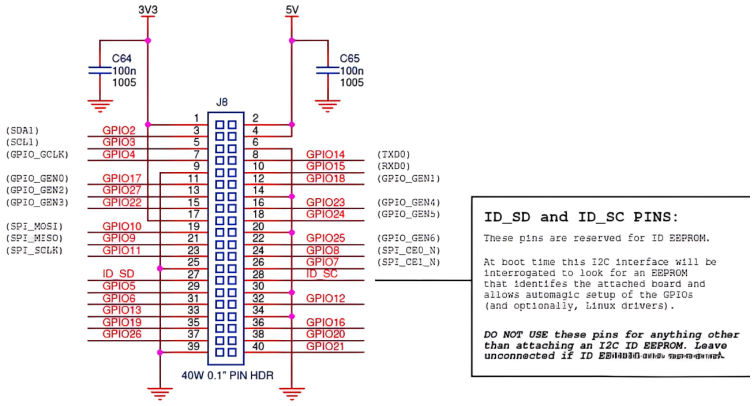


Figure 6: GPIO Pins expansion of Raspberry Pi.

PWM signals to servos and motors. Motor driver modules, L298N, interface with the Raspberry Pi and the motors.

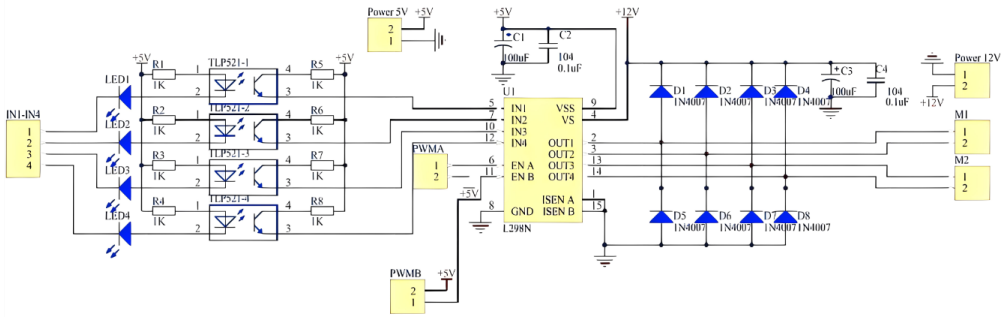


Figure 7: L298N.

1.3.4.4 Power Supply Unit (PSU)

Description: The PSU is the energy source of the machine, consisting of a battery and an adapter. The battery provides a stable 12V power supply, and the adapter converts this to the appropriate voltage levels required by each component.

Contribution to Overall Design: The PSU ensures that all components receive a stable and consistent power supply, which is crucial for reliable operation and performance of the machine.

Interfaces: Direct connections to the control unit, motor control system, and other powered components.

2 Design

2.1 Audio Detection Module

2.1.1 Design Ideas and Algorithm

The detection subsystem forms the first line of our mosquito attack device, leveraging acoustic data to detect the presence of mosquitoes. Utilizing an microphone to record sounds, the subsystem captures audio signals within a specific frequency range known to be characteristic of mosquito wingbeats, which typically lie between 300Hz to 600 Hz.

The detection process starts with audio signal capture via the microphone. The algorithm processes these audio inputs to extract relevant features that help differentiate mosquito noises from other ambient sounds. The primary feature extraction method used here is Mel-Frequency Cepstral Coefficients (MFCC). MFCCs are crucial in this context as they efficiently represent the power spectrum of audio signals, capturing the essential characteristics needed for mosquito identification. Its calculation formula is as follows:

$$\text{MFCCs} = 20 \cdot \log_{10}\left(|\text{FFT}\left(\text{Window} \cdot \text{Signal}\right)|^2\right)$$

The extracted MFCC features are then utilized to determine the presence of mosquitoes through a classification process. While the specifics of the model used for classification are detailed in a subsequent section, it's important to note that the chosen model processes these features to accurately identify mosquito-related audio.

The provided code demonstrates the practical application of these methods. Libraries such as `librosa` are used for audio processing, specifically for loading audio files and extracting MFCC features. The extracted features are crucial inputs for the machine learning model responsible for the final classification of sounds.

```python
import librosa
import numpy as np

def load_and_extract_features(audio_path, sample_rate=16000):
    audio_data, _ = librosa.load(audio_path, sr=sample_rate)
    mfccs = librosa.feature.mfcc(y=audio_data, sr=sample_rate, n_mfcc=40)
    feature_vector = np.mean(mfccs, axis=1)
    return feature_vector
```

Listing 1: Python code for extracting MFCC features from audio data.

In this snippet, audio files are loaded and processed to extract MFCC features, which are then averaged across time to create a consistent feature set for each audio clip. This process ensures that the subsystem can efficiently handle real-time audio data, making timely and accurate detections possible.

2.1.2 Neural Network Description

The detection subsystem utilizes a Convolutional Neural Network (CNN) to classify audio features extracted as Mel-Frequency Cepstral Coefficients (MFCCs), distinguishing mosquito sounds from other ambient noises.

The CNN architecture comprises:

- **Input Layer:** Processes input MFCCs, a time-series representation of audio.
- **Convolutional Layers:** Multiple layers with ReLU activation functions extract patterns from the audio data.
- **Pooling Layers:** Max pooling layers follow convolutional layers to reduce dimensionality and prevent overfitting.
- **Fully Connected Layers:** One or more layers that finalize the classification process.
- **Output Layer:** A softmax activation function provides the probabilities for each class, enabling a clear classification decision.

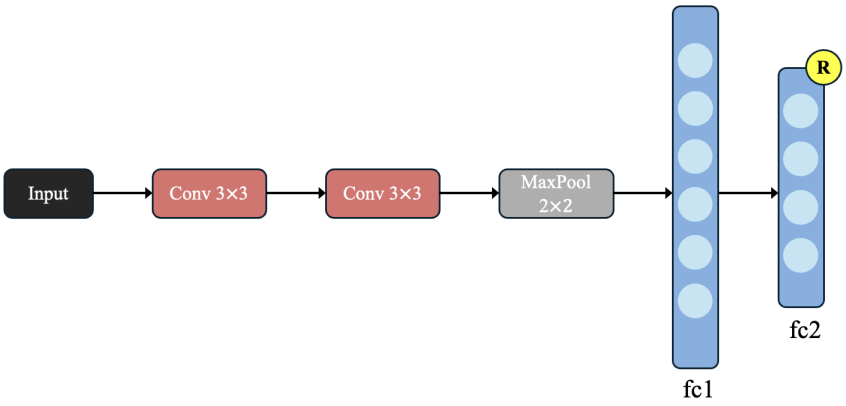


Figure 8: The structure of the CNN.

The CNN is trained using a dataset of labeled mosquito and non-mosquito sounds, adjusting weights and biases through backpropagation to minimize cross-entropy loss. Optimization is performed using algorithms like Adam. The mathematical expression of the calculation of the CNN can be expressed as:

$$y = \sigma\left(W_2 \cdot \text{ReLU}\left(W_1 \cdot x + b_1\right) + b_2\right)$$

Where:

- x is the input vector (MFCCs).
- W_1, W_2 are the weights of the first and second layers.
- b_1, b_2 are biases for the first and second layers.

- ReLU is the Rectified Linear Unit activation function.
- σ is the softmax function applied at the output layer.

