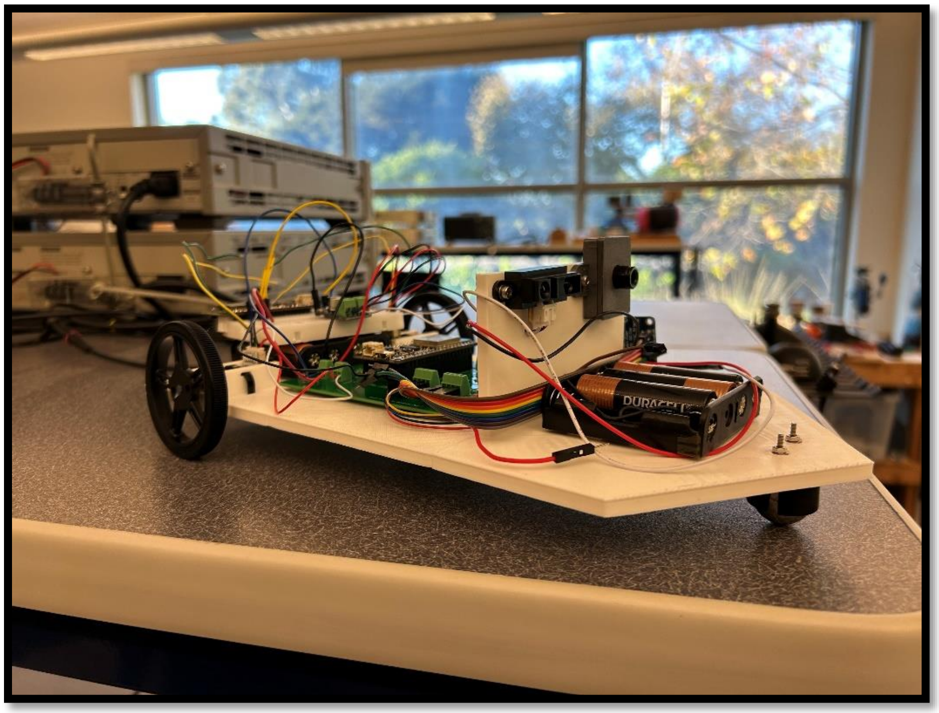
ME 507 Term Project

Heat Seeking Predator Bot



Created by:

Emmanuel Baez

Joseph Balderama

Mechanical Engineering

Cal Poly San Luis Obispo

December 10, 2024

Table of Contents

Introduction

This quarter we set out to create an autonomous robot that would track a person and continuously follow them. We intended to do this by using a thermal camera to identify a person, as well as an infrared distance sensor to be able to keep track of how far that person is. We used two DC motors with magnetic encoders so that the robot could pursue the individual as well as an Inertial Measurement Unit, IMU, to help with motor controls. We were given the ESP32 Firebeetle as the microcontroller of choice for this project. With this microcontroller and sensors in mind, we designed a PCB as well as developed the code to bring the Predator Bot to life.

The motivation behind the design of this project was not only because of our desire to achieve a bot capable of following a person, but also due to the availability of parts readily available to us already. The project began with the idea of using a thermal camera provided by the mechatronics department and using it to our advantage of achieving a follow bot. Since some follow bots use sonar location or other forms of infrared signals, we found it be a promising idea to make use of an object's heat signature to be the primary source of signal acquisition for the bot. The project is visioned to be small but could be increased to be used by a variety of stakeholders who are interested in having a bot follow them around automatically. This bot, with the right modifications, can be used to carry loads, and stay with a reasonable range for a user. Other than these specific uses, this bot is envisioned merely as an interesting project that features multiple sensors and motors. The most outstanding expectation for this project is that it will be able to successfully achieve multitasking capabilities without any studder between the actions desired from the bot.

Specifications

We started the project with the intention of creating a robot that uses heat signatures read by a thermal camera to identify a person, as well as a distance sensor to make sure the robot stays in the person's proximity. It stays in proximity with two independent motors as well as a ball caster which is there for stability purposes. To ensure smooth motor controls, we are also implementing an Inertial Measurement Unit, IMU. We went in with the project with the following specifications for our sensors and for our robot.

Table 1: Measurement Specifications

Parameters	Range	Accuracy	Update Frequency	Test Method
Temperature (°C)	20-40	±5	30	Compare Thermal camera readings to a controlled environment.
Acceleration $(ft\ /\ s^2)$	0-0.5	±0.01	60	Test the IMU by putting it on a linear track.
Distance (in)	2-36	±0.5	60	Verify the distance sensor by measuring readings against a measuring tape or ruler.

Table 2: Robot Pass/ Fail Criteria

Requirement	Criteria	Test Method
Tracks heat signature	Pass/Fail	Have a person move around to see if robot tracks body's heat signature.
Detects objects in a 2-36 in range	Pass/Fail	Move the robot toward and away an object to verify that the distance is being measured.
Communicates thermal heat signatures to web server	Pass/Fail	Verify live heatmap on web page using a test web page.

Design Development

This section will discuss the goal, process, and expectations for the project's hardware and software design development. The following two sections will feature a brief discussion of the options considered, with detailed description of what our chosen design came to be.

Hardware Design

This project initially began with the need for breakout boards for mostly all components for proof of concept. Afterwards, we began to design our own PCB and housing to make this project come to life as we envisioned. The first step following the creation of our partial specification list of this project was to generate a reasonable schematic of the connections needed to achieve desired communication between components. The purpose of this schematic is generally to understand which pins between sensors, motors, and the microcontroller will connect to each other as seen in Figure XX.

