

**Summer 2022**

Kenneth Sutter – skenneth@pdx.edu

Phil Nevins – pnevins@pdx.edu

Electric Vision

Team Project

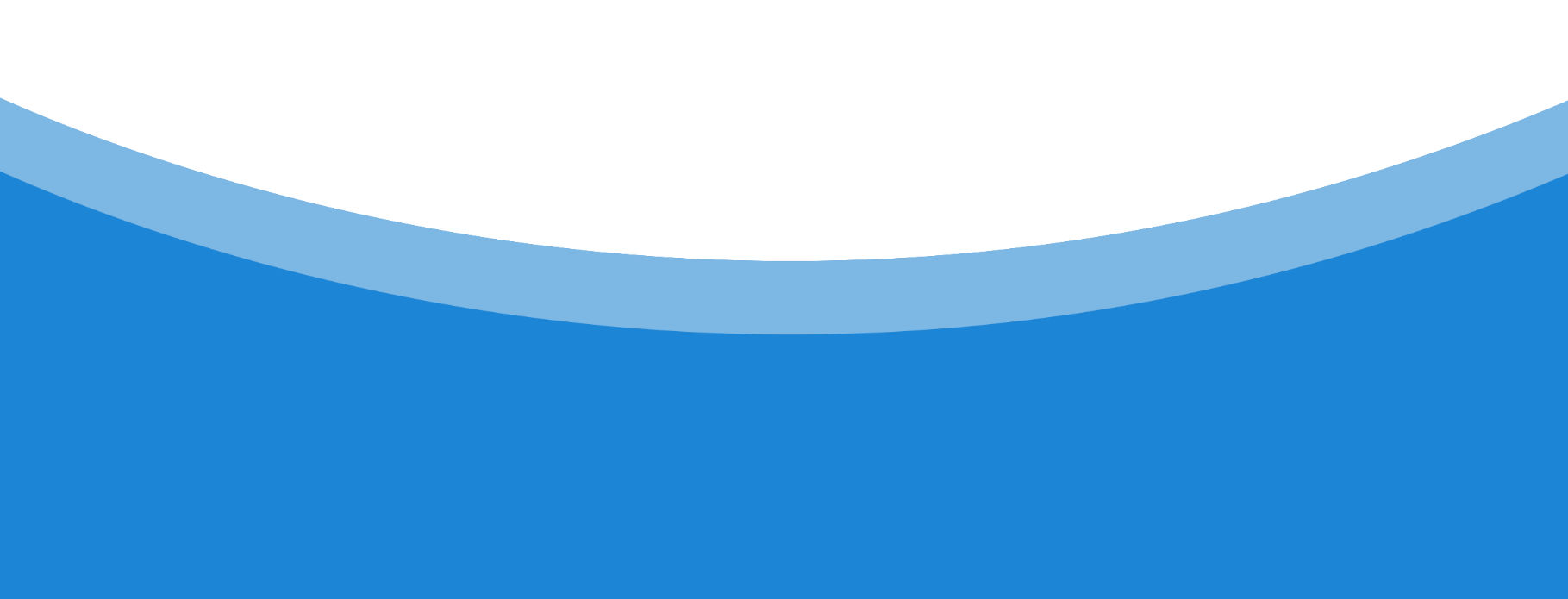


Table of Contents

[**Executive Summary** 2](#_Toc110847278)

[Project Motivation and Inspiration 2](#_Toc110847279)

[Project Requirements 2](#_Toc110847280)

[Project Planning 3](#_Toc110847281)

[SCRUM Implementation 3](#_Toc110847282)

[Prototype Description 4](#_Toc110847283)

[Robot controls 4](#_Toc110847284)

[Temperature and humidity sensor 4](#_Toc110847285)

[Pan tilt arm 5](#_Toc110847286)

[Camera 5](#_Toc110847287)

[Lighting 6](#_Toc110847288)

[Thermal sensor 7](#_Toc110847289)

[Frame 8](#_Toc110847290)

[Final Prototype 9](#_Toc110847291)

[Lessons Learned 10](#_Toc110847292)

[Unfinished and Future Steps 11](#_Toc110847293)

[Conclusion 12](#_Toc110847294)

[Appendix 12](#_Toc110847295)

# Executive Summary

This document serves as guide and timeline to the Electric Visions team project. This was the term project for ENG211/212 for the summer term of 2022 at Portland State University. In this document we will cover the motivation for the project, the project requirements, project planning, how we implemented SCRUM into our team, a description of our prototype, a description of all the subsystems involved, lessons learned, and the steps we would take in the future if we had more time and money to work on this project.

To summarize the project in one paragraph. We decided to build a robot to help firefighter’s complete tasks on emergency scenes. This is in hopes to help reduce risk as well as the number of human resources needed by the fire department. This robot will include a camera, a frame with tracks, a pan tilt arm for the camera, lighting, and a temperature and humidity sensor. We had hoped to have a thermal sensor but sue to supply chain issues this was not possible. The robot is fully wireless and controlled by HTML interfaces. This was an extremely fun project we are proud to have completed. This was an amazing learning opportunity and we have learned copious new skills, techniques, and lessons throughout the process of working on this project.

# Project Motivation and Inspiration

Here we will discuss our team’s motivation as well as inspiration for creating the project. As a team when we were deciding what we wanted to build for our project. We wanted to make sure our project would be solving a real-world problem. Both of us wanted to make a difference in the world and help other humans. We chose to create a robot to help firefighters in the field. The idea was to have a robot that would be able to do tasks for the firefighter’s top help reduce the number of human resources needed for these tasks so the firefighters could focus on the more important tasks. This is an important topic for Ken because he was a volunteer firefighter for five years when he was younger. This also brought along some experience as to what firefighters are dealing with and the tasks, they could need help with.

This is where we where able to do some research and use Kens previous experience to identify a need. The need we identified for firefighters was a robot to help mitigate the vastly diverse situations, terrain, and operations firefighters carry out. As well as the need to prioritize resources to be as effective as possible. Firefighters are tasked with a different situation every time they go on a call. A lot of times the initial arriving unit will have limited or no backup and almost all firefighting operations are done by humans. Vast high-rises, confined spaces, limited visibility, dangerous situations, and unpredictable victims are just some of the issues our firefighters face on a day-to-day basis. Having a robot to be able to survey floors of a high-rise or be able to fit into confined space that humans cannot. Can greatly increase the effectiveness of firefighting operations. For example: firefighters always work in teams and during an active fire must survey every floor during and after the fire for victims and hotspots. If a robot were able to do this being operated by one human and cover multiple floors it would free up that team of firefighters to accomplish more essential tasks in a more efficient manner. This is how our idea was born for the EVrobot. We decided to create a robot with many subsystems to mitigate as many of these tasks as possible.