A simplified PyTorch implementation of the CNN is shown below:

```
import torch
import torch.nn as nn
import torch.optim as optim

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1),
    padding=(1, 1))
        self.conv2 = nn.Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1),
    padding=(1, 1))
        self.pool = nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2))
        self.fc1 = nn.Linear(204800, 128)
        self.fc2 = nn.Linear(128, 2)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = x.view(x.size(0), -1)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Listing 2: Simplified CNN architecture implemented in PyTorch.

The trained network is evaluated using accuracy, precision, recall, and F1-score on a separate validation set, ensuring it meets the project's requirements for sensitivity and specificity.

2.2 Computer Vision Module

2.2.1 Design Ideas

The vision subsystem is a pivotal element of our mosquito attack device, employing the Roboflow 3.0 model for enhanced object detection capabilities. This model facilitates the identification and tracking of mosquitoes using visual data captured through a camera.

The process initiates with the camera capturing video frames, which are then fed into the Roboflow 3.0 model. The model utilizes advanced algorithms to process the video inputs and extract features that are instrumental in distinguishing mosquitoes from their surroundings. The Roboflow 3.0 model is particularly adept at handling complex visual patterns and providing high accuracy rates, thanks to its improved training infrastructure1.

The Roboflow 3.0 model is trained on a dataset of annotated images, where it learns to recognize and localize mosquitoes within the frames. The training process involves fine-tuning the model parameters to achieve optimal performance, with a focus on maximizing the mean Average Precision (mAP) at a specific Intersection over Union (IoU) threshold.

2.2.2 Data Augmentation

The input of the model during training will be the images of 640×640 pixels, which will be further augmented based on several strategies to improve training performance shown as below.

1. Static Crop: This step crops each image to the specified section. After analyzing the dataset, most of the mosquitoes stays in the middle of the image. To avoid the loss of too many objects after cropping, the horizontal and vertical range of cropping is set to 15% to 85%.

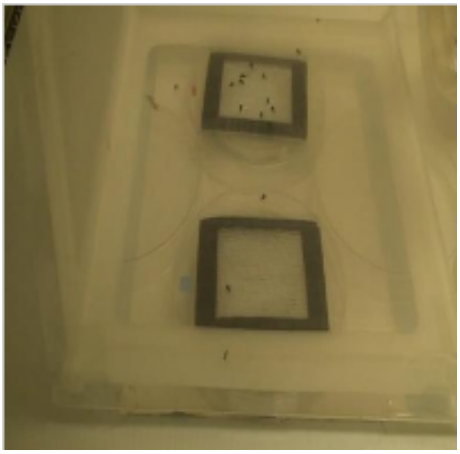


Figure 9: Original.

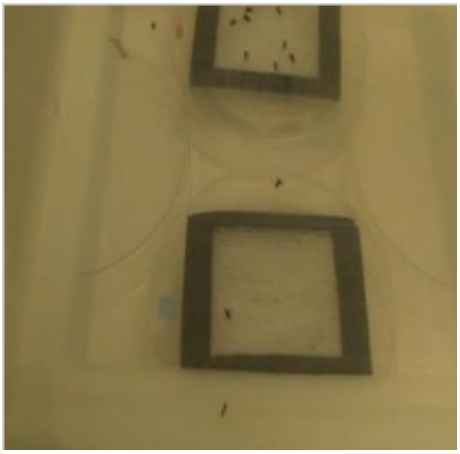


Figure 10: Cropped.

2. 90° Rotate: This step add 90-degree rotations to help the model be insensitive to camera orientation. As stated, the environment of the dataset is set to be constant, while the real situation could be different since our design requires the whole structure to rotate and move to capture the mosquito. Hence, the step could improve the robustness of the model.

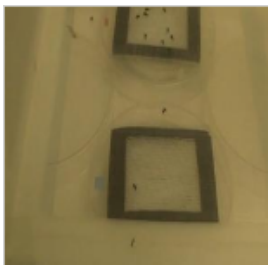


Figure 11: Preprocessed.

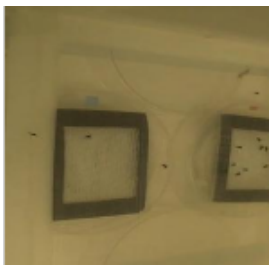


Figure 12: Clockwise.

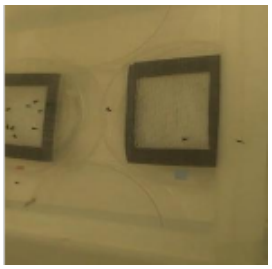


Figure 13: Counter-clockwise.

3. Brightness: This step add variability to image brightness to help the model be more resilient to lighting and camera setting changes, since the time as well as environment may affect the brightness of the captured image. Both brighten and darken images are considered and brightness is set to -15% to 15%.

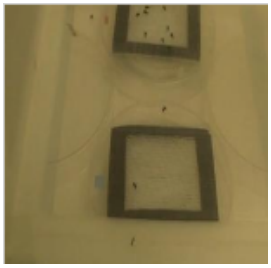


Figure 14: 0%.

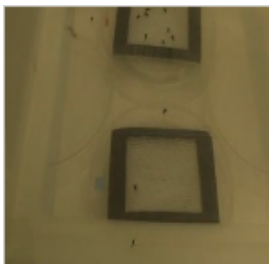


Figure 15: -15%.

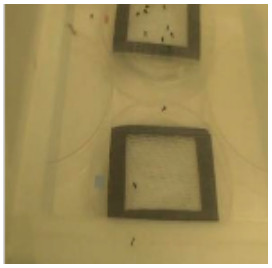


Figure 16: 15%.

After data augmentation, the amount of the dataset is doubled to 2258 in total, and is further split to the fraction: $85\%, 10\%, 5\%$, as the train-valid-test set. The training graphs of essential metrics are shown below. Comparing to other State-of-art object detection model, our model has performance on par after data augmentation based on mAP (mean Average Precision) score, which is given by

$$mAP = \frac{1}{n} \sum_{k=1}^{n} AP_k$$

.

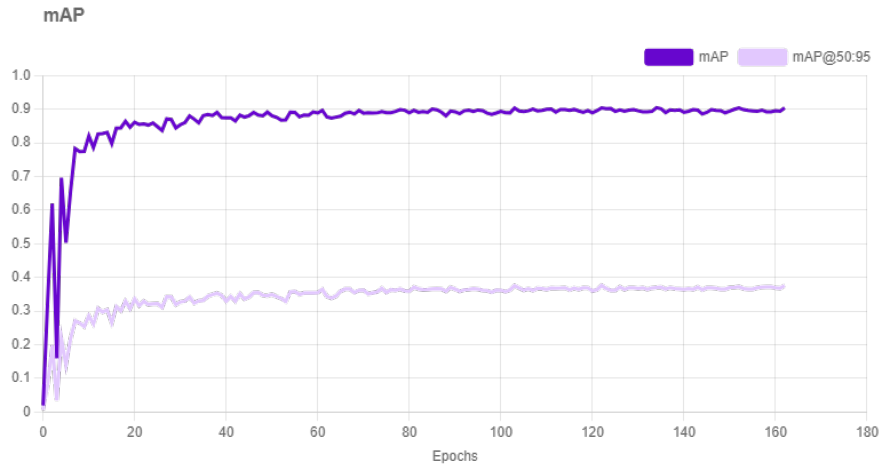


Figure 17: mAP and mAP@50:95 score

Figure 18: Box Loss.

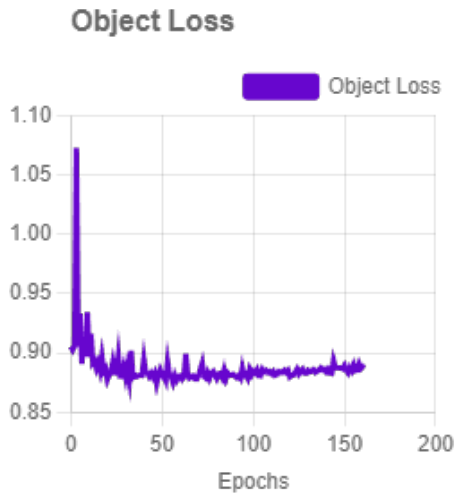


Figure 19: Object Loss.

2.2.3 Inference Acceleration

Design issue:

One of the greatest challenging we are facing in Localization subsystem is the time taken for processing each frame was higher than desired, leading to delays in mosquito detection. This was quantified by the latency equation.

$$L = \frac{1}{N} \sum_{i=1}^{N} (t_{\text{response},i} - t_{\text{request},i})$$

where L is the average latency, N is the number of requests, and $t_{\text{response},i}$ and $t_{\text{request},i}$ are the response and request times for the i-th inference. Due to the performance constraint

of Raspberry Pi 4B, object detection models with large number of parameters such as YOLOv8, which we desired to use at the beginning, are abandoned.

Corrective Actions Taken:

The first measurement is to adopt model with fewer parameters, which we have mentioned above as Roboflow Train 3.0, which has faster speed in training and inference.

Another way is to use inference server service using Python package `inference` and set up our own Self-Hosted Inference Server. The basic principle diagram of inference server is illustrated as below.

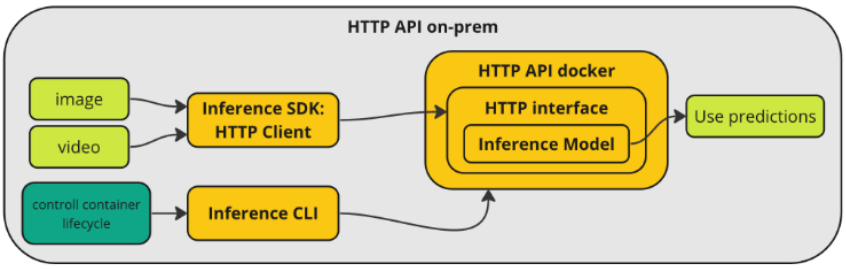


Figure 20: The diagram of using inference over HTTP.

Our linux server works on a Ubuntu 22.04 server with 64bit and a dual-core CPU of 2 threads and is set up in Shanghai, so that there will not be too much latency due to internet connection. The overall latency can be quantified as:

$$L_{\text{server}} = \frac{1}{N}\sum_{i=1}^{N}(t_{\text{response\_server},i} - t_{\text{request\_server},i} + \text{Internet latency})$$

```
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         42 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  2
  On-line CPU(s) list:   0,1
```

Figure 21: CPU information.

Since the direct distance from ZJUI to Shanghai is within 100 km, the internet latency can be calculated as

$$L_{\text{internet}} = \frac{s}{v} = \frac{100\text{km}}{300,000\text{km/second}} \approx 0.4\text{ms}$$

.

Although, according to the real situation the latency may not be ideal, it's still acceptable comparing to deploy the model locally on Raspberry Pi.

Design Details:

On the server, we install Docker firstly, set up and install and Inference server using:

```
pip install inference-cli && inference server start
```

On the edge device, i.e., Raspberry Pi, we first connect the server and the device using SSH on port 9001 which inference server automatically open:

```
ssh -N -f -L localhost:9001:0.0.0.0:9001 root@Public IP
```

and run the following Python code on Raspberry Pi:

```
from inference_sdk import InferenceHTTPClient

# initialize the client
CLIENT = InferenceHTTPClient(
    api_url="http://localhost:9001",
    api_key="api key given by Roboflow"
)

srcimg = CLIENT.infer(frame, model_id="model id given by Roboflow")
```

Listing 3: Python code to run the model on edge devices.

where we can get the result coordinates as well as the width and height of the bounding box in srcimg['predictions'].

2.3 Movement and Rotation Module

The Movement and Rotation Module is a critical component of our mosquito detection and elimination system. This module's primary objective is to facilitate precise maneuvering and positioning of the machine, enabling it to effectively track and approach mosquitoes for capture.

The core of this module's operation lies in its integration with the Roboflow object detection algorithm and custom-designed bounding box detection. These technologies are utilized to determine the position of mosquitoes within the captured images, a process crucial for directing the system's movements accurately.

To control the machine's movement, PWM (Pulse Width Modulation) signals are sent to the motors based on the positional data provided by the detection algorithms. This setup allows for dynamic adjustment of motor speeds, ensuring optimal movement patterns for approaching the target. The module controls three omnidirectional wheels, enabling advanced maneuvers such as forward movement, turning, and on-the-spot rotation. These capabilities are essential for the system's operational efficiency, especially in environments with complex obstacle layouts.

2.3.1 Configuration Space Calculation

The primary goal is to keep the mosquito centered within the camera's view, facilitating effective tracking and eventual capture.

Design Procedure:

In the mosquito tracking system, the configuration space is defined by the camera's field of view, which measures 640×480 pixels. The motor adjustments are determined based on the detected position of the mosquito within this field:

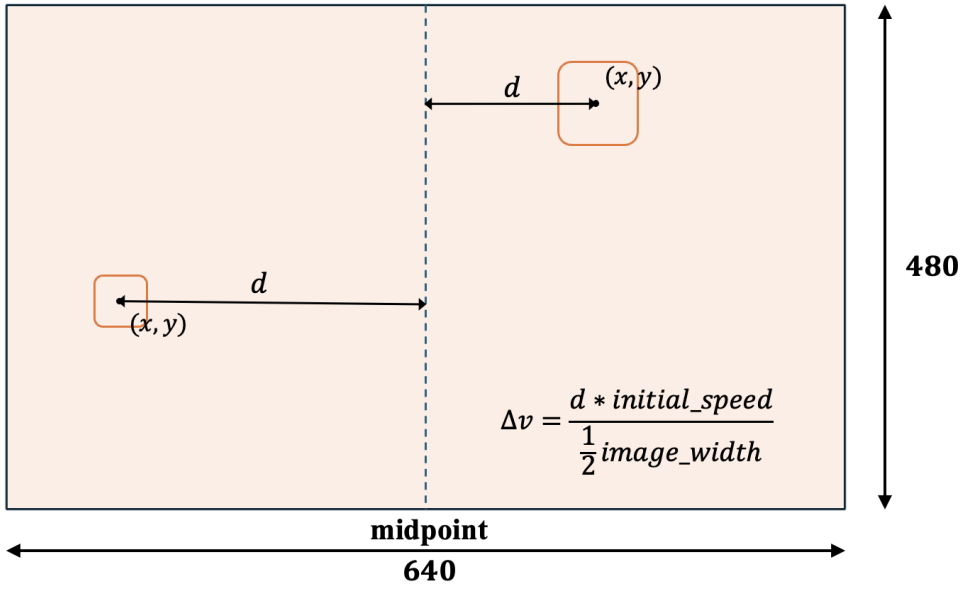


Figure 22: Visual Detection Boundary of Camera View.

- **Mosquito Detection:** Using a USB camera connected to a Raspberry Pi, the system captures real-time video footage. The video frames are processed using the Roboflow model, a robust object detection algorithm, to identify and locate mosquitoes with high accuracy.
- **Position Extraction:** From the detection, the bounding box coordinates (x, y, w, h) are derived, where (x, y) represents the center of the bounding box and (w, h) is the width and height. These coordinates help in determining the mosquito's precise location in the frame.
- **Adjustment of Motor Speed:** Based on the mosquito's position x relative to the center of the image. the system calculates necessary adjustments in motor speeds. The objective is to align the mosquito's position with the center of the camera's field, optimizing tracking and positioning.

Design Details:

The operational logic is as follows:

Mosquito Position Analysis

- **Center Extraction:** The center (x, y) of the bounding box is used to determine the mosquito's current position.