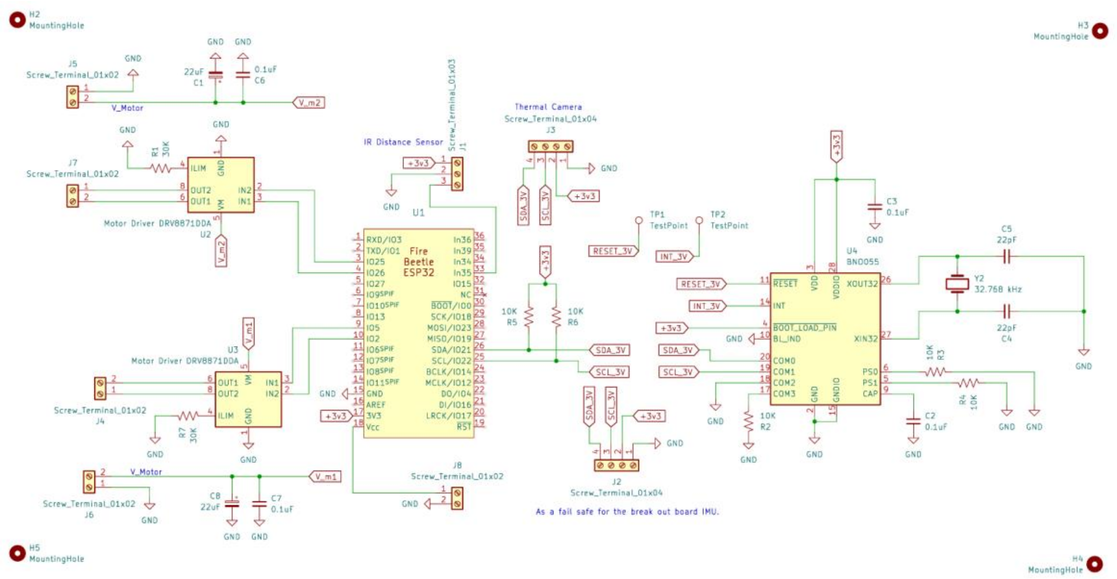


Figure XX: Schematic of the predator bot for designing the connections between the components.

Once we had a verified schematic of our term project, it was essential that we went through each symbol placed and chose adequate footprints to use for our PCB design. This was a very crucial step in the PCB design because we need to acknowledge what sizes of components were feasible to achieve the desire specifications of this project. For example, if we wanted to use a 0.1 uF capacitor rated for a high number of volts, we would not be able to use a very small track pad, so choosing an adequate footprint will ensure we don’t experience issues with component compatibility. The PCB design featured many steps such as placing the components and connecting everything with traces. The process of doing this was more involved than just simply connecting components. We needed to consider voltage ratings, placement/orientation, and thermal properties of this board design. For traces expected to have current flow through them, we would need to make them larger to ensure there isn’t excessive heat buildup. With the heat topic in mind, we also needed to ensure that the motor drivers had proper heat dissipation, in which we had to carefully place vias between the copper plates to allow for heat to leave the chips and be exited through convection in the air. The last consideration was the placement and orientation of the individual components. This stage did not have a right or wrong way of doing it, but rather more of an art form really. The board can be as large as desired to ensure space for everything to connect, but we needed the board to be as compact as possible to reduce weight imbalances. After many iterations of the board design, the final board design can be seen in Figure XX, where it features 7 screw terminals, 2 pads for motor drivers with appropriate use of vias to serve as a heat sink, 1 pad for the BNO05 chip used in the IMU, and many surface mount spots for various resistors and capacitors.

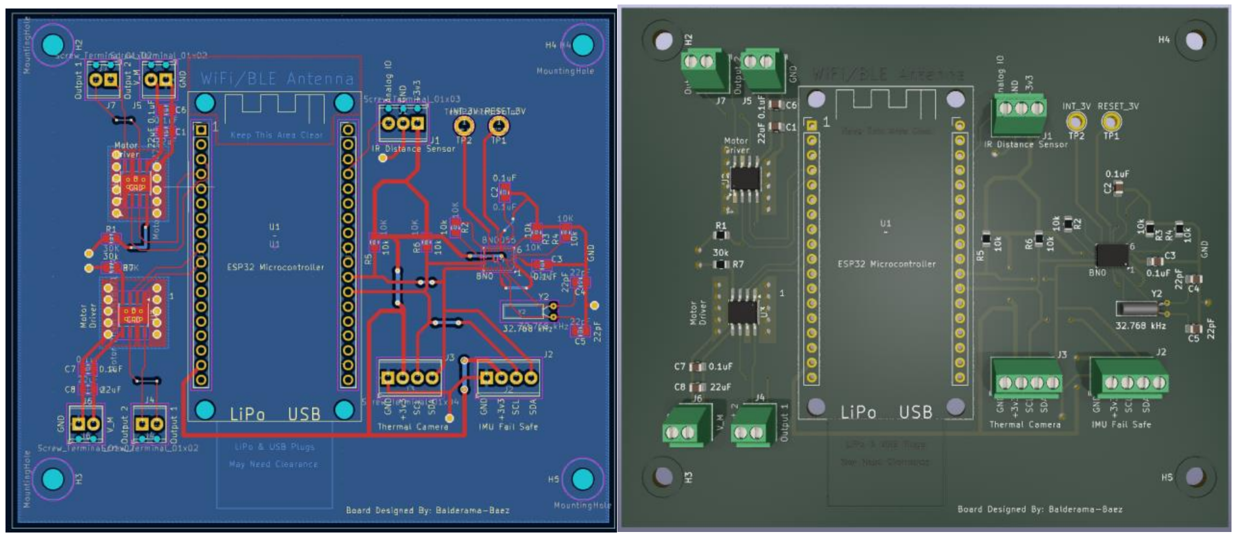


Figure XX: PCB design for the predator bot featuring a 2D design using KiCAD (Left) and a 3D rendered model of our expected outcome of the board (Right).

Once our board design was finalized, we could begin the process of designing housing to mount and connect all of our built or bought components. The housing needed to be large enough that it would hold everything without unnecessary congestion, but small enough that it didn't add excessive weight to the bot and put more stress on the motors. The board design went through multiple iterations, but it was chosen to have a flat base pate, with two rectangular slots to hold the two motors in with the use of zip ties. The base plate also included 4 mounting holes in the center of the plate to mount the finished PCB board, which allowed for some variability of connections between the components. Additionally, the model had a vertical rectangular plate that served merely as a mounting bracket for the thermal camera and distance sensor. Both IR sensors were mounted using fasteners and through holes seen in the vertical plate. The last feature of the base plate was two mounting holes used for attaching the ball caster wheel to the plate using fasteners. The final model design for the housing can been seen in Figure XX.