# Project Requirements

Now we will discuss the project requirements for the ENGR-211 and ENGR-212 classes in summer 2022 at Portland State University. We have been asked to create a project as the final deliverable in this class. For this project we will be designing and building a prototype. To design and build this prototype we are being asked to use the SCRUM method. This includes daily standup meetings with out teams. Designated sprints that end with a new iteration of the prototype. Each iteration should build on the previous prototype and add new functionality or features. We use this method as opposed to the waterfall method because it allows us to have working prototypes and not have to wait for everything to be done before showing our work. Waterfall method can lead to long lead times and lots of conflicting timings. This is because in waterfall method everything is planned out in the planning stage and if things do not follow the master plan it causes huge backups and delays.

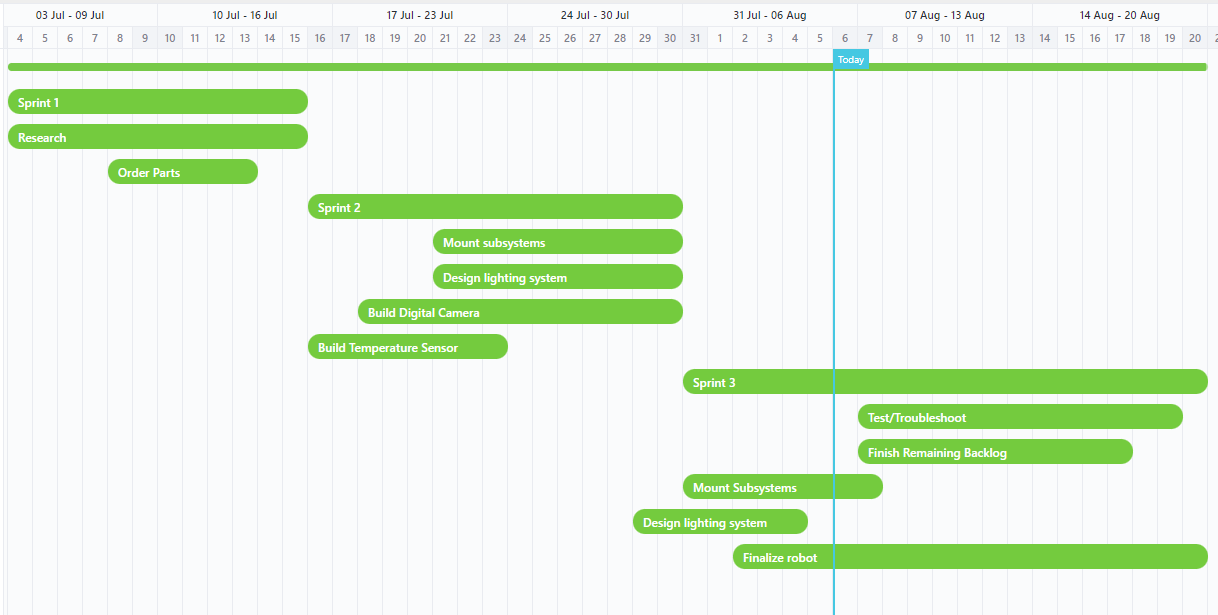
For this project, the class was placed into teams of two to five using a personality and skills test called CATME. This test allowed students to have teams with various skillsets and personality traits to form a more balanced team. After being placed into teams we were asked to begin working on our projects. We had a total of seven weeks to build and complete this project. The final deliverables are a presentation and demonstration of our final project as well as this written report. For each team we presented our initial idea to the class. Other teams as well as the professor would ask questions and give feedback to help the presenting team to solidify their project ideas. This worked as a check to make sure the projects where possible and reasonable to finish in the seven weeks.

Once we moved into the sprint phase of the project, we were asked to meet with our scrum master once a week. The scrum master was a teachers assistant named Rahda. We where able to meet with Rahda 3 times throughout the term. Her goal was to ensure our projects where on track and that we were following the SCRUM methodology. Rahda was also there to help with our projects if we got stuck. We had triweekly class meetings throughout the term. Out team would use these class meetings as out SCRUM meeting times. At the end of the seven weeks of work we then present and demonstrate our final projects for the term.

# Project Planning

Here we will discuss our teams project planning phase. Our team decided to use the first week of sprint one for research and planning. This included making a list of possible components. Creating a cardboard prototype to help with the layout of components. It also included creating a Gantt chart for scheduling and task tracking. Below we have included our final Gantt chart for viewing. We decided at this time that because the frame was not going to arrive for four weeks we should work on subsystems and then bring it all together at the end. This allowed for a more modular structure to pour project. It also allowed us for more flexibility to what subsystems we could try out or prototype. This made the Gantt chart especially important. This is because we wanted the subsystems to be completed or close to completed with the frame arrived. This is so we can work on integrating all the subsystems. Throughout the project we kept to the Gantt chart well we did have to reorganize the chart a couple times, but this was a huge part of our project planning and organization. Overall out team felt highly organized and stuck to our plans as much as possible.

**Figure 1.0: Final Gantt Chart**



# SCRUM Implementation

This is where we will be discussing our teams use and implementation of the SCRUM method. Our team decided to plan our sprints once every two weeks. This would give us time to work on subsystems then get advice or help if needed. Our goals for each sprint where to finish one subsystem or implement something new each sprint. This sprint schedule fit well into our workflow were able to iterate all the subsystem multiple times. We met our sprint objectives most of the time with limited items added to the backlog each sprint. Our team did SCRUM standup meetings two to three times a week with mini check-ins on all the other days. Our mini checking consisted of communication through discord. We would sometimes communicate 3 or 4 times a day for advice or just to check in. this was a benefit of having a small team it made communication extremely easy and effective. Throughout the term we met with Radha 3 times during the term once for each sprint. Our SCRUM meetings were successful. We always had new information to report, and it kept us moving through the project. Overall, the scrum method was extremely helpful to the success of the project. This was a valuable learning experience, and we will be using SCRUM on future projects. In Appendix 1.1-1.3 we have included images of our Trello board for each sprint.

# Prototype Description

In this section of the report, we will be discussing the prototype as a whole and breaking down each subsystem. While designing and building our prototype we had some supply chain issues and did not have the frame or control motors for the first four weeks of the project. This led us to the decision to break the project up into subsystems that way we can use the iterative design process on each subsystem instead of waiting for the frame to be able to start working on the prototype. This allows us to continue working on the project as well as iterate our design to have the best subsystems possible by the time the robot was built and functional. Below I will break down each subsystem then talk about the final prototype.