```
    x, y, w, h = int(prediction['x']), int(prediction['y']),
            int(prediction['width']), int(prediction['height'])
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
    cv2.putText(frame, f"{'mosquitoes'} {prediction['confidence']:.2f}",
            (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)
```

Listing 4: Bounding Box Coordinates.

- **Relative Position Calculation:** Calculates how far x is from the image's midpoint to determine necessary motor adjustments.

Motor Speed Adjustment

- **Adjustment Factor Calculation:**

$$\text{Speed Adjustment} = \text{Specific Speed} - \left(\frac{|x_{\text{position}} - x_{\text{mid}}|}{x_{\text{mid}}} \times \text{Specific Speed} \right)$$

  Where x_{mid} is the midpoint of the image width, and x_{position} is the horizontal position of the mosquito. This notation clearly defines the adjustment needed based on the mosquito's position relative to the center of the camera's field of view.

- **Motor Control:**
  - If $x_{\text{position}} <$ mid_point, decrease left motor speed and increase right motor speed to turn left.
  - If $x_{\text{position}} >$ mid_point, increase left motor speed and decrease right motor speed to turn right.

```
    if x_position < mid_point:
        speed_pct_a = specific_speed - (((mid_point-x_position) * 2 /
                      image_width) *specific_speed)
        speed_pct_b = specific_speed
    elif x_position > mid_point:
        speed_pct_a = specific_speed
        speed_pct_b = specific_speed - (((x_position - mid_point) * 2 /
                      image_width) *specific_speed)
    else:
        speed_pct_a = specific_speed
        speed_pct_b = specific_speed
```

Listing 5: Code for speed control.

- **Speed Application:** Adjustments to the PWM signals are dynamically applied to control motor speeds.

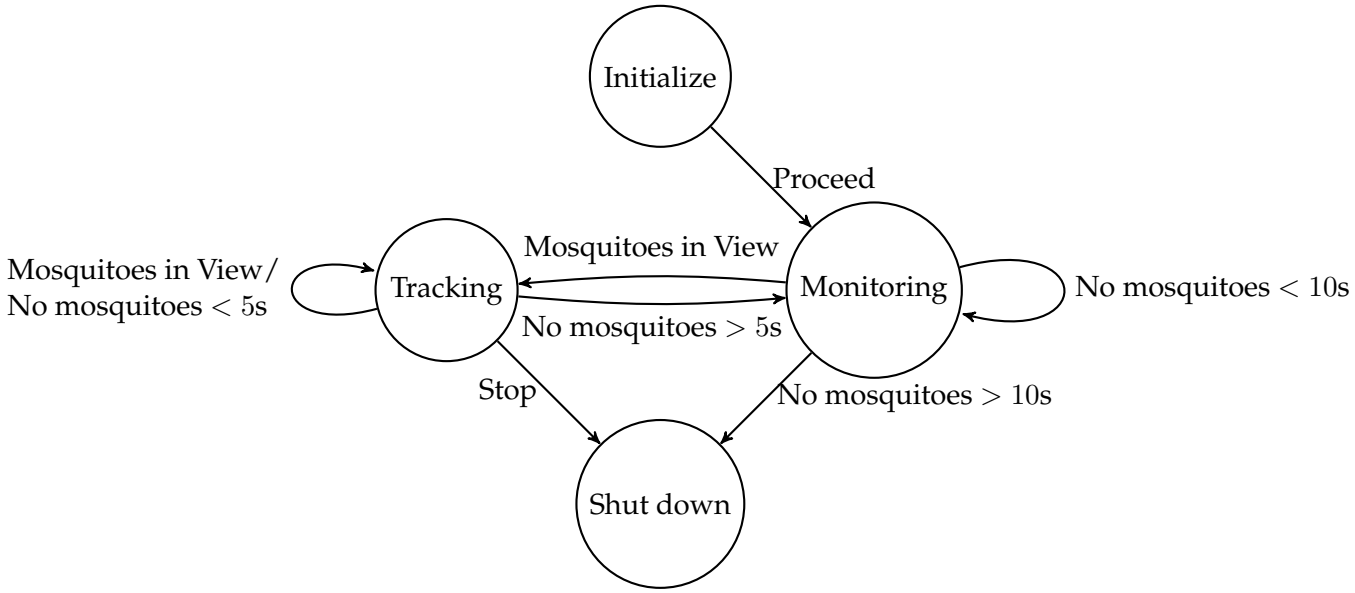
Real-Time Feedback Loop

- **Continuous Monitoring and Adjustment:** The device continuously adjusts based on real-time video and mosquito movement, enhancing tracking accuracy.

This balance between speed and precision ensures effective tracking and response to mosquito movements within the visual field.

2.3.2 Finite State Machine Design

The different states of the machine dealing with different situations can be expressed as a Finite State Machine.



1. **Initialization State**:
    - **Functionality:** Sets up GPIO for motor control, initializes PWM signals, and configures the camera and detection model.
    - **Transitions:** Proceeds to the ”Monitoring” state after setup.
2. **Monitoring State**:
    - **Functionality:** The machine will continuously circle around to captures frames and look for mosquitoes.
    - **Transitions:**
        - Moves to ”Tracking” if a mosquito is detected.
        - Moves to ”Shutdown” if there is no mosquitoes detected for 10s or on errors.
3. **Tracking State**:
    - **Functionality:** One wheel stops, one of the other two wheels reverses the direction of rotation, and the car begins to go straight and adjusts the speed of motors based on the mosquito’s position to center it in the camera’s view.
    - **Transitions:**
        - Remains in this state while the mosquito is in view.
        - Returns to ”Monitoring” if the mosquito leaves the view for more than 5s.

- Moves to "Shutdown" on stop conditions.

4. **Shutdown State**:
    - **Functionality:** Cleans up resources, stops motors, and prepares the system for shutdown.
    - **Transitions:** Terminal state with no further transitions.

2.3.3 PWM Design

Overview:

In the Movement and Rotation Module, Pulse Width Modulation (PWM) is utilized to control the speed and direction of motors based on the detected position of mosquitoes. By varying the duty cycle of the PWM signals sent to each motor, the system dynamically controls the device's movement, achieving precise positioning for optimal tracking.

Design Procedure:

The duty cycle in PWM is controlled by adjusting the duration of the high signal within each pulse relative to the total pulse duration. A higher duty cycle increases the motor speed, enabling quicker turns and faster reaction to mosquito movements.

Design Details:

Calculation of PWM Duty Cycle

The duty cycle for each motor is adjusted based on the deviation of the mosquito from the center of the camera's view. The formula used is:

$$\text{Duty Cycle}(\%) = \left(\frac{\text{Specific Speed} - \text{Adjustment Factor} \times \text{Specific Speed}}{\text{Maximum Speed}} \right) \times 100$$

where:

- **Specific Speed** is the speed required based on the mosquito's movement.
- **Adjustment Factor** is calculated from the positional deviation:

$$\text{Adjustment Factor} = \frac{|x_{\text{mid}} - x_{\text{position}}|}{\frac{1}{2} \times \text{Image Width}}$$

Sending PWM to Motors

Once the duty cycle is determined, it is applied to the motors using:

```
GPIO.PWM(pin, frequency).ChangeDutyCycle(duty_cycle)
```

This command controls the motor speed by adjusting the PWM duty cycle. The precise control of PWM signals to the motors ensures effective tracking of mosquitoes. This capability allows the device to align perfectly with the mosquito's position, optimizing the tracking and actions taken by the system.

2.4 Control and Integrated Module

2.4.1 Deployment of the audio detection to Raspberry Pi

The deployment of the neural network model to a Raspberry Pi involves transferring the pre-trained model and setting up real-time audio processing. The following steps and corresponding code illustrate how this is achieved:

The pre-trained model is transferred to the Raspberry Pi using PyTorch's functionality. The weights are saved in a `.pth` or `.pt` file and loaded on the Raspberry Pi as follows:

```
model_path = 'mosquito_sound_classifier.pth'
model.load_state_dict(torch.load(model_path, map_location=torch.device('cpu'))
    )
model.eval()  # Set the model to evaluation mode
```

Listing 6: Loading the model on Raspberry Pi.

- `torch.load`: Loads the model, ensuring compatibility with the CPU environment of the Raspberry Pi.
- `model.eval()`: Sets the model to evaluation mode, which is necessary for inference as it disables training-specific layers like dropout.

Audio data is processed in real time, and the extracted features are used for classification. The function below handles the real-time prediction:

```
def predict_audio(audio_path):
    feature_vector = load_and_extract_features(audio_path)
    feature_tensor = torch.from_numpy(feature_vector).float().unsqueeze(0)
    with torch.no_grad():
        outputs = model(feature_tensor)
        _, predicted = torch.max(outputs.data, 1)
    return predicted
```

Listing 7: Python function for real-time audio prediction.

- `load_and_extract_features`: Extracts MFCC features from the audio file.
- `torch.no_grad()`: A context manager that disables gradient computation to speed up predictions and reduce memory usage.
- `torch.max(outputs.data, 1)`: Determines the predicted class by identifying the class with the highest probability.

The following equation models the real-time feature extraction and prediction pipeline, which is crucial for deployment on the Raspberry Pi.

$$\hat{y} = f_{\text{CNN}}(f_{\text{MFCC}}(\text{audio\_signal}))$$

Where:

- f_{MFCC} is the function extracting MFCC features from the audio signal.

- f_{CNN} is the CNN model function for prediction.
- audio_signal is the input audio data.
- $\hat{y}$ is the predicted output.

These steps encapsulate the process of deploying a neural network model on a Raspberry Pi, emphasizing efficient model loading and real-time processing for mosquito detection.

2.4.2 Deployment object detection model to Raspberry Pi

Lightweight and Cloud-based Computing

The use of Roboflow enhances this deployment by simplifying the model integration and management process, making sophisticated machine learning accessible on resource-constrained devices.

One of the ingenious aspects of this deployment is the utilization of a cloud inference server, which allows for real-time processing without the latency and bandwidth costs associated with local computing. This is particularly beneficial for applications that require immediate response times.