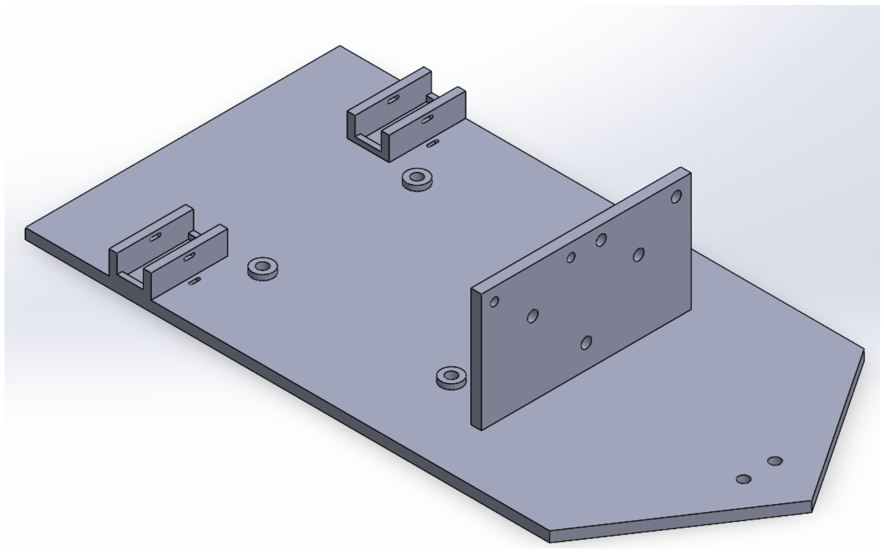
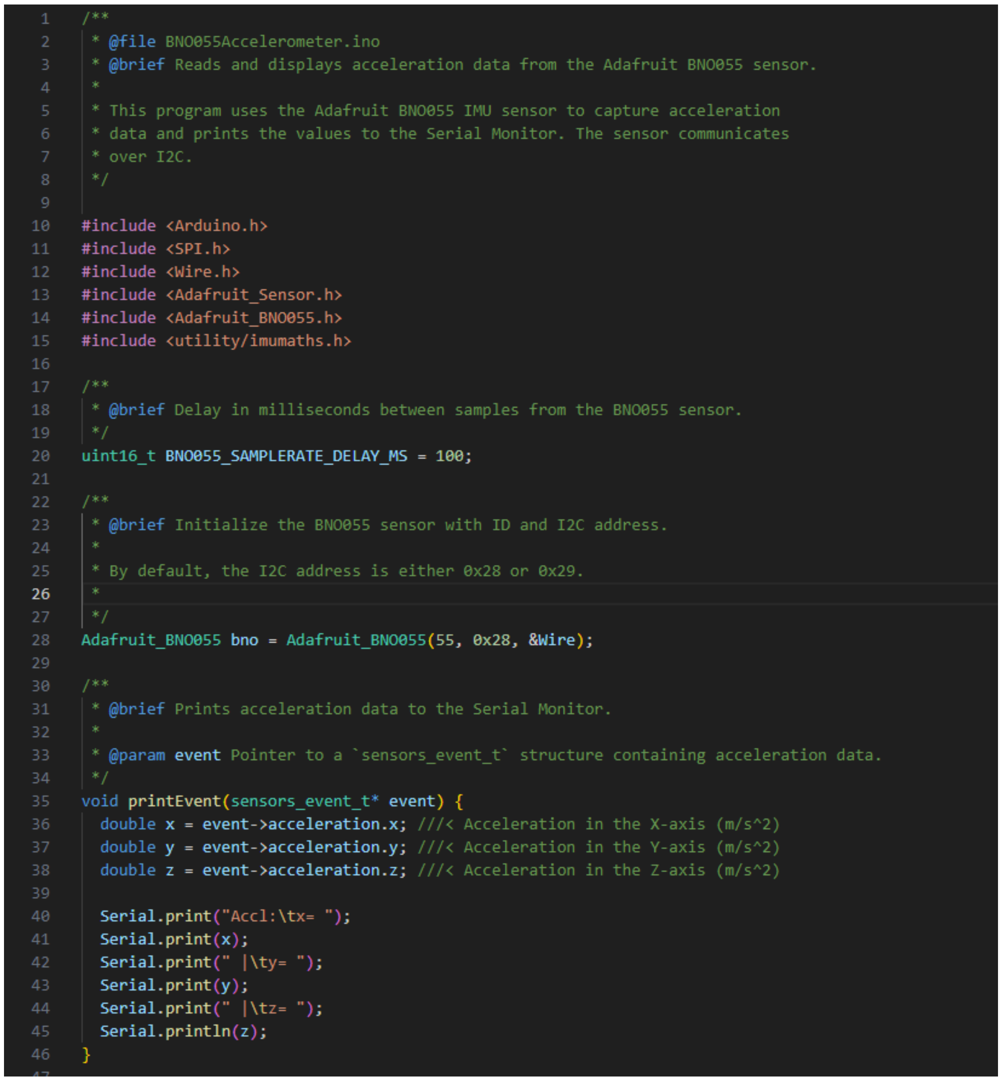


Figure XX: Isometric view of the Predator Bot baseplate that features mounting spots for components of this project.

These hardware designs were an essential step to achieving this project and allowed us to move forward with the project and begin the steps to achieve successful software design.

