# ECE-4740/5740 Final Project

## Pac-Man Image Detecting Obstacle Avoidance Robot

Abdul Wasay, Antonio Ristevski, Bao Nguyen
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
E-mails: wasay@oakland.edu, antonioristevsk@oakland.edu, bqnguyen@oakland.edu

***Abstract*—This project presents an image recognition and obstacle avoidance system using an ESP32-CAM and Arduino Uno. The ESP32-CAM performs real-time image analysis to detect a Pac-Man ghost using a custom decision tree classifier. Upon recognizing it, the ESP32-CAM transmits a signal to the Arduino Uno. Once triggered, the Arduino initiates evasive maneuvers, moving away from the source of detection. To safely navigate the environment, the Arduino uses infrared sensors to detect nearby obstacles. Obstacle avoidance is handled using a fuzzy logic control system, which enables smooth and adaptive motion in response to sensor input. This fusion of vision-based detection with reactive control logic demonstrates a compact and effective solution for autonomous decision-making and real-world navigation.**

**Keywords – Infrared (IR), ESP Camera (ESP-32 Camera AI Thinker)**

## I. Introduction

This project presents the development of an autonomous mobile robot that integrates computer vision, fuzzy logic-based obstacle avoidance, and real-time motor control. An ESP32-CAM module serves as the vision system, performing onboard image classification using a custom decision tree trained on a dataset of approximately 2,000 labeled images. To improve image quality in variable lighting conditions, a high-intensity white LED was mounted next to the ESP32-CAM to provide consistent illumination. When a target object is recognized, the ESP32-CAM sends a detection signal to an Arduino Uno, which is responsible for navigation and motor control. The Arduino responds by initiating a "runaway" maneuver, driving the robot away from the source of detection. Obstacle detection is handled using three Sharp GP2Y0A21YK0F IR sensors positioned on the left, center, and right sides of the robot. Sensor data is processed through a fuzzy logic control system, enabling smooth navigation and real-time obstacle avoidance. Two H-bridge motor drivers control the bidirectional operation of the DC motors, allowing forward, reverse, and turning maneuvers. The entire system is powered with a lithium-ion battery pack, supplying power for the microcontrollers, sensors, and motors.

## II. Methodology

This project consisted of 3 main parts: the Arduino, the ESP32-Camera, and the drive motors. All of these were integrated onto a plastic vehicle chassis that was commercially available. As mentioned earlier, the IR Sensors were positioned at the front of the vehicle while the camera was at the rear. With the DC motors on both sides.

### *A. Physical Implementation*

The robot chassis is a commercially available model purchased from Amazon. It includes 2 DC motors, a central swivel wheel, and a plastic body to mount components onto. Aside from the microcontrollers, the other physical implementations are a battery mount, a 3W LED, and H-bridges for driving motors. The robot car chassis is shown in Fig. 1.



Fig. 1. Robot car chassis utilized in the project.

The batteries utilized are 2 18650 Lithium-ion batteries, each at 3.7V 2,500mAh. The batteries are connected in series, resulting in an overall battery outputting 7.4V 2,500 mAh. This configuration was chosen because the microcontrollers operate best with 5V input, meanwhile battery consumption was not a concern with the project. Voltage regulation is not required here because the components work within their range. The Arduino can accept an input voltage between 6V-20V, and it provides a regulated 5V. The 5V can be used to power the ESP32. The motors are connected to the H-bridges, which receive power by the batteries directly, with the control signals provided by the Arduino.

The H-bridges are located under the car chassis, where they are responsible for the motor power and direction. As mentioned earlier, the motors are powered directly through the battery, while the signals are controlled by the Arduino Uno. The H-bridges utilized are the L9110h, with the wiring shown in Fig. 2.

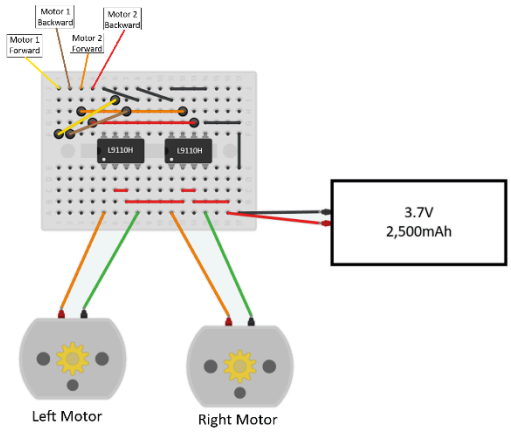


Fig. 2. The H-Bridges circuit with the battery and motors.