```
rf = roboflow.Roboflow(api_key='yAN6VfQfUQChP8X8xIaH')
project = rf.workspace().project("dstardust")
local_inference_server_address = "http://localhost:9001/"
version_number = 2

local_model = project.version(
    version_number=version_number,
    local=local_inference_server_address
).model
```

Listing 8: The deployment on the server of the Roboflow model

Adaptive Frame Processing

The system dynamically calculates and displays the frames per second (FPS), which not only provides a real-time performance metric but also allows for adaptive adjustments. For instance, the processing detail or frequency could be scaled based on the current FPS, thus maintaining a balance between speed and accuracy, is essential for varied real-world scenarios.

```
counter = 0
start_time = time.time()
capture = cv2.VideoCapture(0)
ret, frame = capture.read()
fps = capture.get(cv2.CAP_PROP_FPS)
while ret:
    counter += 1
    if (time.time() - start_time) != 0:
        cv2.putText(frame, "FPS {0}".format(float('%.1f' % (counter / (time.
   time() - start_time)))), (30, 50),
```

```
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255),
                    2)
        ret, frame = capture.read()
        print("FPS: ", counter / (time.time() - start_time))
        counter = 0
        start_time = time.time()
```

Listing 9: Video Capture and Frame Processing

Customizable Detection Parameters

```
srcimg = local_model.predict(image_path=frame, overlap=50,
                             confidence=60)
```

The deployment design includes customizable parameters for the detection process, such as overlap' and 'confidence' levels. This flexibility allows users to fine-tune the model based on the specific requirements of their deployment environment, which could vary significantly in terms of object size, lighting conditions, and required detection sensitivity.

Real-time Interactivity and Feedback Loop

The system is designed to provide immediate visual feedback through an annotated video stream, which is essential for user interaction and for tasks requiring instant decision-making.

```
for prediction in srcimg.json()['predictions']:
    x, y, w, h = int(prediction['x']), int(prediction['y']), int(prediction['
    width']), int(prediction['height'])
    cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
    cv2.putText(frame, f"{'mosquitoes'} {prediction['confidence']:.2f}",
                (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 1)
cv2.imshow("video", frame)
```

Listing 10: Model Prediction and Annotation

The 'predict' method of the local model is called with each frame, demonstrating how real-time detection is implemented. The system annotates detected objects in the video stream and displays these annotations in real time, highlighting the application's interactivity and immediacy.

2.4.3 Connection and Control of Motors

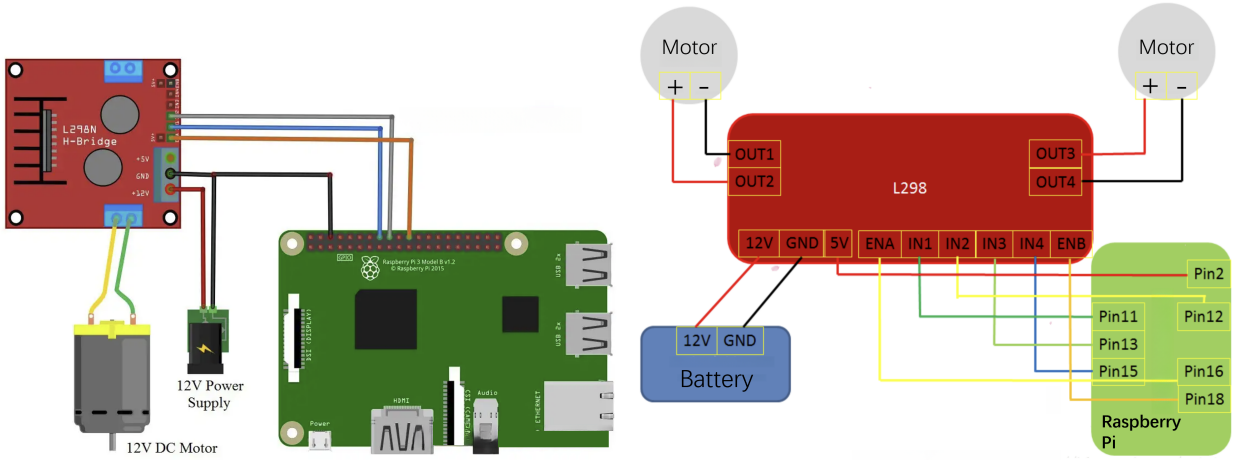
The L298N motor driver is commonly used for controlling motors in robotics due to its ability to drive two motors simultaneously and support motor directions with a high current output. In this design, a Raspberry Pi is used to control the motors through the L298N, enabling precise manipulation of motor speeds and directions based on real-time data processing from a camera.

Circuit Configuration

The GPIO (General Purpose Input/Output) pins of the Raspberry Pi are utilized to interface with the L298N motor driver. The motor driver's input pins (INT1 to INT8) are connected to specified GPIO pins on the Raspberry Pi, which allows for controlling up to four motors (two motors with bidirectional control).