Software Design

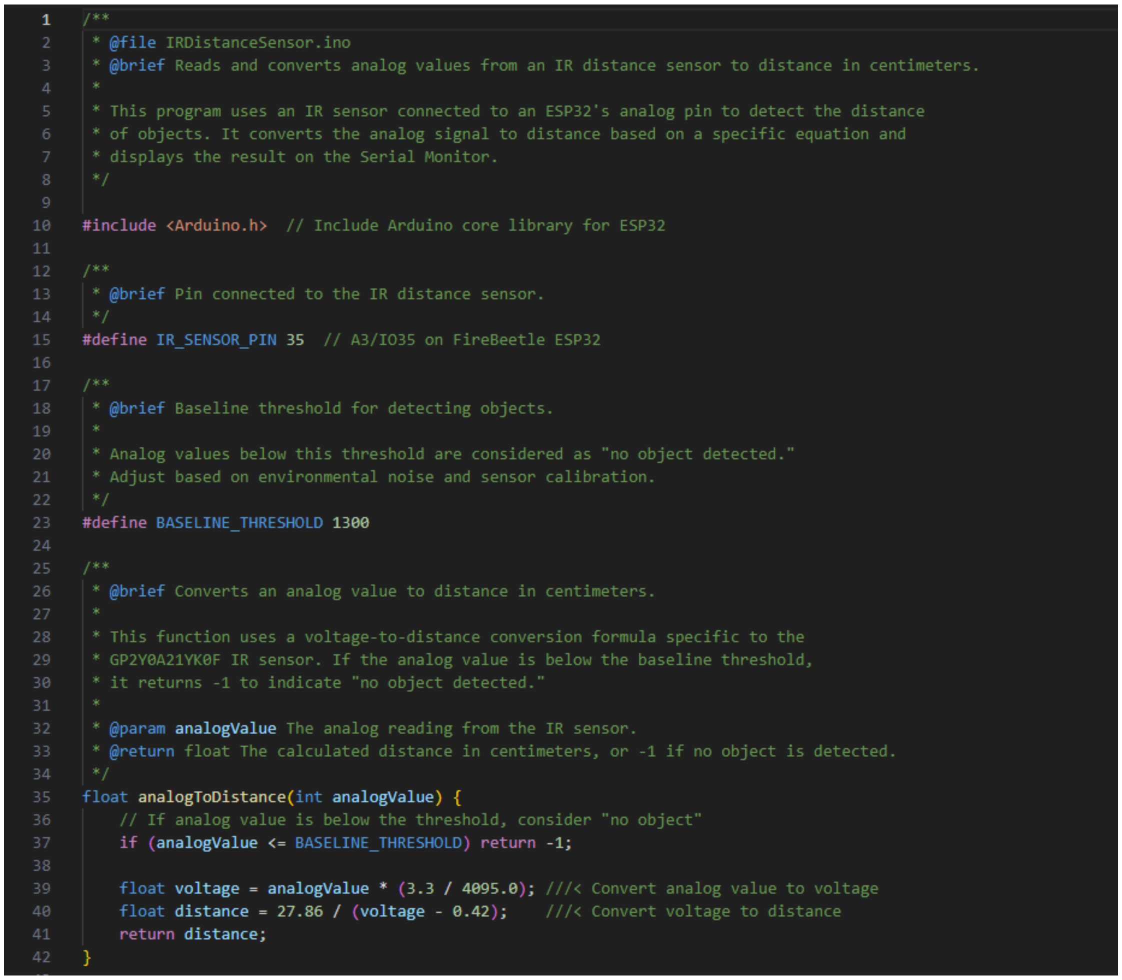
When we first started this project, we believed the best course of action was to get each sensor to work separately, and then at the end integrate them together. This has left us with code for each sensor, but not much for the overall robot. One of the sensors we first were able to get to work was for the IMU. The specific model we used was the Adafruit BNO055. The one nice thing about this sensor is its ease of use and the many resources out there for it. We were able to use Adafruit's public repository on GitHub to learn how to use this sensor. The BNO055 is composed of many other sensors like an accelerometer, gyroscope, and even a magnetometer. So, the hard part of using this IMU is knowing how to get the data you want from the sensor. Luckily with the GitHub repository, we can find the I2C address for any of the sensors we wanted. We were more interested in using the acceleration data the sensor can provide, in hopes of using the robot's acceleration in each axis to refine the motor control.



```
/**
 * @file BNO055Accelerometer.ino
 * @brief Reads and displays acceleration data from the Adafruit BNO055 sensor.
 *
 * This program uses the Adafruit BNO055 IMU sensor to capture acceleration
 * data and prints the values to the Serial Monitor. The sensor communicates
 * over I2C.
 */

#include <Arduino.h>
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>

/**
 * @brief Delay in milliseconds between samples from the BNO055 sensor.
 */
uint16_t BNO055_SAMPLERATE_DELAY_MS = 100;

/**
 * @brief Initialize the BNO055 sensor with ID and I2C address.
 *
 * By default, the I2C address is either 0x28 or 0x29.
 *
 */
Adafruit_BNO055 bno = Adafruit_BNO055(55, 0x28, &Wire);

/**
 * @brief Prints acceleration data to the Serial Monitor.
 *
 * @param event Pointer to a `sensors_event_t` structure containing acceleration data.
 */
void printEvent(sensors_event_t* event) {
  double x = event->acceleration.x; ///< Acceleration in the X-axis (m/s^2)
  double y = event->acceleration.y; ///< Acceleration in the Y-axis (m/s^2)
  double z = event->acceleration.z; ///< Acceleration in the Z-axis (m/s^2)

  Serial.print("Accl:\tx= ");
  Serial.print(x);
  Serial.print(" |\ty= ");
  Serial.print(y);
  Serial.print(" |\tz= ");
  Serial.println(z);
}
```

Figure X: A snippet of the setup code for the IMU. The reason the I2C address is either 0x28 or 0x29 is because if you check the BNO055 repository, you can see that those two addresses correlate to the least and most significant bit for the x linear acceleration data as well as the start for the linear acceleration data registers.