## 

## Robot controls

To make the robot frame mobile we have two DC 6V-12V 300RPM motors. These connect to the drive gear of the tracks. Our first iteration of controlling these motors to perform motion was just taping the positive and negative wires to a 9V battery. This allowed the motors to move but continuously. This was not verry helpful since we needed to control stop, start, and directional controls. This led to the second iteration which was using a Arduino UNO with the L298N motor driver board. This allowed us to control the motors with pulse width modulation. The L298N motor control board controls the voltages sent to the motor from the battery. The one bis issue with using the Arduino UNO was the Arduino needed to be plugged in to control the motors. So the third iteration was with ESP32 and the L298N motor driver board. This allowed us to have the ESP32 create a Wi-Fi web server and create an interface using HTML. we where also able to use the webserver this ESP32 created to connect to the ESP32 of the temperature and humidity sensor and display their data as well. This was a big breakthrough in the project for making it be wirelessly controlled. The fourth iteration included implementing battery power system so it would be wireless. For this we used a 9.6V battery pack. We needed to have plenty of current because the Wi-Fi on the ESP32 uses a lot of current and it will cut out if it does not have a consistent supply of current to the board. This took awhile to troubleshoot since we could not figure out why the Wi-Fi was cutting out. We where able to determine this by using a multimeter to see the current drop. We have included our code in Appendix 1.5 for the subsystem.

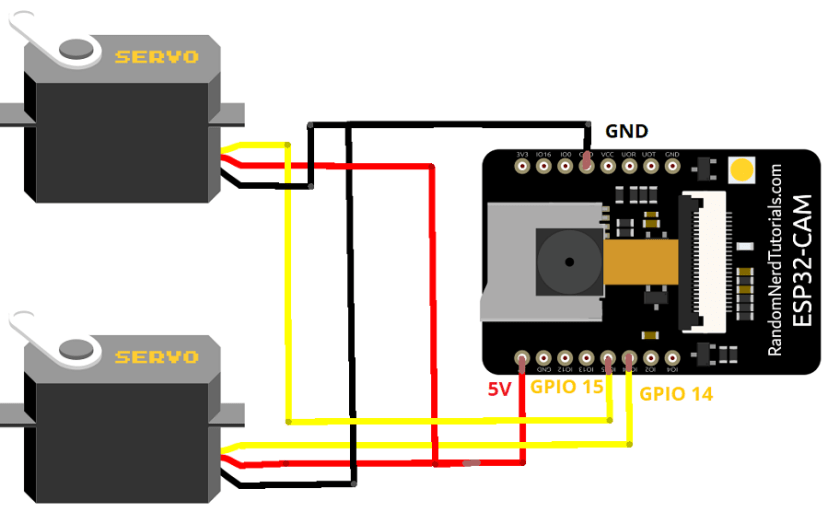
## Temperature and humidity sensor

The temperature and humidity subsystem is designed to take ambient temperature and humidity readings. This was intended for the operator to use this data to ensure they are operating the robot in a safe environment. The first iteration we used the LM35 with Arduino UNO we decided on the LM35 and an Arduino UNO because the LM35 is a more precise sensor than the DHT11. This led to issues with the Arduino UNO needing to be plugged in to power and get the data from the sensor. Since we wanted a wireless robot, we decided to move to our next iteration. The second iteration we moved to using ESP-32 with LM35. This fixed the issue with the bot needing to be wired but we kept encountering a lot of temperature drift with this sensor as well and inaccurate readings. To fix this we decided to use the less precise sensor the DHT11. So, our final iteration is ESP32 and DTH11 sensor. This allowed us to be fully wireless and have accurate temperature readings. Even though they are not as precise as the LM35. To achieve the wireless connectivity and data streaming. We connected this ESP32 to the ESP32 webserver that controls the motors. This allowed us to control the robot as well as read the sensor reading on the same webserver. Overall, even though we would have rather had the better sensor we are still happy that we where able to get this system to be fully wireless as well as working. Below we have included an image of the wiring of the DHT11 and the ESP32. We have also included a copy for the code in in Appendix 1.6.

## Pan tilt arm

The pan tilt arm shown below was added to move the camera to get various camera angles. Our first iteration of a camera mount was, we taped the camera sensor to the frame. This worked but we had a lot of blind spots as well as most of our image was looking at the ground. The second iteration we tried to design was a mechanical swivel or gimble for the camera to be attached to. This proved to be difficult and time consuming. We found this cheap pan tilt arm that achieved what we needed for the design. this arm had two basic servos and needed to be wired for 5V, ground, and data. We then had our third iteration the pan-tilt arm and super glues the camera sensor to the assumably. In this iteration of the project, we had used and Arduino UNO and a joystick to control the arm. This allowed us to have many different viewing angles and well as be able to still see the surroundings if the robot get stuck. This was problematic because we wanted the robot to be fully wireless. This led to our fourth iteration of the pan tilt arm. In the fourth iteration we connected the arm to the ESP32 cam. When writing the code for the ESP32 cam we learned how to create a web server and page using HTML. It then would share the information on an HTML based web server. This allowed us to have the arm be controlled and the video to be viewed on the same HTML page. It also allowed us to use one device to view both on the same IP address. The interface for this HTML code was unrefined but it worked for controlling the pan tilt arm as well as viewing the video feed. We also ended up using this webserver with an HTML page for the robot controls and broadcasting the temperature and humidity sensor data. Below we have included an image of the wiring diagram for ESP32 cam and the pan tilt arm. We have also included the code in the in Appendix 1.7.

**Figure 1.1: Servo to ESP32 Cam Wiring Diagram**



## Camera

The camera subsystem is used for identifying victims as well as for vision when the robot is out visual site of the operator. The first iteration of the camera subsystem was a cardboard box placed on the cardboard prototype named camera. This helped us visualize where the camera sensor needed to be placed. The second iteration was using Arduino Uno and DAOKI OV7670 Camera Module this was a working version of the camera but provided some issues with quality as well as using an Arduino uno we would have to stay wired to a computer to see the video feed. One of our project specifications was we wanted a fully wireless robot. To remedy this, we created the third iteration of the camera subsystem. For the third iteration we decided to use ESP32 cam because of space limitations and Wi-Fi connectivity. The ESP32 cam allowed us to control the pan tilt arm servos with the ESP32 cam wirelessly as well as view the video feed wirelessly. We achieved this by having the ESP32 cam create a web server. It used HTML to provide an interface for controlling the arm and viewing the camera feed. Our fourth iteration was ESP32 cam mounted on pan-tilt arm. We used super glue to mount the ESP32 cam to the pan tilt arm. We also used a 9.6V battery to run this subsystem. It was important to use a battery with plenty of current. The Wi-Fi on the ESP32 cam uses a lot of current and if its underpowered the Wi-Fi will cut in and out. We are incredibly happy with how this subsystem operates. We have also included the code in the in Appendix 1.7.

## Lighting

When designing our lighting subsystem, we initially planned to use ARGB LEDs so we could use a microcontroller to program various lighting sequences. As we moved forward with the design, we realized we were running out of space on out microcontroller, and we could achieve our goal with a much more simplistic design. We decided to use more of an analog circuit design with just wires, resistors, LEDs, a 9V battery, and a switch. This was a cheaper and more simplistic design. Our first iteration of this circuit was just the bare wires taped to LEDs and resistors. In this iteration we burnt some LEDs do to trying various resistors and power supply voltages. The second iteration of the lighting subsystem we were able to narrow down what resistor values we would need as well as we decided to use a 9V battery for a DC voltage supply. For calculations we use 9.4V instead of 9 because that is the actual voltage read off the battery using a multimeter. This iteration was the first testable prototype of the lighting subsystem. For the third iteration we added the switch and properly soldered and covers all the connections. This allowed us to be able to control when the lights are on as well as eliminate drain on our battery. We were also able to properly cover the connections to keep from shorting out the circuit.