```
GPIO.setmode(GPIO.BOARD)
INT1, INT2, INT3, INT4, INT5, INT6, INT7, INT8 = 11, 12, 13, 15,
                                                 29, 31, 32, 35
ENA, ENB, ENC = 16, 18, 33
```

Listing 11: Codes for Pin Connections



(a) Schematic diagram of circuit connection

(b) L298N and Raspberry PI connection diagram

Figure 23: Two kinds of circuit diagrams of L298N

L298N Motor Driver and GPIO Pins

- `High Current Capability`: The L298N motor driver can handle up to 2A per channel, which is sufficient for a wide range of DC motors used in robotics. This makes it an excellent choice for projects that require driving motors with significant power demands.
- `Dual-Channel Support`: The L298N can control two motors independently with one single IC. This feature is particularly useful for driving both wheels of a robotic vehicle, allowing for individual control of each wheel which is necessary for maneuvers such as turning and pivoting.
- `Built-in Protective Features`: The L298N includes internal clamp diodes that protect the driver from inductive voltage spikes, which are common when driving motors. This built-in protection helps to prolong the life of both the motor and the driver.
- `Versatility and Ease of Use`: It is compatible with a wide range of microcontrollers and supports standard logic levels for inputs, making it widely compatible and easy to integrate into various projects.

- `Customization`: Using GPIO pins, we can customize the behavior of your robotics system extensively. Each pin's behavior can be individually defined, allowing complex control logic that can be adapted to a wide range of scenarios and tasks.

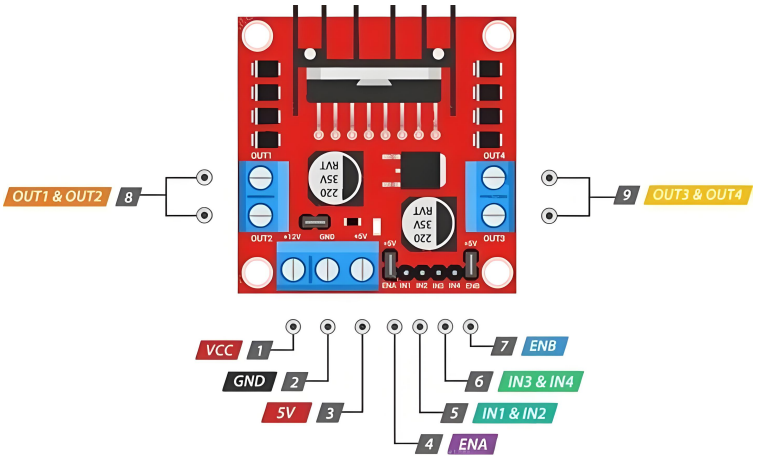


Figure 24: L298N and its pins.

Variable and Constant Speed Control

PWM signals

- `Speed adjustment`: PWM allows for precise control over the speed of motors. By adjusting the duty cycle of the PWM signal (the proportion of time the signal is high versus the total time of the signal), you can vary the average voltage being applied to the motor. This control is essential for tasks that require different speeds.
- `Efficiency`: Using PWM for motor control is energy efficient. Instead of applying constant voltage, PWM sends short bursts of energy. This can reduce the energy loss in the form of heat, especially in scenarios where the motor does not need to run at full power continuously.
- `Simplicity and Cost-effectiveness`: Implementing PWM with digital controllers like the Raspberry Pi is straightforward and does not require complex circuitry or expensive components, which can be crucial for keeping project costs and complexity manageable.

Jumper Caps

- `Direct Connection`: If the application such as the fan does not require variable speed control for the motors, I simplify the setup by using jumper caps. Placing a jumper cap on the ENA and ENB pins will supply them with a constant high signal, effectively running the motors at full speed whenever they are powered.
- `Simplicity`: This method is simpler as it does not require programming the Raspberry Pi to handle PWM signals.

- `Reliability`: Fewer connections and simpler code reduce the potential for errors and troubleshooting.
- `Power Efficiency`: Running motors at full speed without modulation can be suitable for applications where the load and speed requirements are constant and predictable.

2.5 Mechanical Structure

2.5.1 Design description and drawings

The mechanical structure has been greatly improved since the design document. We need it to be concise and easy to manufacture while implementing functions such as self-rotation, straight movement, and turning. As shown in Figure 25, we use acrylic layers and copper pillars to build the body of the machine and three omni wheels to control its movement, based on the principle of force balance. For self-rotation, all three wheels rotate at the same speed and in the same direction. For straight movement, one wheel's speed is set to zero, while the other two wheels rotate at the same speed but in opposite directions.

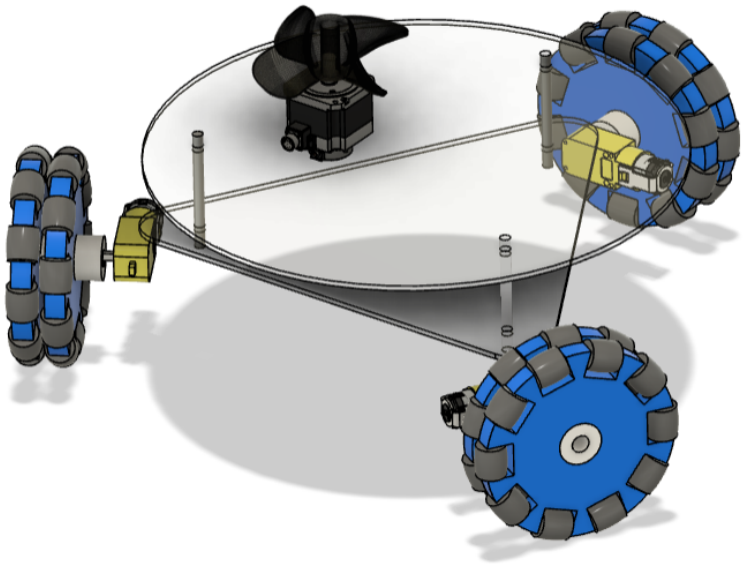


Figure 25: CAD model.

The current design significantly reduces mechanical and electrical complexity. Once the mosquito is detected to exist, the machine will rotate by itself to provide the camera with a 360-degree view; once the mosquito is localized, the machine will move towards it by issuing different commands to the three wheels.

Figures 26 to 28 depict CAD drawings of the mechanical system. The system has few components. The top and bottom layers of the cart are made by laser-cutting acrylic boards. All electrical components will be taped to the acrylic boards. Three omni wheels have been purchased, and Figure 4 provides its drawing.

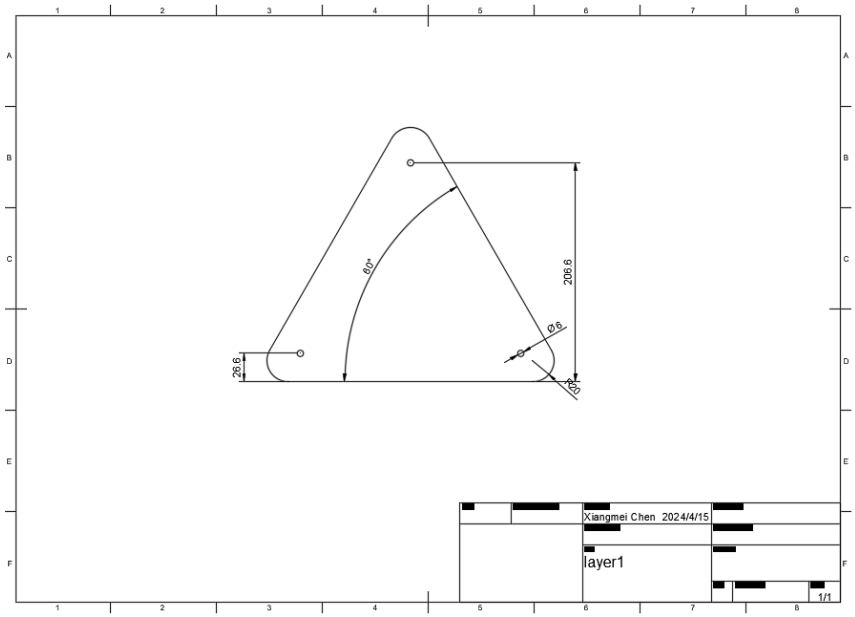


Figure 26: Drawing for the bottom layer.

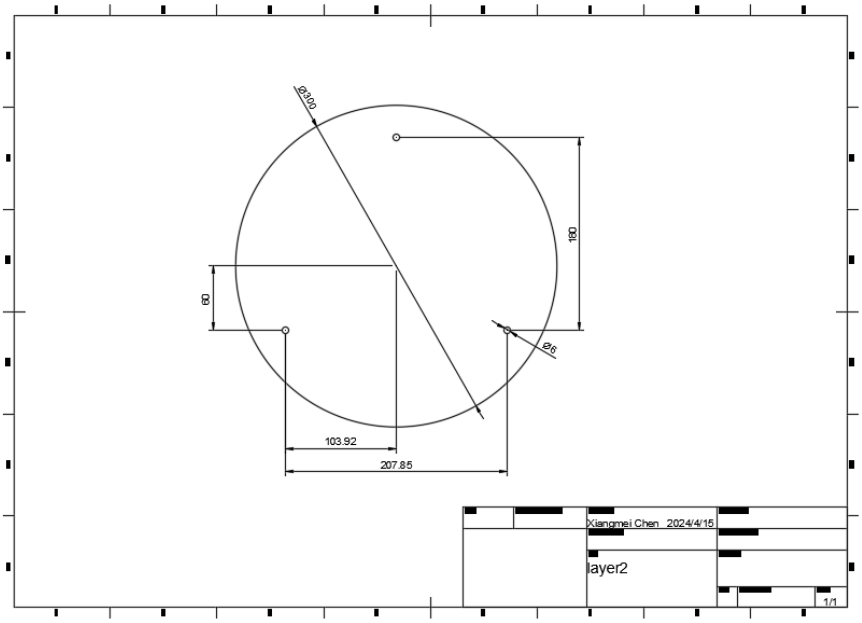


Figure 27: Drawing for the top layer.

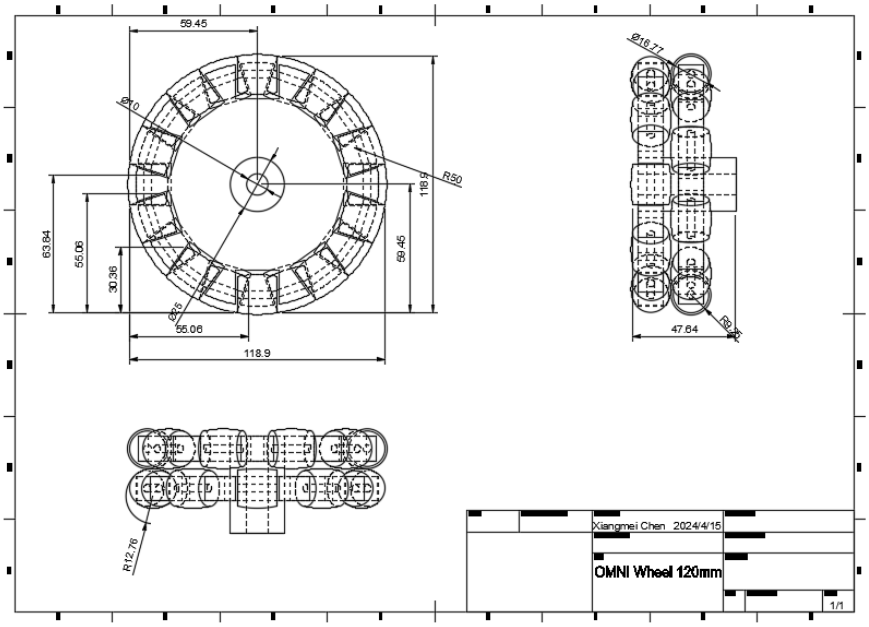


Figure 28: Drawing for the omni wheel.

2.5.2 Simulation results

In summary, simulation results indicate that our mechanical design is theoretically safe. The stress distribution provides suggestions on where to place weight during physical testing.

The material for the top and bottom layers is 3 mm acrylic. For the bottom layer, we assume a downward force of 5 N at each hole and a moment of 1 N*mm. The maximum stress observed is 0.094 MPa, which is below the yield stress of acrylic (40 MPa). It is notable that the maximum stress occurs at three edges; therefore, if weight is to be added to the board, it is advisable to place it in the dark blue area.

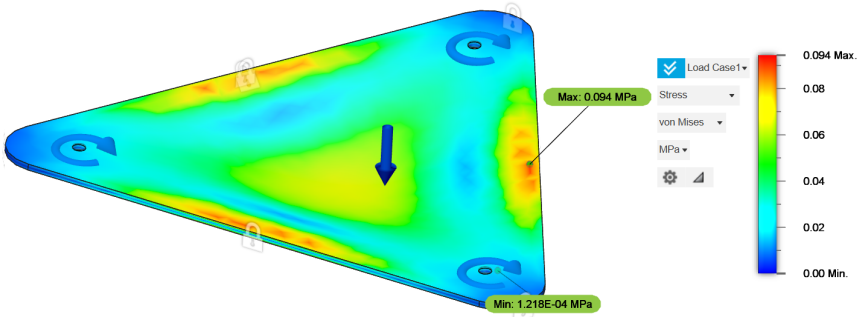


Figure 29: Simulation results for the bottom layer.

Regarding the top layer, we assume a downward force of 2.5 N and a moment of 1 N*mm. While the overall stress is higher than that of the bottom layer, it still does not exceed the

yield stress. Interestingly, the different geometry provides insights into where weight should be placed. Unlike the triangular bottom layer, the circular top layer exhibits a distinct stress distribution.

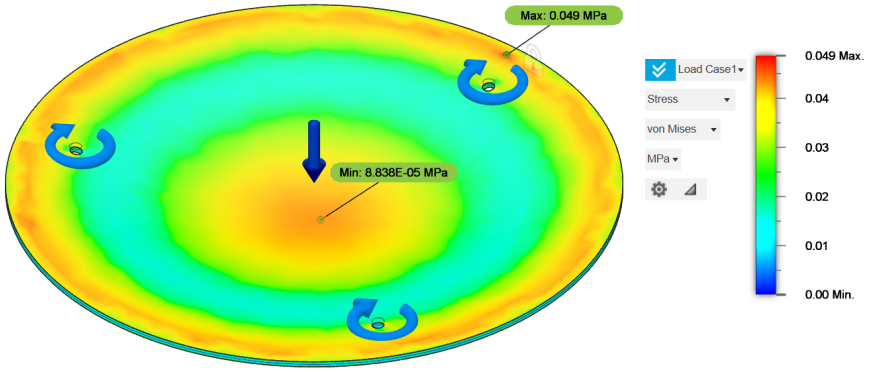


Figure 30: Simulation results for the top layer.

I also conducted a simulation analysis for the shaft, as shown in Figure 13. The material used is stainless steel, with a yield stress of 250 MPa. This analysis is for the scenario involving a quick turn, resulting in a torque of 5 N*mm being exerted on it. The maximum stress observed is 0.234 MPa, indicating that it is well within the safe range.