Geared DC motors were used to drive the robot. These were 12VDC motors and attached on both sides of the robot. To control the motors based on the fuzzy logic output, a function was developed that takes a single turn radius as input and adjusts the speed and direction of each motor accordingly, allowing the robot to turn with the specified radius.

The 3W LED is an Adafruit Ultra Bright 3-Watt LED mounted to the back of the car chassis, next to the ESP Camera. It is utilized to illuminate the objects in front of the camera to improve visibly for AI processing. The LED is wired to the Arduino to be supplied with constant power. Fig. 3 shows the project with the LED attached.

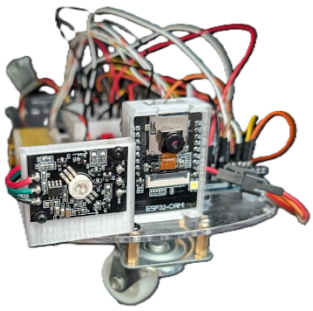

Fig. 3. The backside of the Pac-Man Robot.

The brain of this project was the Arduino Uno microcontroller. It operates at a clock frequency of 16 MHz and features 32 KB of flash memory, 2 KB of SRAM, and 1 KB of EEPROM. The Uno provides 14 digital I/O pins, of which 6 support PWM output, used to drive the DC motors, and 6 analog input channels with a 10-bit resolution. These were used for IR sensors. It operates baud rates of 115200 bps. The board runs at 5V logic levels, and because of this, it needs its ground tied to the ESP32-CAM so that they have a common reference ground. Of all the GPIO, we used a total of 6 digital pins and 3 analog pins.

The Second most important hardware item was the ESP32-CAM. It is a microcontroller module with an onboard camera (OV2640) and wireless communication features. At its core, the ESP32-S chip includes a dual-core 32-bit LX6 microprocessor running at up to 240 MHz, with 520 KB of internal SRAM, 448 KB of ROM, and support for external PSRAM (typically 4 MB on the ESP32-CAM) to handle image buffers and more complex processing tasks. The module features built-in Wi-Fi (802.11 b/g/n) enabling wireless data communication. It includes a microSD card slot for local storage. This microSD card was used to store images when the robot was training. In addition, the camera on this microcontroller is capable of being swapped out for many different EO sensors allowing for application specific customizations. For our application, the standard camera that came with the board worked well. One shortcoming with this sensor was the lack of available GPIO pins. While the standard ESP32 has upwards of 20 GPIO pins, this one only had 4 that could be used. This was the reason of needing an Arduino to ingest all of our GPIO and analog signals and process them. The ESP32 received 5v power and ground from the Arduino and had one of its GPIO pins connected to a GPIO pin on the Arduino. This connection allowed the ESP to send signals to the Arduino, informing the Arduino that a ghost has been detected.

The IR sensors used in this project were the Sharp GP2Y0A21YK0F. These sensors consist of an IR transmitter and receiver along with a built-in logic circuit. It is capable of detecting between 10 and 80 cm accurately. This was deemed to be adequate for our usage. In addition, it runs on up to 7 volts input. and will output from -0.3v to (Vcc + 0.3v). The voltage to distance however is not linear and requires us to use an equation to model this relationship. The figure below shows this relationship. In addition, the sensor will output a distance reading every 38.9 ms (+/- 9.8 ms). This was one of our constraints. The logic circuit on the sensor limited how often and how fast it could output a value.The output timing diagram is shown below. These sensors were placed at 3 different locations and angles. One at 30 degrees, one at 45 degrees, and one pointing straight ahead at 180 degrees. We initially had 2 sensors however because the sensors cannot accurately read under 10cm we were getting a lot of collisions in tight spaces. The inclusion of another sensor looking down the middle mitigated this by a lot. The figure below shows the connection between the IR sensors and the arduino uno. These sensors needed a stable power supply and were therefore powered by the arduino. When powering them with the battery pack via a voltage regulator, we found that our sensor outputs were very erratic and did not give us good usable results.