For the lighting subsystem we used 5mm, white, 2V, and 20mA LEDs. The team decided to wire the LEDs in parallel to reduce the total voltage drop across all resistors. When wiring in parallel the voltage drop is the same across all resistors. From our calculations using these LED values and our 9V power supply. We were able to calculate the proper resistor value for this circuit is 330 Ohms. To decide the value of the resistors we would need for the lighting system we used this equation:

Voltage drops for each LED: 9.4V - 3V = 6.4V

Resistor value: 6.4V / 0.02A = 330 Ohms

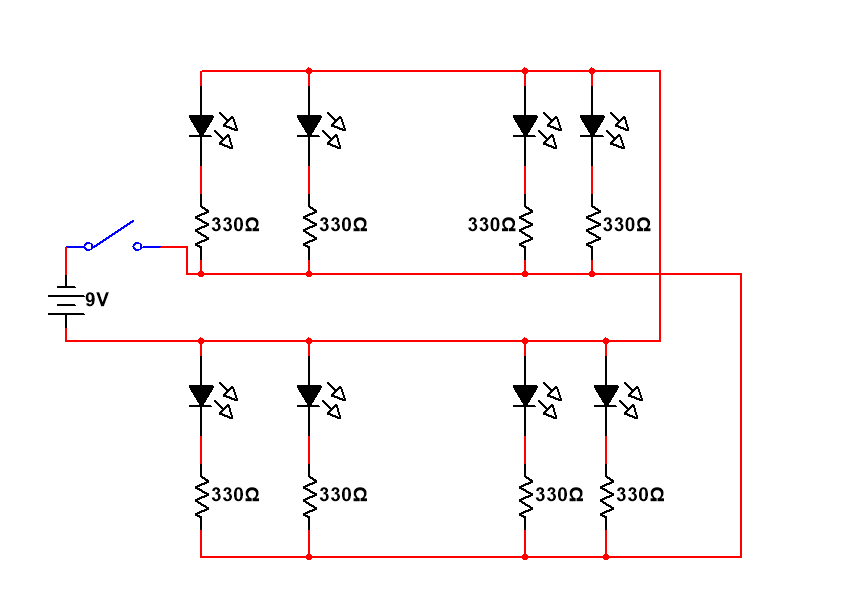
Resistor power rating: 0.02A2 \* 330 Ohms = 0.132W

Total current = 0.02A \* 8 LEDs = 0.16A

Battery Life = 500mAh / 160mAh = ~ 3.12 hours

From the above calculations you can see we can run this subsystem on a brand new 9V battery for approximately 3.12 hours. Below I have included a wiring schematic for the lighting schematic. Overall, we are happy with our current lighting system. We feel it meets the need of the design and specifications for the robot. Although they can always be improved upon.

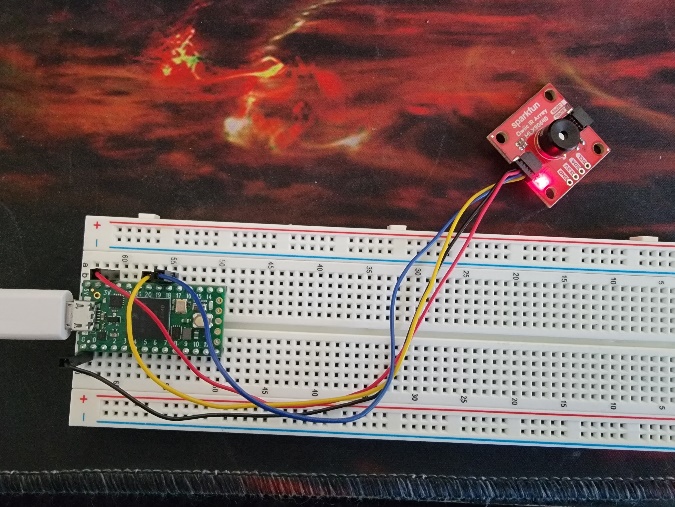
**Figure 1.2: LED Lighting Subsystem Wiring Diagram**



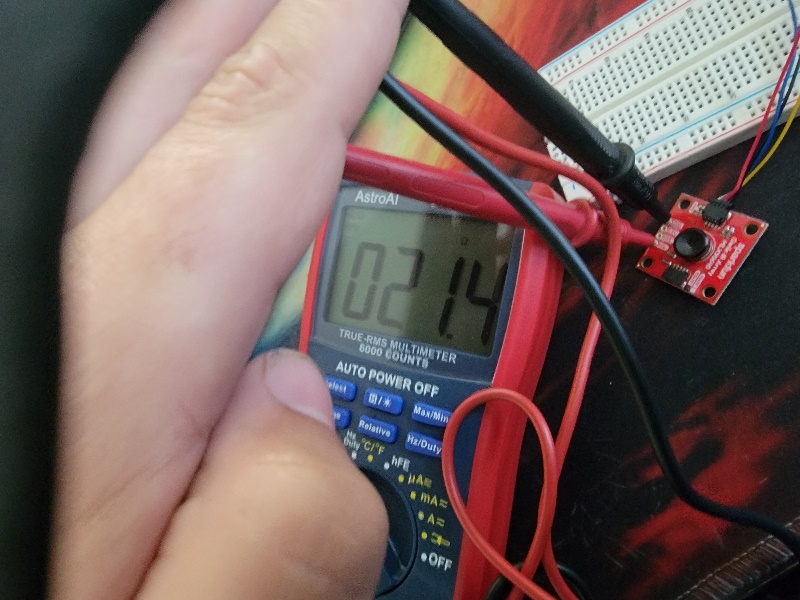
## Thermal sensor

The thermal sensor subsystem was one of the biggest disappointments for the project. In the beginning this was one of our focal points for the project. We started with the MLX94640 sensor on Sparkfun breakout board. In the beginning we could not get the sensor to connect via I2C connection. We spent over 2 weeks troubleshooting this sensor and why it would not connect. We searched all relevant forums related to this issue. To our surprise this is a verry common issue. After spending a couple hours on the phone with customer service testing and troubleshooting the board we were able to identify that the sensors metal backing was shorting out the I2C paths on the board. We where able to discover this through testing the resistance between the SDA and ground as well as testing the SCL to ground resistance. We found the resistance in each line to be about 20 Ohms. Customer service told us it was supposed to be above 10 kilo-Ohms. This proves there was a short since it was bypassing most of the resistance in the circuit. After concluding that we had received a dead board and spent all this time troubleshooting it. We also realized the replacement would not be coming in time to have the sensor implemented into the project. So, at this point we changed the focus of the project to be more of a utility robot to help firefighters. At this point we also decided to scrap the idea of using a thermal sensor since we could not afford the money or time to order a new one from a different company. Below I have included pictures of the Multimeter tests as well as how we wired the sensor to the microcontroller.