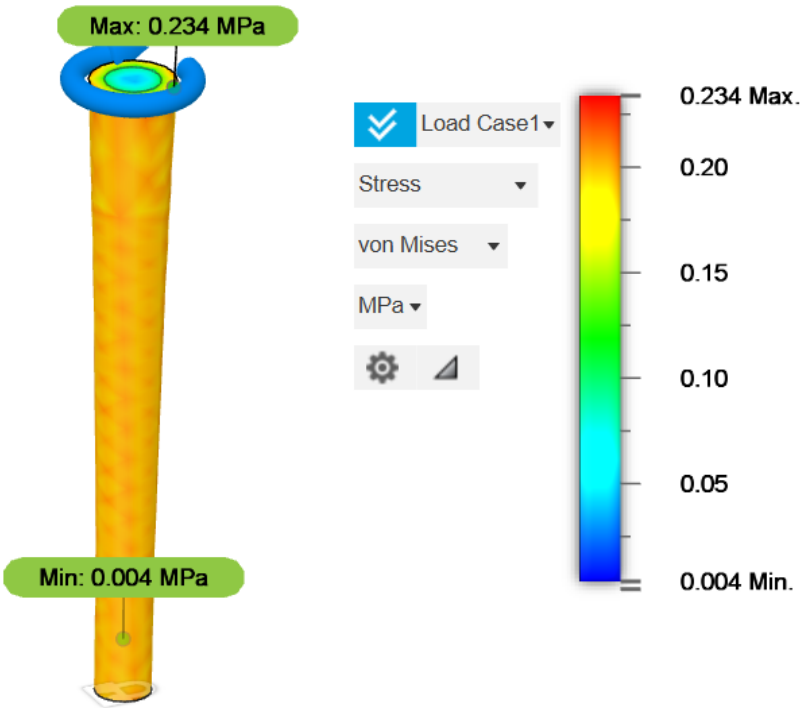


Figure 31: Simulation results for shaft.

2.5.3 Design alternatives

As seen in Figure 32, we manufactured and built the first physical model. During testing, we found that electrical components were crowded at the bottom layer, which was very messy. Another issue was that the bottom layer was large compared to those three wheels. Though the acrylic board can withstand the weight, it was deformed, resulting in the three wheels not holding a vertical angle with the ground. There were also issues with the way the motor was fixed because screws and nuts would loosen during moving.

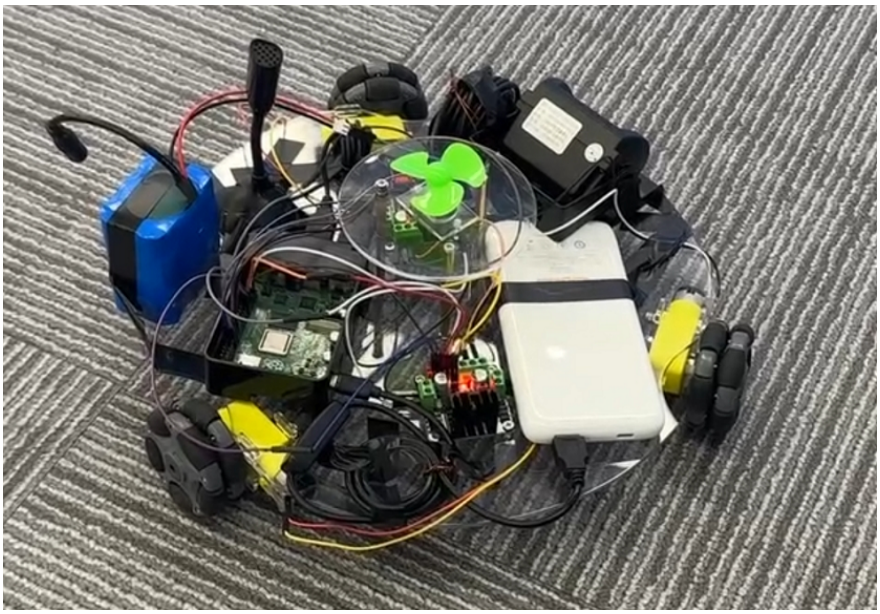


Figure 32: Physical model during the first test.

To address these issues, we improved the design. We made the machine in three layers while reducing the size of the bottom layer. The drawings for the three acrylic layers are shown in Figures 33 to 35. We added a support and ball bearing underneath the bottom layer to overcome the deformation, as seen in Figure 36. The connection between the motor and acrylic board is made using 502 glue. The overall appearance of the machine is shown in Figure 36.

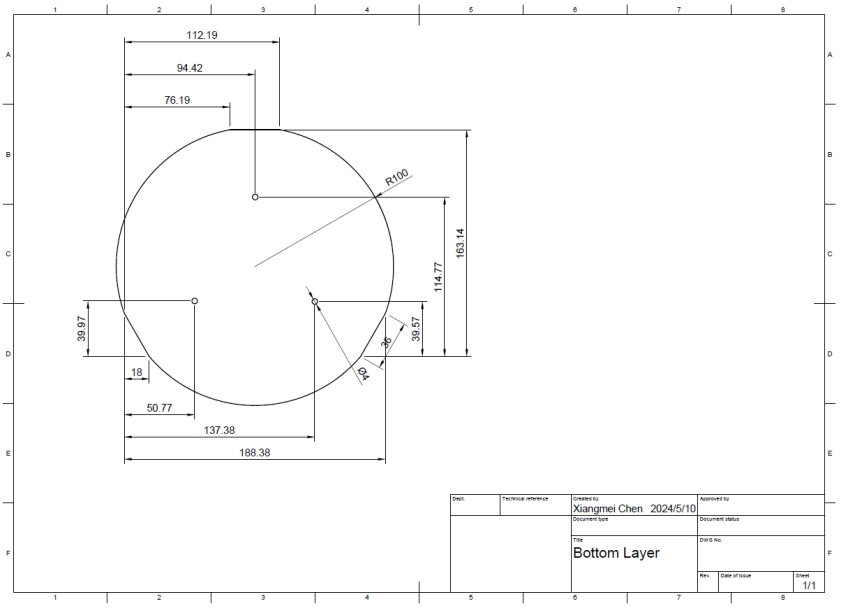


Figure 33: Drawing for the bottom layer.

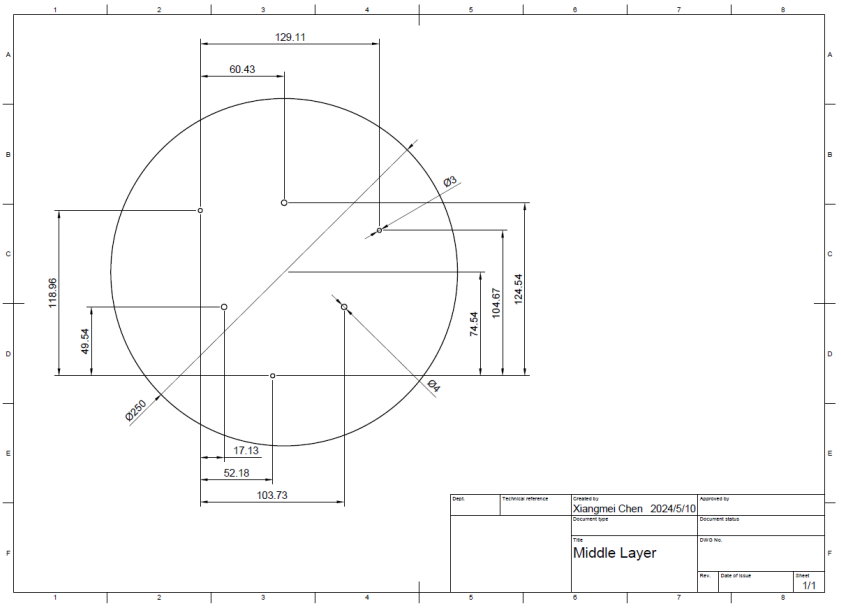


Figure 34: Drawing for the middle layer.

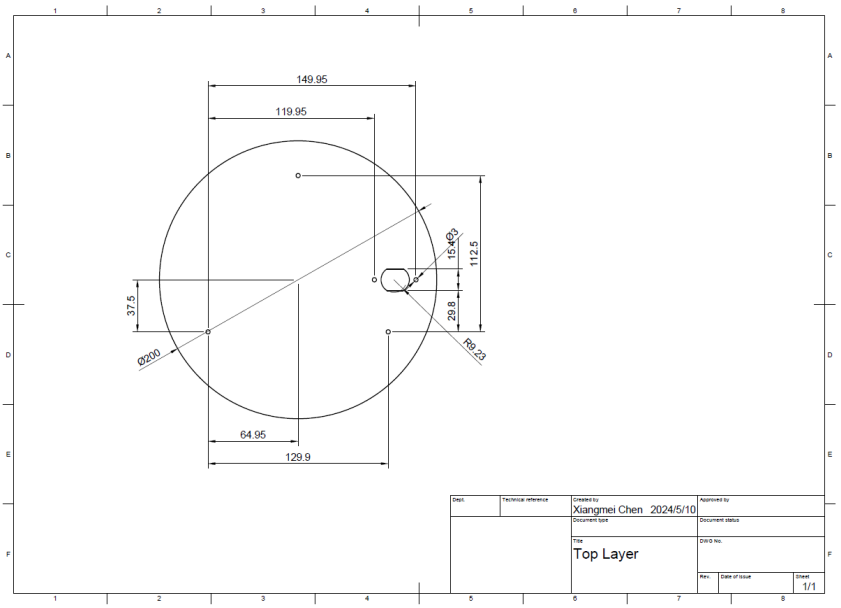


Figure 35: Drawing for the top layer.

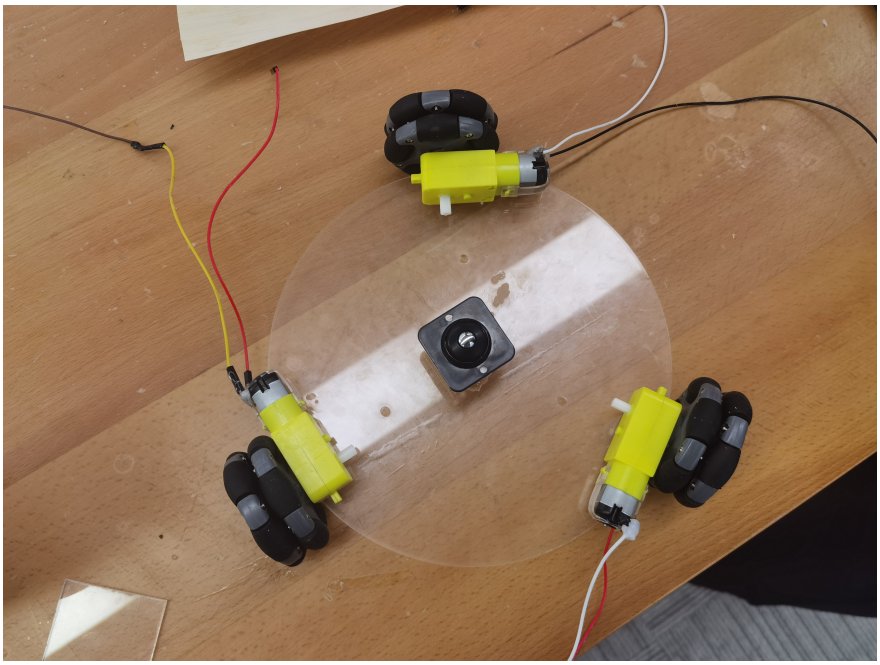


Figure 36: Bottom layer with support and ball bearing added.

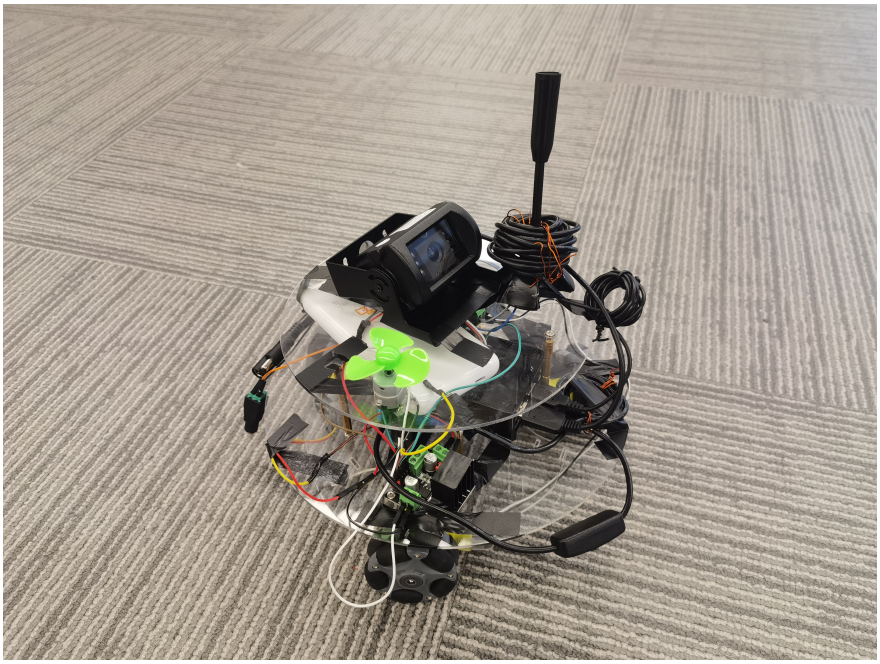


Figure 37: Appearance of the design alternative.

3 Cost and Schedule

3.1 Cost Analysis

3.1.1 Cost of labor

We take the average salary of UIUC graduates as our hourly wage, which is $20 per hour. Assume our team works three hours a day and five days a week, and there are 13 weeks to work. So, the total labor is $\$20 \times 3 \times 5 \times 13 \times 2.5 \times 4 = \$39000$.

3.1.2 Cost of parts

Part #	Description	Manufacturer	Quantity	Cost
1	Raspberry Pi 4B plus camera	Raspberry Pi Foundation	1	589 RMB
2	Arduino Development Board ATMEGA16U2	ArduinoLLC	1	80 RMB
3	Microphone for Raspberry Pi 4B	ArduinoLLC	1	9 RMB
4	Small fan	Telesky	1	7 RMB
5	Single chip small car	Beikemu	1	30 RMB
6	Bogie	Boxi	1	90 RMB
7	L298N	STMicroelectronics	1	6 RMB
8	Delipow 18650 lithium battery pack	Delipow	1	46 RMB
9	1080P Camera	Linboshi	1	196 RMB
10	printed PCB & components	JLC Technology Group	1	57 RMB
11	ball bearing	Tao Factory	1	3 RMB
Total				1113 RMB

Table 1: Cost Table.

3.1.3 Sum of Costs

The grand total costs is approximately $40000.

3.2 Schedule

Week	Xiangmei Chen	Peiqi Cai	Yang Dai	Lumeng Xu
3/25	Finish CAD model version 1. Finish purchases. Laser cut major parts.	Set up the running environment of Yolov8 on raspberry pi, and deploy the Yolov8 model.	Improve the YOLOv8 algorithm and start work on PCB design.	Research and identify a suitable dataset for mosquito wingbeat sounds, download and organize the dataset for further processing.
4/1	Assemble things together and test stability.	Ensure that the microphone is properly connected to the Raspberry Pi and set up the audio processing software to capture sound data.	Research on the codes and algorithms to perform data augmentation on the visual and audio datasets.	Begin coding the neural network architecture for sound classification and implement preliminary training using a portion of the dataset.
4/8	Test motor to see if the cart can move, revolute, and move up and down. Refine CAD if needed. Add features by 3D printing. Work on individual progress report.	Develop or adapt existing software to process the audio input from the microphone and detect mosquito sounds.	Collect and organize actual image and audio datasets and perform data augmentation on them.	Continue training the neural network with the full dataset. Validate the neural network's performance using a separate validation dataset.
4/15	Cooperate with detection subsystem and localization subsystem.	Create a program that can generate PWM signals to control the speed and direction of the motors connected to the servos.	Label the new datasets with a more complex background and train the model on them.	Connect the microphone to the Raspberry Pi. Set up the audio processing software to capture sound data accurately and test microphone functionality and ensure high-quality audio input.
4/22	Test the entire system. Add features by 3D printing or laser cutting if needed.	Build a PWM generation program on Raspberry Pi, responsible for sending PWM signals to the servos based on the mosquito's position in the camera's field of view.	Research on the codes and algorithms to locate the target based on the images feedback from Raspberry Pi.	Conduct tests with new, unseen sounds to verify the neural network's accuracy and begin integrating the neural network into the real-world environment for initial testing.
4/29	Make sure that the entire system is robust.	Calibrate the system for mosquito tracking, ensure that the camera and motors work in harmony.	Build the interfaces for the Roboflow 3.0 to control the rotation information contained in PWM, based on the target position.	Collaborate with the team on the design of the attacking subsystem and begin manufacturing or prototyping components for the subsystem.
5/6	Work on final report draft.	Develop or implement an algorithm that uses the mosquito's position data to control the motors, keeping the mosquito centered in the camera's view.	Test the performance of the localization subsystem in normal cases.	Finalize the design of the attacking subsystem, integrate the neural network into the system for real-time mosquito detection and response.
5/13	Work on demo and revision of individual report.	Test the system components individually.	Collaborate with other teammates with the testing on the components.	Test each component of the system individually to ensure proper functionality and identify and address any issues that arise during testing.
5/20	Prepare presentation and final report.	Ensure that the Raspberry Pi and all connected components are properly powered, and optimize power usage for efficient operation, especially if the system is battery-operated.	Ensure the performance of the device and make refinement on logic bugs.	Ensure the code can be run correctly and the connection of all parts are correct, check for overall integration.

Table 2: Schedule of the project.