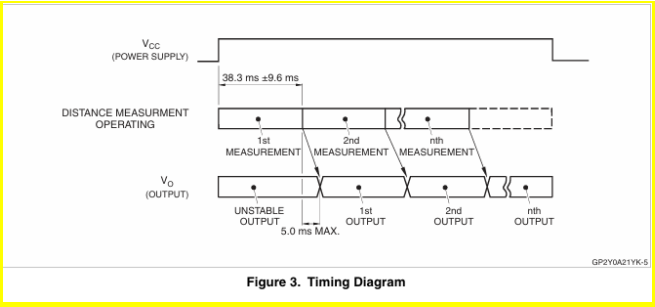


**Figure 3. Timing Diagram**

Fig. 4. Sharp IR sensor Timing diagram

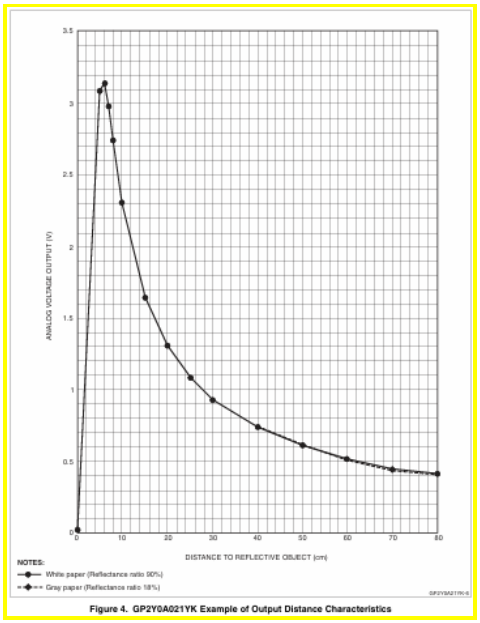


**Figure 4. GP2Y0A021YK Example of Output Distance Characteristics**

Fig. 5. Sharp IR sensor Output voltage

### *A. Decision Tree*

For image recognition, a decision tree was chosen for the implementation. This was chosen because it was a simple AI model that was easy to implement. To use the decision tree, we must be able to extract features from the images capture.

Utilizing the ESP-32 camera, the microcontroller could capture grayscale images for immediate processing. Afterwards, the ESP could perform matrix convolutions on each of the pixels. The ESP would first perform a 5x5 Gaussian Blur on the grayscale image. This was to remove any of the noise produced in the image as well as help some of the edges smoothen out. Afterwards, an edge detection filter was applied to the image to remove all the features except for the edges of shapes in the image. To help with the image clarity, a bright 3W LED was placed next to the camera to allow for the image being captured to be amplified to the camera. This helped the most with noise reduction between the background objects and the main object trying to be captured. With the image now processed, it was now required to determine what features to extract from the image.

In terms of extracting features, we used the following to help determine whether there is a Pac-Man ghost in the image. The first being the sum of pixels in the left, right, and center regions of the image. The next was to find the bounding box of the main object captured. This was done by finding the perimeter of the object, using a threshold grayscale value to remove the noise, and generate a box around it. Afterwards, the coordinates of the bound box, the area, and the ratio were all pulled as features for training. Finally, we looked at the amount of edge detected pixels in an image. Due to our bright LED light, we were able to set a threshold value such that, if the number of pixels is higher than the threshold, the pixel would be counted in the final count. An image of the ghost captured and processed with the ESP is shown in figure #.

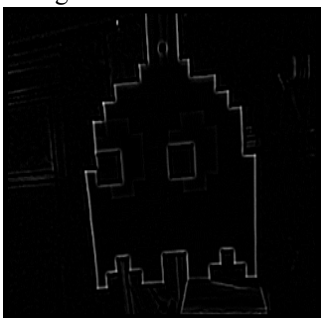

Fig. #. Ghost captured and processed with the ESP

The decision tree used in our project was trained using roughly 2,000 different photos. We let the robot free in our training environment to take photos to train with. We placed Pac-Man ghosts around the area. Then we sorted the images taken into 2 different categories: “ghost” or “no ghost”. The images taken by the camera were grayscale images that were saved as 2D arrays. We then performed gaussian blur and edge detection on them. These arrays were then inputted into a python program that would ingest the training images and then generate a decision tree based on these images.