**Figure 1.3: Thermal Sensor Subsystem Wiring Diagram**



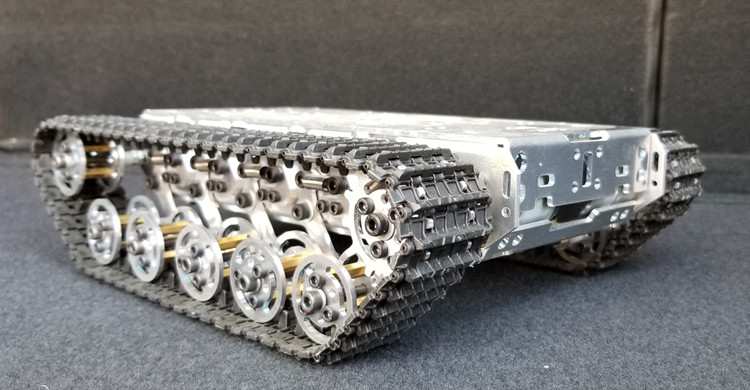
**Figure 1.4: Thermal Sensor Troubleshooting**



## Frame

The frame assembly for this robot is something be bought prefabricated. The reason we decided to go with a prefabricated frame was because we wanted to focus on the electrical subsystems and not on the mechanical side of engineering tracks a frame and motors. This kit was ordered from China and took almost four weeks to get here. This caused us to focus on our subsystems in the meantime. This robot tank frame kit came with a dual layer flat stales steal frame. It came with 2 sets of tracks with plastic track tread. The tracks have full suspension as well as can be adapted to use all metal tracks for future iterations of the prototype. The kit came with two 6V-12V 300RPM DC motors. The kit also included all bolts, nuts, washers, and spacers necessary to complete the kit. One thing we found interesting with this kit is it came with no instructions. So, we had to assemble the kit based on the photos on the website. Below I have included an image of the prefabricated frame. We are incredibly happy with the kit we got for this project. It has done everything we needed as well as has had plenty of space for all our subsystems.

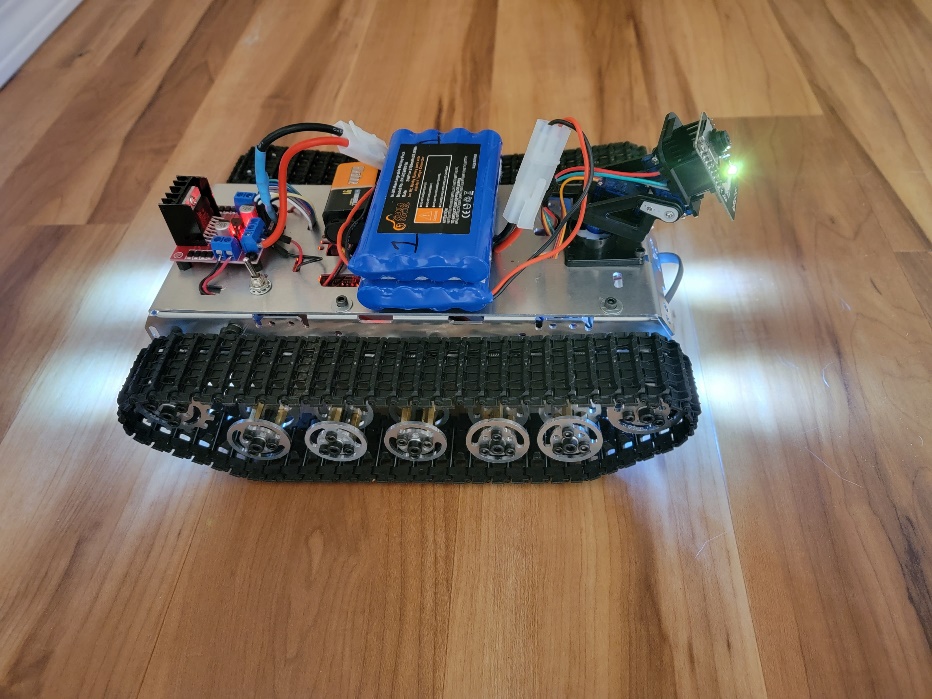
**Figure 1.5: T-300 Tank Frame**



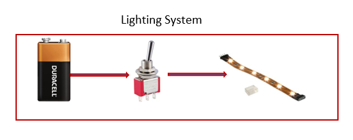
## Final Prototype

Our final prototype is the integration and compilation of all the subsystems mentioned above. The final prototype can be driven and controlled wirelessly as well as all data and camera feeds are also able to be viewed wirelessly. We feel we have achieved the goals we set out to reach at the beginning of the term. One of the complications we ran into while mounting and wiring all the subsystems was space. We had many different iterations of the layout. We also have a mess of wires everywhere in the inside of the robot. If we had more time we would properly run and mount all the wires, so it is easy to see where everything is wired to as well as troubleshoot. Currently you would probably have to unwire many components to track down a problem. Overall, all of the systems work nominally and have a decent amount of runtime. Below we have included pictures of the integrated subsystems diagram for the final prototype. We have also included a final picture of the prototype.

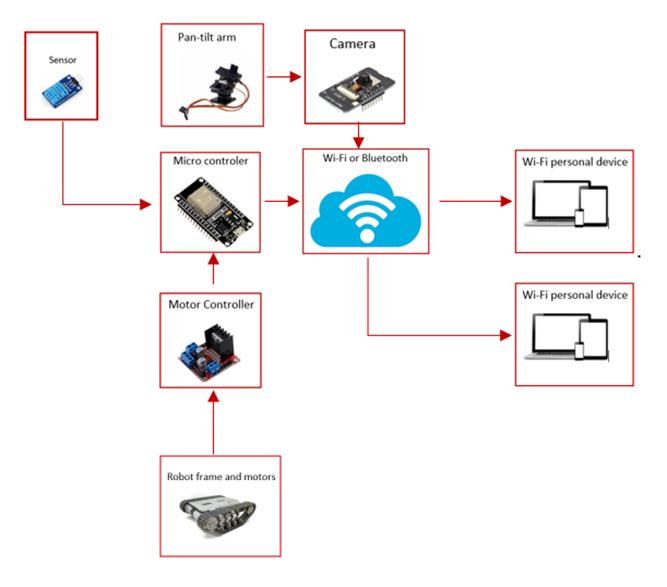
**Figure 1.6: Final Prototype**

****

**Figure 1.7: Lighting Subsystem**



**Figure 1.8: Final Prototype integrated Subsystems**



# Lessons Learned

Now we will dive into the lessons we learned as a team and individuals during the creation of this project and prototype. As individuals and as a team we feel we have learned copious new skills, techniques, and lessons throughout the process of working on this project. To list some of the lesson we have learned as a team. We learned about being extremely specific with our scope. It was easy to let the scope creep throughout the term working on this project. We also learned to be flexible as issues arise for us it was supply chain issues but we where able to manage them well. As far as specific skills the team learned: how to code an ESP32 to create a wireless server then use HTML to create a useable interface for data or controls. We also learned how to connect multiple ESP32s to one wireless server to work on the same network. The team learned about the timing issues with the ESP32 and in general how to operate the ESP32. The Team learned a lot about coding and connecting microcontrollers. Some of the more basic skills we learned was soldering, wiring components, mounting hardware, planning, working as a team, communication as a team, and in general presenting our ideas. This project was extremely educational and helpful for our careers.