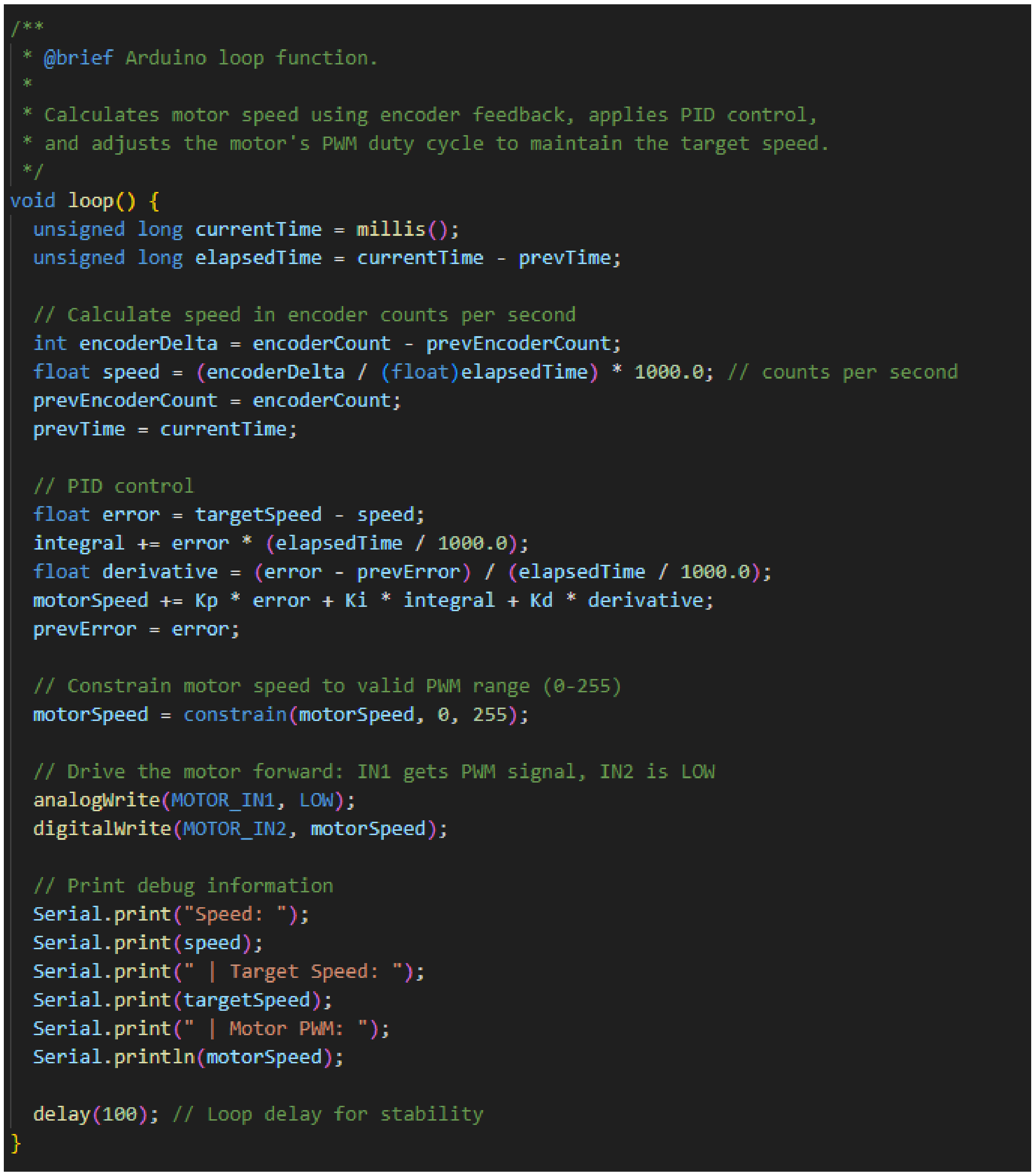
One of the other sensors we got to work early on was the IR distance sensor. This sensor was a bit weird to work with because it worked off specifically off an analog pin. The reason this is weird is because analog pins can measure a range of voltages. So, when coding for this sensor we needed a way to convert the what the analog sensor is reading into a voltage and then convert that voltage into a distance.



```
/**
 * @file IRDistanceSensor.ino
 * @brief Reads and converts analog values from an IR distance sensor to distance in centimeters.
 *
 * This program uses an IR sensor connected to an ESP32's analog pin to detect the distance
 * of objects. It converts the analog signal to distance based on a specific equation and
 * displays the result on the Serial Monitor.
 */

#include <Arduino.h>  // Include Arduino core library for ESP32

/**
 * @brief Pin connected to the IR distance sensor.
 */
#define IR_SENSOR_PIN 35  // A3/IO35 on FireBeetle ESP32

/**
 * @brief Baseline threshold for detecting objects.
 *
 * Analog values below this threshold are considered as "no object detected."
 * Adjust based on environmental noise and sensor calibration.
 */
#define BASELINE_THRESHOLD 1300

/**
 * @brief Converts an analog value to distance in centimeters.
 *
 * This function uses a voltage-to-distance conversion formula specific to the
 * GP2Y0A21YK0F IR sensor. If the analog value is below the baseline threshold,
 * it returns -1 to indicate "no object detected."
 *
 * @param analogValue The analog reading from the IR sensor.
 * @return float The calculated distance in centimeters, or -1 if no object is detected.
 */
float analogToDistance(int analogValue) {
    // If analog value is below the threshold, consider "no object"
    if (analogValue <= BASELINE_THRESHOLD) return -1;

    float voltage = analogValue * (3.3 / 4095.0); ///< Convert analog value to voltage
    float distance = 27.86 / (voltage - 0.42);    ///< Convert voltage to distance
    return distance;
}
```

Figure X: A snippet of code from our IR distance sensor. So, we set a value for how far our sensor could “see” as well as set up a function that converts what is being read to a voltage and then that voltage gets turned into a distance. The voltage is calculated based on the 3.3V it takes to operate the sensor as well as 4095 being the max value for a 12-bit ADC. The 27.86 for the distance comes from a calibration constant as well as the 0.42 being and offset to account for the sensor’s non-zero voltage range.