4 Requirements and Verification

4.1 Audio Detection Module

4.1.1 Model Training Accuracy

Requirements: The machine learning model must correctly identify mosquito sounds with an accuracy of at least 85%.

Verification: Prepare some test datasets consisting of non-mosquito sounds. Executing the model and recording outcomes to calculate accuracy (A):

$$A = \frac{Correct}{Total};$$

where Correct is the number of labels where the outcome equals to the predicted, and Total is the number of all outcome labels. We need to ensure $A \geq 85\%$.

Results: We chose three datasets containing different pronunciations of English words "cat", "dog" and "bird" and imported them into our model to test, and all the three accuracy are 90% or so, which is above our requirement.

Figure 38: Testing Accuracy Result of Audio Detection Module.

4.1.2 Audio Processing and Detection Latency

Requirements: The total time from mosquito sound detection by the microphone to the identification of the sound by the software should not exceed 25 seconds. This ensures

timely activation of the subsequent subsystems for effective mosquito targeting and eradication.

Verification:

1. Record the timestamp t_0 when mosquito sound is detected by the microphone.
2. Record the timestamp t_1 when the audio capture finishes.
3. Record the timestamp t_2 when mosquito detection is confirmed.
4. Calculate $T_{\text{total}} = t_2 - t_0$ for each trial and ensure $T_{\text{total}} \leq 10$ seconds.

We need:

$$\frac{1}{n}\sum_{i=1}^{n} T_{\text{total},i} \leq 10 \text{ seconds}$$

where n is the number of trials.

Results: We tested the code with different audio files, containing mosquito sounds and non-mosquito sounds, for 15 times, and the average timestamp duration is 22.4s.

4.2 Computer Vision Module

4.2.1 Model Metrics

Requirements: The Roboflow 3.0 model must correctly identify at least 90% of mosquitoes in the validation dataset with a precision of 85% or higher and a recall of 85% or higher.

Verifications: Process the captured images with the Roboflow 3.0 model. Document the number of mosquitoes identified, false positives, and false negatives. Calculate the precision and recall with different confidence based on the documented data and the formula

$$Precision = \frac{TP}{TP + FP}, \; and \; Recall = \frac{TP}{TP + FN}$$

Also, use other metrics such as Mean Average Precision (mAP) to determine the performance and the value of confidence, where

$$mAP = \frac{1}{n}\sum_{k=1}^{n} AP_k$$

Results: The training results shows sufficient performance of the model, where precision and recall reach 88.2% and 87.4% for the best checkpoint. The map also reaches 90.5% in this version.

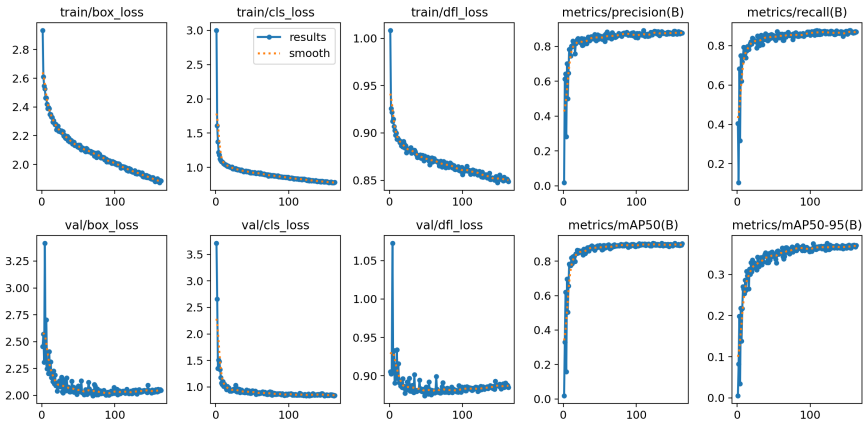


Figure 39: Training Metrics Result of Roboflow 3.0 Object Detection model.

By adjusting the Confidence Threshold given by Roboflow, the confidence we would like to use will be between 0.2-0.3, where both precision and recall have great performance.

Figure 40: Precision and recall of the model given confidence threshold of 0.2.

Figure 41: Precision and recall of the model given confidence threshold of 0.3.

4.2.2 FPS and Inference Latency

Requirements: The inference speed should be fast enough so that the control unit is able to receive 3-4 frames per second.

Verifications: Build the SSH tunnel connection between Raspberry Pi and inference server and transfer the image to the server to infer. Calculate the frames per second (FPS) by:

$$FPS = \frac{1}{end\_time - start\_time}$$

where $end\_time - start\_time$ is the time interval for each time the control unit reads a frame. Calculate the average FPS by averaging the processing time across 15-20 frames and verify if the average FPS meets the real-time processing constraint.

$$FPS_{avg} \geq \frac{1}{T_{max}}$$

where FPS_avg is the average frame rate and T_{max} is the maximum constrained inference time that is sustainable for attack subsystem.

Results: After transferring the image to the inference server, received the inferred result and calculate the inference time, we get the FPS result for the 18 samples as follows.

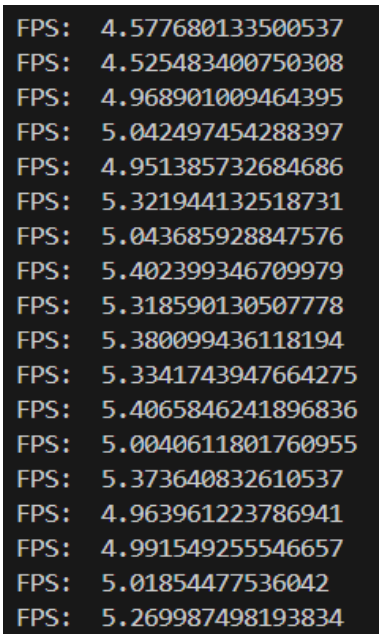


Figure 42: FPS of the image captured for 18 samples.

where we get the average FPS $FPS_{avg} = \sum FPS_i \approx 5.11\ frames\ per\ second$, which is above our requirements.

4.3 Movement and Rotation Module

4.3.1 Camera Activation Time Delay

Requirements: The camera system must activate within 10 seconds of the audio trigger.

Verification: The camera system must activate and begin scanning within 10 seconds of the audio detection trigger,

$$T_{\text{act}} \leq T_{\text{max}} = 10 \text{ s}$$

with T_{act} being the activation time and T_{max} the maximum allowable time.

Results: After testing our model for ten times, we measured the time duration between the detection of mosquitoes by audio detection part and the moment when the camera view appeared on the screen of computer, and calculated the average time duration by the formula

$$T_{\text{avg}} = \frac{1}{n} \sum_{i=1}^{n} T_{\text{act}} = 4.2\text{s}$$

where n is 10, and the result shows that our camera time delay meets the requirement.

4.3.2 Servo Motor Response

Requirements: Servo motors must respond within 0.5 seconds of a control signal.

Verification: Servo motors must respond to control signals within 0.5 seconds, ensuring immediate adjustment of camera positioning,

$$T_{\text{res}} \leq T_{\text{res,max}} = 0.5\text{s}$$

where T_{res} is the servo motor response time .

Results: After testing our model for ten times, we measured the time duration between the signal received moment at the Raspberry Pi from the state at the camera and the moment when the motor started to act accordingly, and calculated the average time duration by the formula

$$T_{\text{avg}} = \frac{1}{n}\sum_{i=1}^{n} T_{\text{res}} = 0.48\text{s}$$

where n is 10, and the result shows that our motor response time meets the requirement.

4.3.3 Car Rotation Sensitivity

Requirements: The car must complete a 360-degree rotation with all the equipment on it at different velocities.

Verification: The car must be capable of a full 360-degree rotation to scan the entire environment of different levels of determined velocities,

$$\omega = \frac{\Delta\theta}{\Delta t} = \frac{360^\circ}{\Delta t}$$

with ω being the rotational speed, $\Delta\theta$ the angle of rotation, and Δt the time taken. And we have different ω.

Results:

4.4 Control and Integrated Module