# Unfinished and Future Steps

In this section of the report, we will be discussing unfinished and future steps moving forward. If we had more time and resources to work on this project, there are many features we would upgrade as well as implement to make this robot the most helpful it could be for firefighters.

One of the biggest features we wanted to implement but where not able to was the thermal sensor. We where unable to implement the thermal sensor because the original board we where sent was shorting out and it took us multiple weeks of troubleshooting with customer service to figure out it was a broken board and not user error. Once we figured out it was a broken board and initiated the replacement process it was to late in the term to receive and implement the sensor. This sensor would have allowed the robot to take directional temperature readings in an array and interpolate those readings into a heatmap for the user to see. This would have helped in hotspot recognition in an active fire scene.

If we had more time to work on the robot, we would have added bright paint and labeling. This would be helpful for victims to be able to identify the robot as a working bot for the fire department. This would allow the bot to be able to lead victims to safety. It would also allow for easier retrieval of the bot if it is lost or gets disabled in the line of duty.

After finishing the project to this point, we realized one system that would be greatly beneficial to firefighters and victims is to implement a communication system. This would be a two-way system with a speaker and microphone. This would allow firefighters or medical workers to speak with victims in need. This could be used in many cases such as trapped victims, confined spaces, medical emergencies, visually impaired victims, scared victims, and victims that are lost. This would add a whole new element of usefulness to the design, and we would love to implement something like this if we had more time.

As this project developed from a hotspot finding robot to a more general-purpose robot for firefighters. We moved away from developing the heat shielding for thew robot. This was due to time constraints as well we felt this is something that could be easily designed later if needed. If we had more time and resources, we would develop a modular heat shielding system to be used specifically on scenes of fires. Heat shielding would keep the electrical components and the sensors safe from high temperatures. This would not make the robot fireproof but allow it to operate in much higher temperature environments.

Create an app to control and monitor everything from one screen would be of huge benefit for our robot. Currently we must use 2 different devices that can connect to Wi-Fi to control and monitor the robot. This makes operating the robot more cumbersome and less efficient than it could be. If we could have it all be monitored and operated by 1 device with a seamless app it would make out robot much more useful and marketable.

An upgrade we would like to make would be to the lighting system. This would allow for various lighting based on the conditions on the scene of an incident. It would also allow for easy identification of the robot by victims, bystanders, and firefighters.

Another upgrade we would like to make would be the DHT-11 sensor and the ESP32 camera. Currently we are using a very cheep temperature sensor if we would like to be more precise with our temperature readings, we would need to upgrade this sensor currently it is precise to plus or minus 1 degree. If we used an upgraded sensor, we could make that precision to be closer to 0.1 or even 0.001 degrees. We would also like to upgrade the ESP32 camera. The camera module we used

The final upgrade we would like to make if we had more time and resources would be upgraded motors and tracks. Currently our bot operates off of plastic tracks this would not be viable in a active fire situation we would like to upgrade the tracks to metal making the more fire resistant as well as adding weight to the bot to give better traction in slippery conditions. We would also need to upgrade the motors at this point since the added weight would make it difficult for the current motors to work effectively. Also upgraded motors would give us more torque and higher speeds.

Overall, we are verry happy with the robot we built for the project. Even though there are a lot of additions and upgrades we would like to make if we had more time. We still think we have accomplished the goal we set out to achieve for the project.

# Conclusion

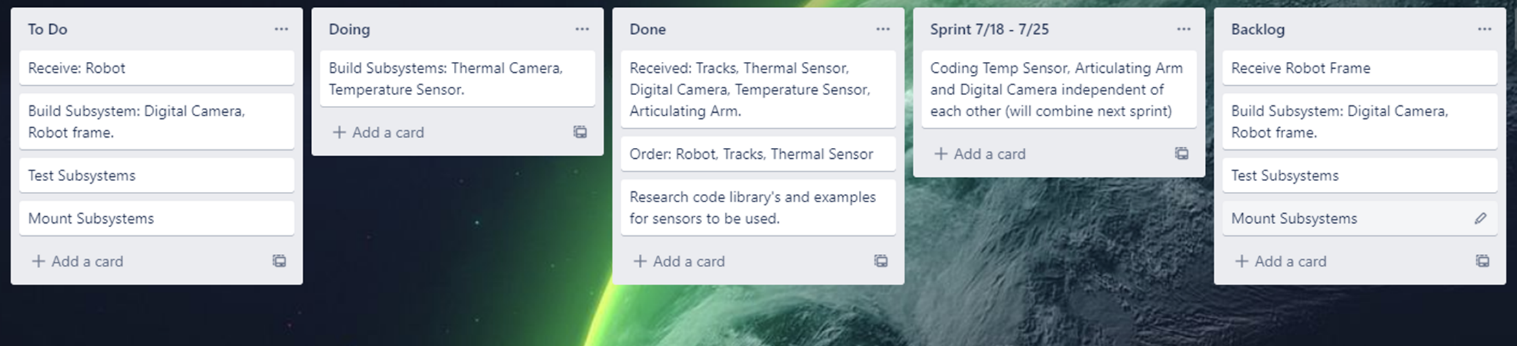
In conclusion this project was an incredibly fun and educational way for us to spend out summer working on. We learned a lot of new skills, techniques, and lessons throughout the process of working on this project. We are immensely proud of what we created even though we feel there is a lot still that could be upgraded and implemented. We achieved most of the goals and tasks we set out to achieve. Even though we were not able to implement the thermal sensor we learned a lot about the ever-evolving landscape of working on a project or prototype. This class and project and prepared us for future class projects and our capstone. We wish we had more time to work on the project. Overall, we had a lot of fun learned a lot and are happy with what we created.