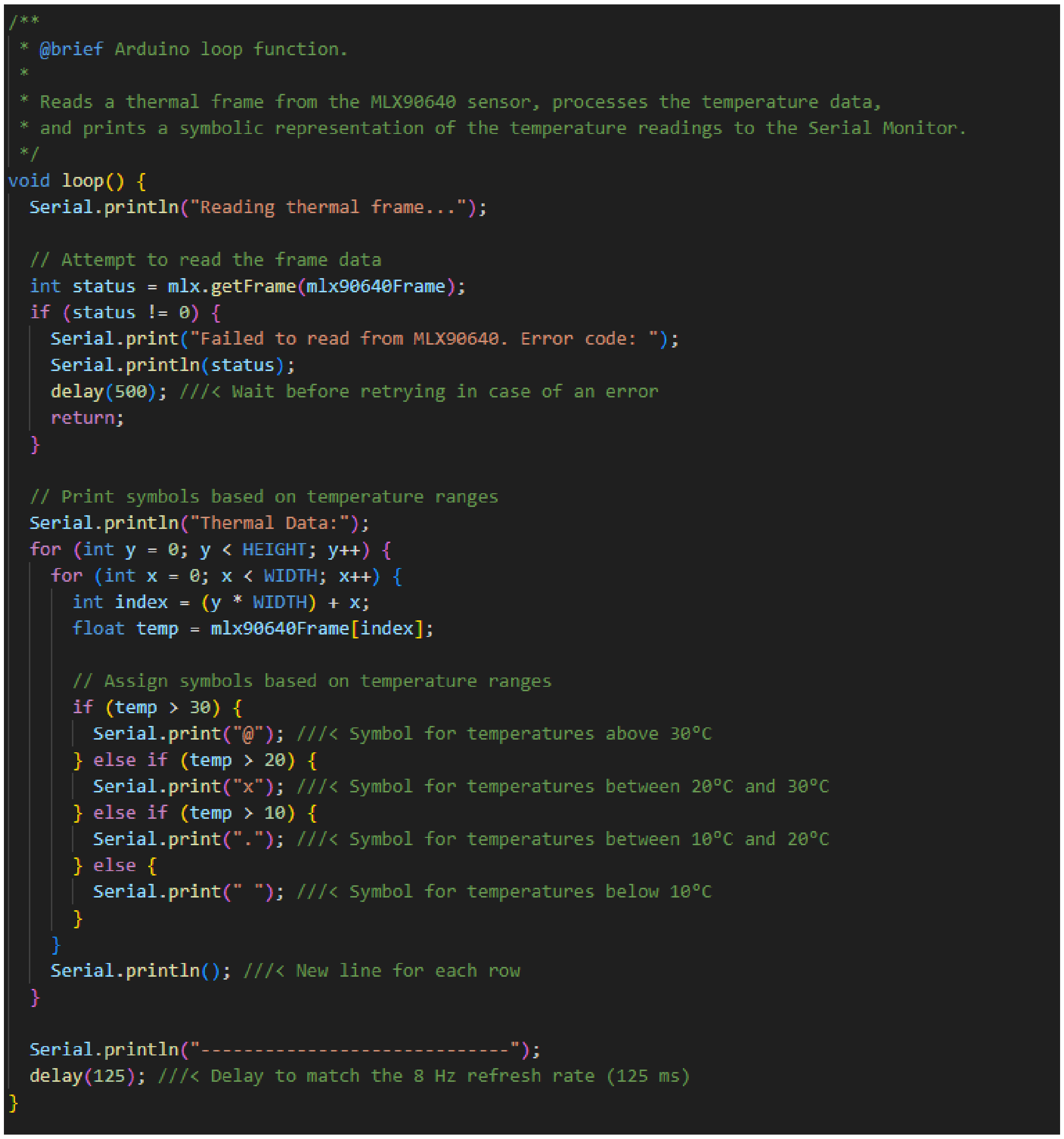
The one part that took us the most time to figure out were the motors. We used N20 DC motors with Magnetic Encoders with a 1:50 gear ratio. For testing we paired these motors with the DRV8871 DC Motor Driver. The biggest issue between the motor and the motor driver was understanding how to connect everything. Once we understood how the motors should be connected, we had to understand how the pulse width modulation, PWM for these motors. From what we found out during our testing is that the motors we were using had two motor inputs for two different PWM signaling. What made us stuck for a while was that when we were conducting our testing for the motors, we had initially set the two motor inputs to be on high. This caused the motor to not function at all, and we learned for it to spin the motors, we needed to set one input to high and the other to low. Once we made that change, we noticed we can get the motor to spin in the other direction by reversing the inputs and setting the other input to low and the other to high.



```
/**
 * @brief Arduino loop function.
 *
 * Calculates motor speed using encoder feedback, applies PID control,
 * and adjusts the motor's PWM duty cycle to maintain the target speed.
 */
void loop() {
  unsigned long currentTime = millis();
  unsigned long elapsedTime = currentTime - prevTime;

  // Calculate speed in encoder counts per second
  int encoderDelta = encoderCount - prevEncoderCount;
  float speed = (encoderDelta / (float)elapsedTime) * 1000.0; // counts per second
  prevEncoderCount = encoderCount;
  prevTime = currentTime;

  // PID control
  float error = targetSpeed - speed;
  integral += error * (elapsedTime / 1000.0);
  float derivative = (error - prevError) / (elapsedTime / 1000.0);
  motorSpeed += Kp * error + Ki * integral + Kd * derivative;
  prevError = error;

  // Constrain motor speed to valid PWM range (0-255)
  motorSpeed = constrain(motorSpeed, 0, 255);

  // Drive the motor forward: IN1 gets PWM signal, IN2 is LOW
  analogWrite(MOTOR_IN1, LOW);
  digitalWrite(MOTOR_IN2, motorSpeed);

  // Print debug information
  Serial.print("Speed: ");
  Serial.print(speed);
  Serial.print(" | Target Speed: ");
  Serial.print(targetSpeed);
  Serial.print(" | Motor PWM: ");
  Serial.println(motorSpeed);

  delay(100); // Loop delay for stability
}
```

Figure X: A snippet of code for our motor controls. For this iteration of the code, we tried to use PID controls to adjust the speed of the motors. So, for PID controls we find the error of desired speed and current speed, the accumulated error over time, as well as the rate of change for the error. These along with Kp, Ki, and Kd constants we set in the setup help calculate the adjustments for our motor. We also try to limit the calculated motor speed in range of a valid PWM range since our microcontroller uses an 8-bit timer for generating a PWM signal.