In the Python decision tree library, there are three main parameters that can be changed to affect the decision tree. The parameters are maximum depth, minimum samples leaf, and maximum leaf nodes. Max depth defines how deep the decision tree is allowed to span to. Minimum samples leaf defines what the minimum sample count is needed to justify making a new leaf. And finally, max leaf nodes define how many final leaves were allowed to be made. To determine the best combinations of parameters, a sweep was performed on discrete values for the three parameters, resulting in 180 different combinations. After the sweep, it was determined that the best configuration was: max_depth = 12, min_samples_leaf = 4, max_leaf_nodes = 25.

The program used the ~2000 samples as well as the parameters to create the decision tree based on the two categories. The figure below shows our decision tree with all its branches.

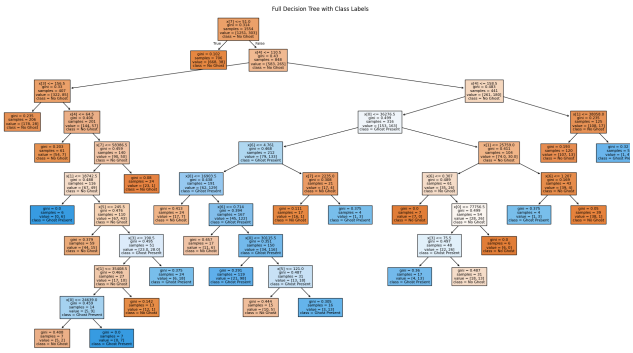


Fig. 6. Decision tree

The decision tree produced highly satisfactory results. When performing a K-fold with 10 K segments, the average error rate came out to be 8.09%. When observing the confusion matrix (shown below in figure #), the confusion matrix shows that the AI is biased to choose the "no ghost" option the most. However, the AI is still able to detect when ghosts present most of the time.

| | | | |
|---|---|---|---|
| | True Class | 1,908 | 153 |
| | nt | 66 | 580 |
| | | No Ghost | Ghost Present |
| | | Predicted Class | |

Fig. 7. Decision Tree K-Fold Confusion Matrix

## B. *Fuzzy Logic*

The obstacle avoidance is driven with the use of fuzzy logic. We utilized the Arduino Fuzzy.h library to accomplish this. We first defined the inputs of our fuzzy system. Those being the three different IR sensors. Those distances were used within our membership functions. Then we defined the output of our fuzzy system which was essentially a speed command to the motors. We had a total of 27 rules that related our inputs to our outputs. A problem we experienced with our rule set was that due to hardware limitations, we were not able to have more than 27 rules.

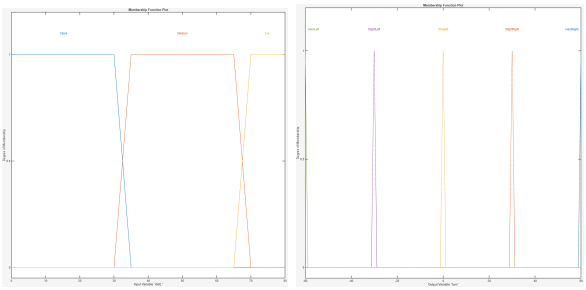

Fig. 7. Input and Output Membership Functions

The fuzzy system, based on the obstacle measurements, would determine what distance to turn the robot to best avoid the obstacle. The fuzzy system could either make the robot go forward or backward. In addition to turning right or left with varying degrees of intensity. WHen turning right or left, our implementation was that the inner wheel would be at a constant speed, while the outer wheel would be the constant speed + a fuzzified value between our determined minimum and maximum speeds. This would allow us to make tight and efficient turns.