# Appendix

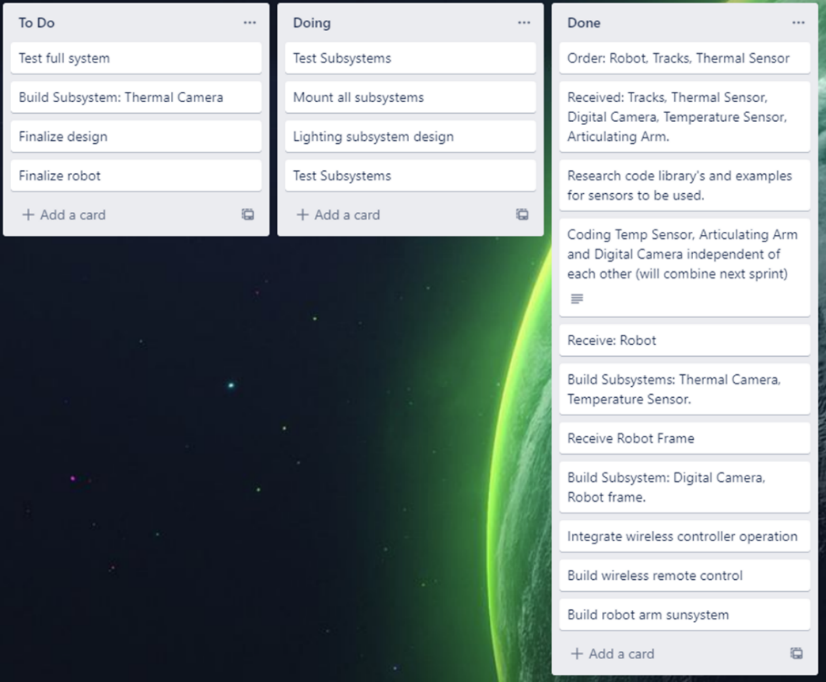
**Appendix 1.0: Components List**

|  |  |
| --- | --- |
|  | **Components List** |
| **Quantity** | **Name** |
| 1 | Robot tank frame |
| 1 | Pan and tilt arm assumably |
| 2 | ESP32 |
| 1 | ESP32 cam |
| 1 | Standard 9V battery |
| 2 | 9.6V batteries rechargeable 2100 mAh |
| 1 | Rechargeable USB battery block 10000 mAh |
| 1 | L298N motor driver board |
| 1 | Arduino power module |
| 1 | DHT11 temperature and humidity sensor |
| 1 | Phone/laptop/Wi-Fi device |
| 8 | 5mm 3V 20mA White LEDs |
| 1 | Switch |
| 1 | Breadboard |
| 1 | Resistors |
| 1 | Wires and jumper wires |
| 1 | USB cable |
| 1 | Various bolts, nuts, spacers, and washers |

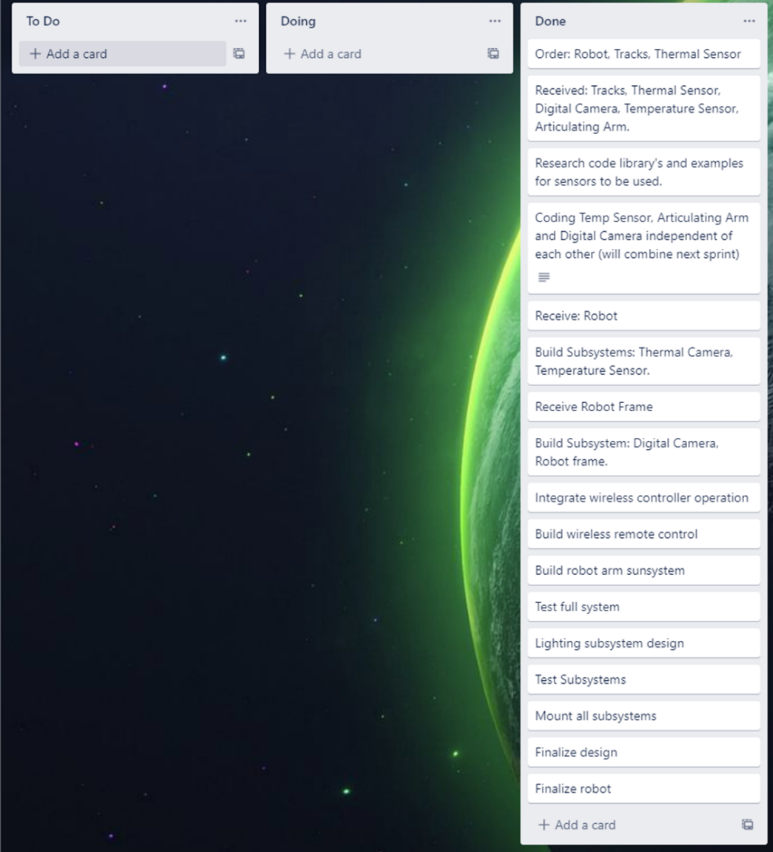
**Appendix 1.1: Sprint 1 Trello Board**



**Appendix 1.2: Sprint 2 Trello Board**



**Appendix 1.3: Sprint 3 Trello Board**



**Appendix 1.5: Code for Robot Control Subsystem**

//define librarys

#include <Arduino.h>

#ifdef ESP32

#include <WiFi.h>

#include <AsyncTCP.h>

#elif defined(ESP8266)

#include <ESP8266WiFi.h>

#include <ESPAsyncTCP.h>

#endif

#include <ESPAsyncWebServer.h>

#include <iostream>

#include <sstream>

//set motor pins

struct MOTOR\_PINS

{

int pinEn;

int pinIN1;

int pinIN2;

};

std::vector<MOTOR\_PINS> motorPins =

{

{22, 16, 17}, //RIGHT\_MOTOR Pins (EnA, IN1, IN2)

{23, 18, 19}, //LEFT\_MOTOR Pins (EnB, IN3, IN4)

};

//define move varibales

#define UP 1

#define DOWN 2

#define LEFT 3

#define RIGHT 4

#define STOP 0

#define RIGHT\_MOTOR 0

#define LEFT\_MOTOR 1

#define FORWARD 1

#define BACKWARD -1

//define pwm specs

const int PWMFreq = 1000; /\* 1 KHz \*/

const int PWMResolution = 8;

const int PWMSpeedChannel = 4;

//define wifi ssid and password

const char\* ssid = "ElectricVision";

const char\* password = "12345678";

AsyncWebServer server(80);

AsyncWebSocket wsCarInput("/CarInput");

//define HTML parameters for interface

const char\* htmlHomePage PROGMEM = R"HTMLHOMEPAGE(

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">

<style>

.arrows {

font-size:40px;

color:red;

}

td.button {

background-color:black;

border-radius:25%;

box-shadow: 5px 5px #888888;

}

td.button:active {

transform: translate(5px,5px);

box-shadow: none;

}

.noselect {

-webkit-touch-callout: none; /\* iOS Safari \*/

-webkit-user-select: none; /\* Safari \*/

-khtml-user-select: none; /\* Konqueror HTML \*/

-moz-user-select: none; /\* Firefox \*/

-ms-user-select: none; /\* Internet Explorer/Edge \*/

user-select: none; /\* Non-prefixed version, currently

supported by Chrome and Opera \*/

}

//define slider perameters for html interface

.slidecontainer {

width: 100%;

}

.slider {

-webkit-appearance: none;

width: 100%;

height: 20px;

border-radius: 5px;

background: #d3d3d3;

outline: none;

opacity: 0.7;

-webkit-transition: .2s;

transition: opacity .2s;

}

.slider:hover {

opacity: 1;