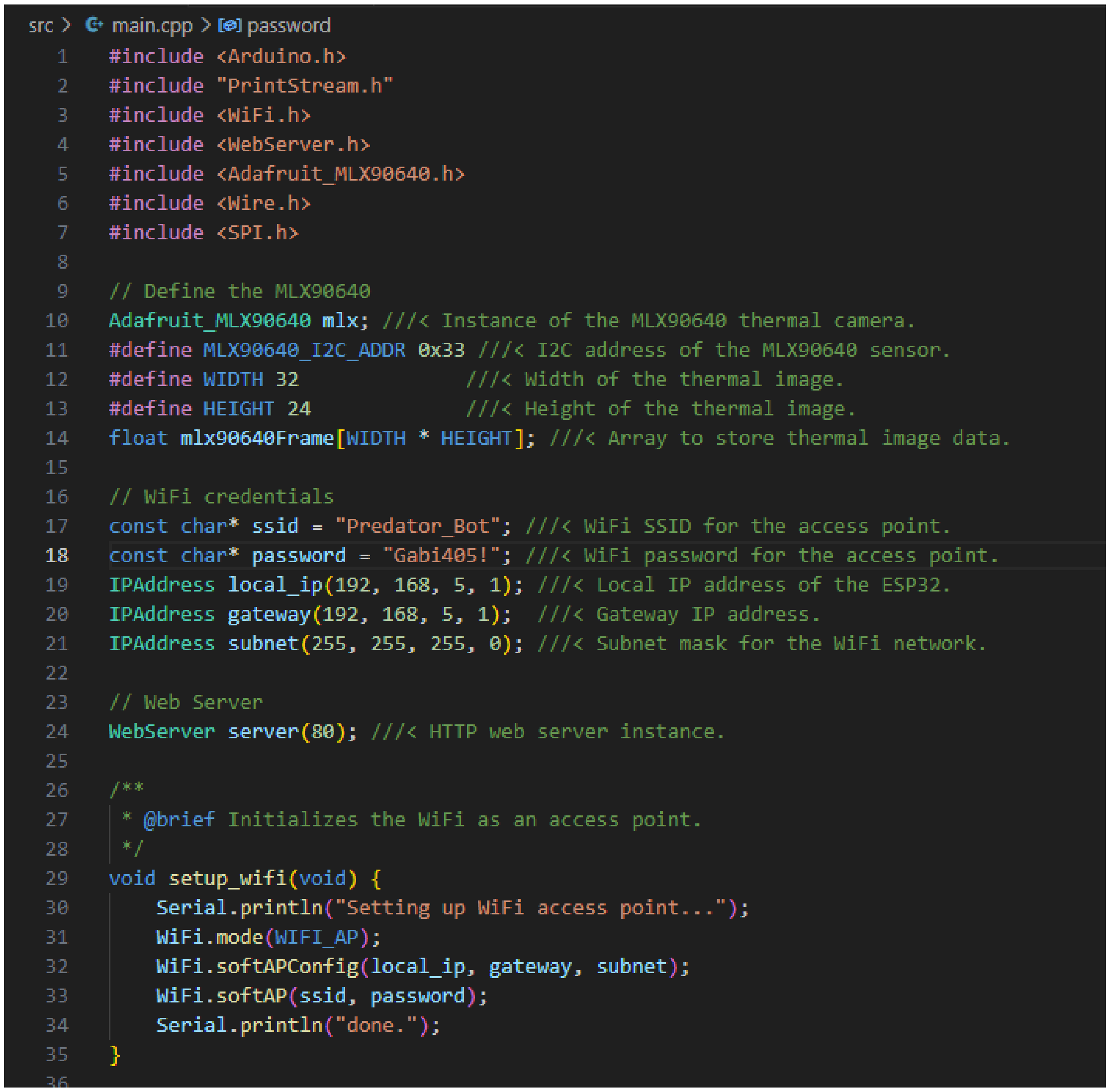
The last sensor used was the Thermal camera. This sensor works very similar to the IMU in sense that they both use I2C. So similarly to debugging the IMU, we needed additional libraries to see how the thermal camera functioned. The Thermal camera we used was an Adafruit MLX90640, and Adafruit also had a repository for it on GitHub. One thing we did to help in the debugging process was assigning heat temperatures an ASCII symbol. This makes it so instead of a block of numbers being printed a low-resolution image can be captured.



```
/**
 * @brief Arduino loop function.
 *
 * Reads a thermal frame from the MLX90640 sensor, processes the temperature data,
 * and prints a symbolic representation of the temperature readings to the Serial Monitor.
 */
void loop() {
  Serial.println("Reading thermal frame...");

  // Attempt to read the frame data
  int status = mlx.getFrame(mlx90640Frame);
  if (status != 0) {
    Serial.print("Failed to read from MLX90640. Error code: ");
    Serial.println(status);
    delay(500); ///< Wait before retrying in case of an error
    return;
  }

  // Print symbols based on temperature ranges
  Serial.println("Thermal Data:");
  for (int y = 0; y < HEIGHT; y++) {
    for (int x = 0; x < WIDTH; x++) {
      int index = (y * WIDTH) + x;
      float temp = mlx90640Frame[index];

      // Assign symbols based on temperature ranges
      if (temp > 30) {
        Serial.print("@"); ///< Symbol for temperatures above 30°C
      } else if (temp > 20) {
        Serial.print("x"); ///< Symbol for temperatures between 20°C and 30°C
      } else if (temp > 10) {
        Serial.print("."); ///< Symbol for temperatures between 10°C and 20°C
      } else {
        Serial.print(" "); ///< Symbol for temperatures below 10°C
      }
    }
    Serial.println(); ///< New line for each row
  }

  Serial.println("----------------------------");
  delay(125); ///< Delay to match the 8 Hz refresh rate (125 ms)
}
```

Figure X: A snippet of code for our thermal camera code. This is the loop function that is reading the data from the thermal camera and assigning those values an ASCII symbol. The range of temperatures comes from initial testing of the sensor. The average body temperature is 37°C and the average room temperature is 20°C, so we made it easier to capture what we assumed were body temperatures, but the sensor could see a hotter signature and assume it is a person.

One function we hope to do alongside our thermal camera was to have the ASCII heat signature streamed to a website, so that people could see what the robot is looking at. Creating the webserver wasn't the hardest issue, it was more about getting the data transferred. We created a test code using the example given to us on canvas and tried to use that as a reference in creating our own web server. Unfortunately, we weren't able to get done in time.



src > main.cpp > password

```cpp
#include <Arduino.h>
#include "PrintStream.h"
#include <WiFi.h>
#include <WebServer.h>
#include <Adafruit_MLX90640.h>
#include <Wire.h>
#include <SPI.h>

// Define the MLX90640
Adafruit_MLX90640 mlx; ///< Instance of the MLX90640 thermal camera.
#define MLX90640_I2C_ADDR 0x33 ///< I2C address of the MLX90640 sensor.
#define WIDTH 32              ///< Width of the thermal image.
#define HEIGHT 24             ///< Height of the thermal image.
float mlx90640Frame[WIDTH * HEIGHT]; ///< Array to store thermal image data.

// WiFi credentials
const char* ssid = "Predator_Bot"; ///< WiFi SSID for the access point.
const char* password = "Gabi405!"; ///< WiFi password for the access point.
IPAddress local_ip(192, 168, 5, 1); ///< Local IP address of the ESP32.
IPAddress gateway(192, 168, 5, 1);  ///< Gateway IP address.
IPAddress subnet(255, 255, 255, 0); ///< Subnet mask for the WiFi network.

// Web Server
WebServer server(80); ///< HTTP web server instance.

/**
 * @brief Initializes the WiFi as an access point.
 */
void setup_wifi(void) {
    Serial.println("Setting up WiFi access point...");
    WiFi.mode(WIFI_AP);
    WiFi.softAPConfig(local_ip, gateway, subnet);
    WiFi.softAP(ssid, password);
    Serial.println("done.");
}
```

Figure X: A snippet of code for creating the web server. So instead of making the thermal camera a separate class, we tried to recreate it alongside the code that would generate and manage the webpage. This code tries to make use of the ESP32 Wi-Fi capabilities letting the ESP32 create its own Wi-Fi network. So, an individual could look for Wi-Fi connections on any of their devices and should see "Predator_Bot" as one they could join. Once connected they could open any browser and type in the IP address and that they would be taken to the webpage associated with our bot.

Although we got each individual sensors to work on their own, we had little time to integrate them together. We would have created a class for each sensor and used the given code to create tasks and shares. Although we did not get to implement it, we did have a task diagram outlining what we believed to work for our robot.