The fuzzy system was integrated with the rest of our project such that upon startup, our robot would enter a searching mode. THis would cause it to pivot in a circle until the ESP32-cam detects a ghost. Upon detection of the pac man ghost, the robot will then transition into the obstacle avoidance mode. Here it will use the fuzzy system to flee the ghost for 10 seconds before transitioning back into the searching mode. The figure below shows our state machine.

## C. *Final Implementation*

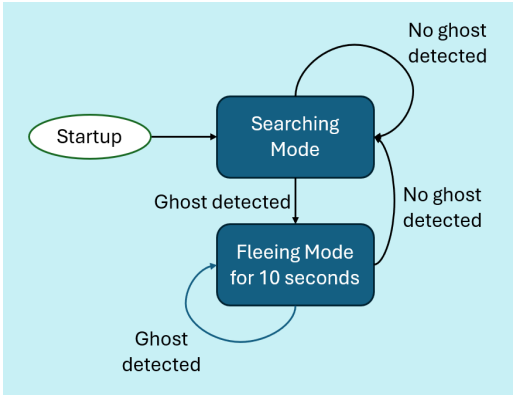


Fig. 8. Program State Machine

# III. Experimental Setup

Training and testing of our robot was done in the same enclosed location. The room was about 10x10 feet with varying lighting in different sections. We scattered multiple different colored pac man ghosts throughout the room for the robot to photograph as training material. The primary purpose of using different color ghosts was so that we can introduce variation to the "ghost detected" dataset and therefore create a robustly trained decision tree. As mentioned earlier, the ghost would roam around the room searching for the ghosts and then run away from the ghost as soon as it was detected.

## IV. Results

The results of our project were very successful. It was successfully able to detect the pac man ghost with our trained decision tree. In addition, the robot was able to successfully avoid obstacles using fuzzy logic.

Looking at the numbers, our decision tree had an error rate of 8.09%. This can easily be lowered further by introducing more training data. Especially in different environments. Another method of lowering our error would be to introduce more features for our decision tree to take into consideration. Which would make it easier for the model to predict our pac man ghost.

Our execution times were also quite optimized. The table below shows our times. The arduino uno was able to process a fuzzy logic motor command very quickly, being only 7ms - 9ms. Looking at our ESP32-CAM, we can see that different image sizes will greatly increase our execution times. The difference between capturing a 240p image and a 600p image is almost 5 times greater. We had to determine what was more important to us, execution time or image quality. Image quality would greatly improve our decision tree results. Our training and test images would have alot more data that we can use for classification. However, if the time to take and process an image is too slow to capture the ghost, there wasn't an advantage of having the higher image quality.

| | Execution Time (ms) |
|---|---|
| Arduino Uno | 7 – 9 |
| ESP-32 (320x240) | 219 – 287 |
| ESP-32 (640x480) | 600 – 764 |
| ESP-32 (800x600) | 919 – 1094 |

Fig. 9. Program State Machine

## Conclusions

This project successfully demonstrated a robot capable of real-time object detection and obstacle avoidance, using Pac-Man training models. Through the integration of an ESP32-CAM for image recognition and an Arduino Uno for motor control, the robot was able to detect and respond to visual threats and navigate around physical obstacles using fuzzy logic. The combination of a decision tree-based vision system with an adaptive fuzzy control scheme resulted in smooth, reliable performance during testing, even in a dynamic and cluttered environment.

One major component was the development and implementation of a decision tree classifier trained on over 2,000 images, allowing the ESP32-CAM to distinguish Pac-Man ghosts with a relatively low error rate of 8.09%. Another was the IR-based fuzzy logic control allowed for obstacle avoidance using the fuzzy logic to create relationships between the input IR sensors and the outputs at the DC motors.

Despite its success, the project revealed several limitations and areas for improvement. The ESP32-CAM, while compact and powerful, was limited in both GPIO availability and processing capability, leading to compromises between image quality and system responsiveness. Additionally, the decision tree could improve with more extensive and diverse training data. Environmental variations like lighting and object occlusion also affected performance.

In terms of future upgrades, experimenting with more sophisticated AI models, like neural networks is something we could take into consideration. A more powerful microcontroller with multiple cores would be preferred so that we can do our GPIO inputs and computations all on one microcontroller. This would also allow us to implement parallel processing techniques such as RTOS.