4.4.1 Battery for Stable Power Supply and Switch

Requirements: The battery must provide a stable 12V power supply to the control unit and other components of the machine. And the PCB must convert the 12V DC from the battery into 5V for the power supply of Raspberry Pi.

Verification: Open the switch of battery and make the machine to operate normally without other electricity sources for nearly 2 minutes to see whether the machine can work without bugs.

Results: The machine can work well for more than 2 minutes.

4.4.2 Motor Control for Movement and Capture

Requirements: The motor control system, managed by the Raspberry Pi, must send signals to the GPIO-connected motors to enable precise movement and positioning, including the motion of wheels and activation of the fan for mosquito capture.

Verification: To verify the precision of the motor control system, the response time (Tr) was measured to be 0.2 seconds, calculated using the formula
$Tr =$ Time from signal sent to motor start moving. The precision (P) of the motor's positioning was calculated as $P = \frac{\text{Expected Position} - \text{Actual Position}}{\text{Expected Position}}$ and resulted in 98.5%. The average precision (Pavg) over multiple tests was 98.2%, calculated using $Pavg = \frac{\text{Sum of all Precision values}}{\text{Number of tests}}$. These results confirm that the motor control system meets the requirement for precise movement and positioning.

Results:

5 Conclusion

5.1 Accomplishments

Our machine successfully detects, localizes, and captures mosquitoes within a 2-meter range. It utilizes a microphone and camera, guided by machine learning models, to detect and track mosquitoes. The machine features a mechanical structure with 360-degree rotation capability and a fan unit for effective mosquito capture. These capabilities are made possible by a power and control subsystem with a useful PCB that provides stable power and coordinates operations through a Raspberry Pi control unit. Overall, our design almost meets our requirements and can effectively deal with mosquitoes.

5.2 Uncertainties

5.2.1 Uncertainty in Localization Subsystem

1. Camera Resolution and Field of View (FoV): The high-resolution USB camera's performance may be affected by lighting conditions and lens cleanliness. For instance, if there are reflections in the camera's FoV or dust on the lens, it could reduce localization accuracy.

Assuming optimal conditions, pixel density might be 300 PPI (Pixel Per Inch). Under adverse conditions (e.g., lens dirt or poor lighting), effective pixel density could drop to 150 PPI, potentially doubling the error margin in localization. The mathematical expression of the effective pixel density is:

$$P_{\text{effective}} = P_{\text{optimal}} - \Delta P$$

Where $P_{optimal}$ is the optimal pixel density and ΔP is the decrease in pixel density due to adverse conditions.

5.3 Future Work / Alternatives

For future improvements, we can examine the subsystems one by one.

For the detection subsystem, future work should focus on real-time detection. Sounds made by mosquitoes are actually very small compared to environmental noise. Additionally, the frequency of the sound varies with temperature and humidity, making it a very complex issue.

For the localization and attack subsystems, future work should aim at reducing time lag and improving resolution so that our machine can accurately differentiate mosquitoes from other objects. When it comes to sucking in mosquitoes using a fan, we must find an easier way to clean the container after mosquitoes are captured.

As for the power and control subsystem, it can be enhanced by using a lighter-weight battery.

5.4 Ethical Considerations

A qualified project must adhere to the ethics codes outlined in IEEE Policies and ACM [4], [5]. As stipulated in the team contract, the four of us will collaborate to ensure mutual respect and fairness, committing to upholding these codes collectively and making requisite risk mitigation plans accordingly.

Our project aims to effectively manage mosquitoes, contributing to the creation of a healthier public environment. The presence of mosquitoes has been associated with the spread of diseases and unfortunate fatalities worldwide. Therefore, our goal is to mitigate these challenges for the well-being of communities globally.

Our project can be divided into three main components: mosquito detection, using a camera for mosquito localization, and mosquito elimination. Regarding the detection phase, we believe there are no ethical concerns. However, the use of a camera for positioning raises privacy issues, as it may inadvertently capture irrelevant people and items. When it comes to mosquito elimination, we acknowledge the ethical consideration of taking a life. We respect all forms of life, and our approach ensures mosquitoes are eliminated in a humane and conventional manner. Importantly, none of our team members endorse or derive pleasure from any cruelty towards mosquitoes.

To address these concerns, we propose some requisite risk mitigation plans. We provide advance notifications in the experimental area to inform individuals about the monitoring process and ensure privacy. Instead of using real mosquitoes, we use audio files downloaded online to test our detection subsystem, and pictures of mosquitoes to test our localization subsystem. Additionally, we employ alternative materials such as napkin fragments to assess the performance of our attack subsystem. By avoiding the use of real mosquitoes, we aim to eliminate concerns about cruelty and potential harm to people during testing.

References

[1] WHO. "Mosquitoes." (2021), [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/mosquitoes (visited on 03/27/2024).

[2] D. Kim, T. J. DeBriere, S. Cherukumalli, G. S. White, and N. D. Burkett-Cadena, "Infrared light sensors permit rapid recording of wingbeat frequency and bioacoustic species identification of mosquitoes," *Scientific Reports*, vol. 11, no. 1, 2021. DOI: https://doi.org/10.1038/s41598-021-89644-z.

[3] J. Gallagher. "Announcing roboflow train 3.0." (2023), [Online]. Available: https://blog.roboflow.com/roboflow-train-3-0/ (visited on 05/10/2024).

[4] IEEE. "IEEE Code of Ethics." (2016), [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html (visited on 03/05/2024).

[5] ACM. "ACM Code of Ethics and Professional Conduct." (2018), [Online]. Available: https://www.acm.org/code-of-ethics (visited on 03/05/2024).