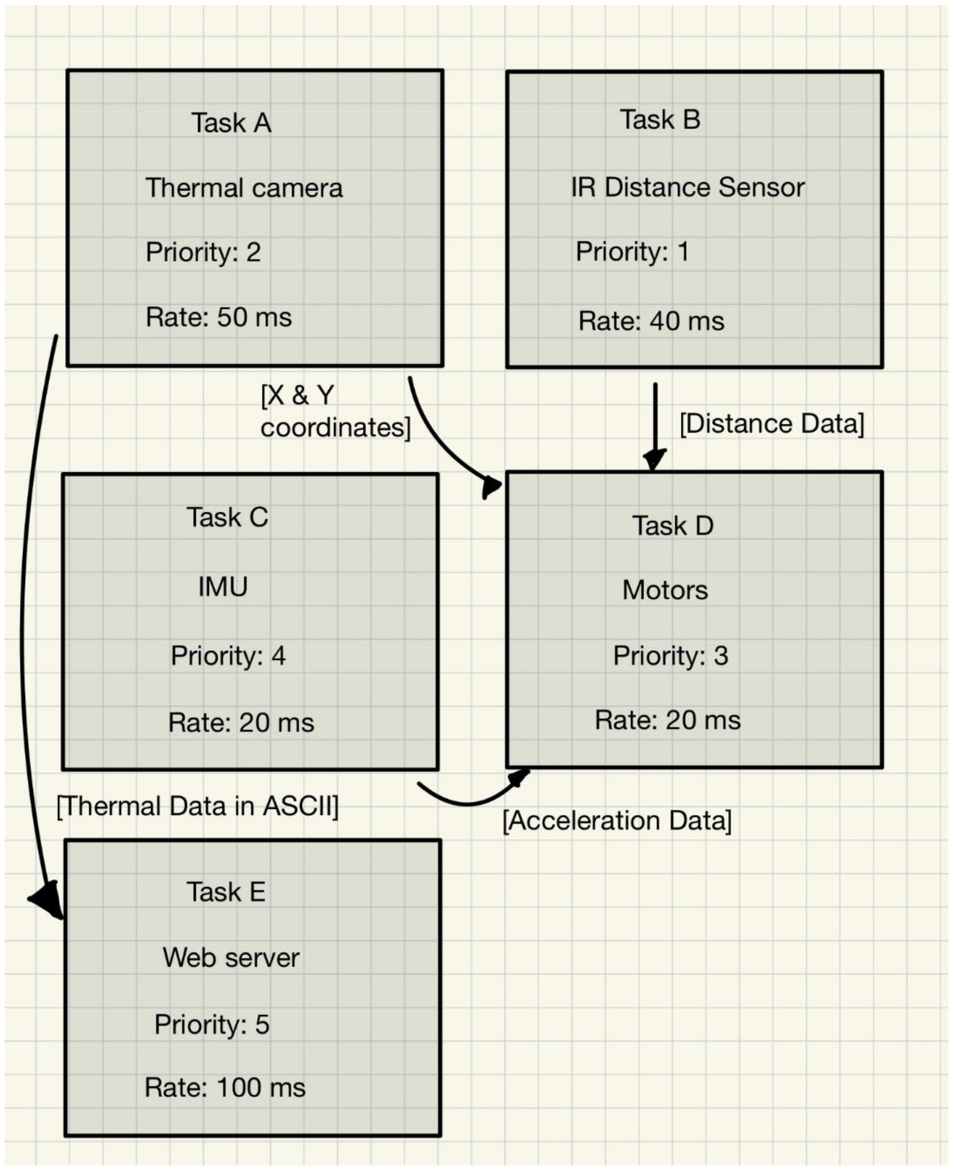


Figure X: Task Diagram for our robot.

Results

The results of this project were far from straightforward, but the process exemplified Cal Poly's "Learn by Doing" philosophy. From the outset, the PCB design for the bot faced significant challenges. While the PCB included all the major components necessary for the project to function, it lacked several minor but critical elements. Missing features included screw terminals, grounded pins, and essential connections for communication and power. To address these shortcomings, we had to manually solder jumper wires to create operational connections. Despite these adjustments, we were able to power and integrate the IMU, distance sensor, and thermal camera. These sensors communicated adequately with the microcontroller, producing reasonable outputs on the computer.

However, issues arose with the motors and their connections to the motor driver and microcontroller. We discovered that our design lacked a way to power the motor drivers and monitor or adjust the motor encoders, which were critical for achieving the project's intended functionality. To address this, we used motor driver breakout boards and jumper wires with a circuit board to enable some degree of motor control via PWM signal adjustments. Once the motors were operational, a new challenge emerged: the motors' maximum speed was too low for the project specifications, and their torque was insufficient to overcome internal friction and the bot's weight-induced friction on the wheels.

Although the final results fell short of our original vision, the project offered invaluable lessons. In hindsight, the design phase required greater attention and would be approached differently in the future. Conducting a more thorough literature review and performing preliminary hand calculations would be essential for developing a successful design. We now recognize the importance of the early stages of project development in achieving desired outcomes.

For PCB design, we learned that a clear conceptualization of how all components connect is crucial. Power traces often require more consideration than communication traces, and future designs should include screw terminals to accommodate breakout boards, reducing the likelihood of overlooked features.

Despite its challenges, this project was one of the most enjoyable experiences we've had. The lessons learned have strengthened our understanding of mechatronics, and we look forward to applying this knowledge to future projects.

Appendices

A. Links

Adafruit. “Adafruit/ADAFRUIT_MLX90640: MLX90640 Library Functions.” *GitHub*, github.com/adafruit/Adafruit_MLX90640. Accessed 9 Dec. 2024.

Adafruit. “Adafruit/ADAFRUIT_BNO055: Unified Sensor Driver for the Adafruit BNO055 Orientation Sensor Breakout.” *GitHub*, github.com/adafruit/Adafruit_BNO055. Accessed 9 Dec. 2024.

Our repository at https://github.com/Emnabz909/Predator-Bot

B. Specification List

ME507 **Fall 2024**

Partial Specification List for Projects

Team Last Names Balderama, Baez

Description of Project

This project will be a thermal seeking robot that features a thermal sensor/camera track someone's heat signature and a distance sensor to determine how far the bot is from the user. The bot will be moving with 2 independent motors and a ball caster wheel for stability purposes. To ensure proper control of the bot, the project will also feature motor drivers and an IMU.

Measurements

Measurand (units)	Range	Accuracy	Frequency (Hz)
Temperature (F)	95 - 105	+- 5	30
Acceleration (ft/s^2)	0 - 0.5	+- 0.01	60
Distance (in)	2 - 36	+- 0.5	60

Actuation

Thing to be Pushed	Range (max, min)	Frequency (Hz)
Both Motors	(0,300)	30
Motor Drivers	(0,30)	60
IMU	(0,60)	60

Networking

Information to be communicated	This bot will send a live heat signature map in aschii format to notify the user of signal acquisition. The page will also inform the user of the current speed and distance away from it's target.
Method (HTML, MQTT, Sockets, *etc.*)	HTML
Update Frequency	60Hz