}

.slider::-webkit-slider-thumb {

-webkit-appearance: none;

appearance: none;

width: 40px;

height: 40px;

border-radius: 50%;

background: red;

cursor: pointer;

}

.slider::-moz-range-thumb {

width: 40px;

height: 40px;

border-radius: 50%;

background: red;

cursor: pointer;

}

</style>

</head>

//define css styles in html

<body class="noselect" align="center" style="background-color:white">

<h1 style="color: teal;text-align:center;">Electric Vision</h1>

<h2 style="color: teal;text-align:center;">WiFi Robot Control</h2>

<table id="mainTable" style="width:400px;margin:auto;table-layout:fixed" CELLSPACING=10>

<tr>

<td></td>

<td class="button" ontouchstart='sendButtonInput("MoveCar","1")' ontouchend='sendButtonInput("MoveCar","0")'><span class="arrows" >&#8679;</span></td>

<td></td>

</tr>

<tr>

<td class="button" ontouchstart='sendButtonInput("MoveCar","3")' ontouchend='sendButtonInput("MoveCar","0")'><span class="arrows" >&#8678;</span></td>

<td class="button"></td>

<td class="button" ontouchstart='sendButtonInput("MoveCar","4")' ontouchend='sendButtonInput("MoveCar","0")'><span class="arrows" >&#8680;</span></td>

</tr>

<tr>

<td></td>

<td class="button" ontouchstart='sendButtonInput("MoveCar","2")' ontouchend='sendButtonInput("MoveCar","0")'><span class="arrows" >&#8681;</span></td>

<td></td>

</tr>

<tr/><tr/>

<tr/><tr/>

<tr/><tr/>

<tr>

<td style="text-align:left;font-size:25px"><b>Speed:</b></td>

<td colspan=2>

<div class="slidecontainer">

<input type="range" min="0" max="255" value="150" class="slider" id="Speed" oninput='sendButtonInput("Speed",value)'>

</div>

</td>

</tr>

</table>

<script>

var webSocketCarInputUrl = "ws:\/\/" + window.location.hostname + "/CarInput";

var websocketCarInput;

//define websocket specifications

function initCarInputWebSocket()

{

websocketCarInput = new WebSocket(webSocketCarInputUrl);

websocketCarInput.onopen = function(event)

{

var speedButton = document.getElementById("Speed");

sendButtonInput("Speed", speedButton.value);

};

websocketCarInput.onclose = function(event){setTimeout(initCarInputWebSocket, 2000);};

websocketCarInput.onmessage = function(event){};

}

function sendButtonInput(key, value)

{

var data = key + "," + value;

websocketCarInput.send(data);

}

window.onload = initCarInputWebSocket;

document.getElementById("mainTable").addEventListener("touchend", function(event){

event.preventDefault()

});

</script>

</body>

</html>

)HTMLHOMEPAGE";

//define motor movement and pins

void rotateMotor(int motorNumber, int motorDirection)

{

if (motorDirection == FORWARD)

{

digitalWrite(motorPins[motorNumber].pinIN1, HIGH);

digitalWrite(motorPins[motorNumber].pinIN2, LOW);

}

else if (motorDirection == BACKWARD)

{

digitalWrite(motorPins[motorNumber].pinIN1, LOW);

digitalWrite(motorPins[motorNumber].pinIN2, HIGH);

}

else

{

digitalWrite(motorPins[motorNumber].pinIN1, LOW);

digitalWrite(motorPins[motorNumber].pinIN2, LOW);

}

}

void moveCar(int inputValue)

{

Serial.printf("Got value as %d\n", inputValue);

switch(inputValue)

{

case UP:

rotateMotor(RIGHT\_MOTOR, FORWARD);

rotateMotor(LEFT\_MOTOR, FORWARD);

break;

case DOWN:

rotateMotor(RIGHT\_MOTOR, BACKWARD);

rotateMotor(LEFT\_MOTOR, BACKWARD);

break;

case LEFT:

rotateMotor(RIGHT\_MOTOR, FORWARD);

rotateMotor(LEFT\_MOTOR, BACKWARD);

break;

case RIGHT:

rotateMotor(RIGHT\_MOTOR, BACKWARD);

rotateMotor(LEFT\_MOTOR, FORWARD);

break;

case STOP:

rotateMotor(RIGHT\_MOTOR, STOP);

rotateMotor(LEFT\_MOTOR, STOP);

break;

default:

rotateMotor(RIGHT\_MOTOR, STOP);

rotateMotor(LEFT\_MOTOR, STOP);

break;

}

}

void handleRoot(AsyncWebServerRequest \*request)

{

request->send\_P(200, "text/html", htmlHomePage);

}

void handleNotFound(AsyncWebServerRequest \*request)

{

request->send(404, "text/plain", "File Not Found");

}

void onCarInputWebSocketEvent(AsyncWebSocket \*server,

AsyncWebSocketClient \*client,

AwsEventType type,

void \*arg,

uint8\_t \*data,

size\_t len)

{

switch (type)

{

case WS\_EVT\_CONNECT:

Serial.printf("WebSocket client #%u connected from %s\n", client->id(), client->remoteIP().toString().c\_str());

break;

case WS\_EVT\_DISCONNECT:

Serial.printf("WebSocket client #%u disconnected\n", client->id());

moveCar(STOP);

break;

case WS\_EVT\_DATA:

AwsFrameInfo \*info;

info = (AwsFrameInfo\*)arg;

if (info->final && info->index == 0 && info->len == len && info->opcode == WS\_TEXT)

{

std::string myData = "";

myData.assign((char \*)data, len);

std::istringstream ss(myData);

std::string key, value;

std::getline(ss, key, ',');

std::getline(ss, value, ',');

Serial.printf("Key [%s] Value[%s]\n", key.c\_str(), value.c\_str());

int valueInt = atoi(value.c\_str());

if (key == "MoveCar")

{

moveCar(valueInt);

}

else if (key == "Speed")

{

ledcWrite(PWMSpeedChannel, valueInt);

}

}

break;

case WS\_EVT\_PONG:

case WS\_EVT\_ERROR:

break;

default:

break;

}

}

//define pin modes

void setUpPinModes()

{

//Set up PWM

ledcSetup(PWMSpeedChannel, PWMFreq, PWMResolution);

for (int i = 0; i < motorPins.size(); i++)

{

pinMode(motorPins[i].pinEn, OUTPUT);

pinMode(motorPins[i].pinIN1, OUTPUT);

pinMode(motorPins[i].pinIN2, OUTPUT);

/\* Attach the PWM Channel to the motor enb Pin \*/

ledcAttachPin(motorPins[i].pinEn, PWMSpeedChannel);

}

moveCar(STOP);

}

//define pin modes and strat serial communication

void setup(void)

{

setUpPinModes();

Serial.begin(115200);

//wifi connection

WiFi.softAP(ssid, password);

IPAddress IP = WiFi.softAPIP();

Serial.print("AP IP address: ");

Serial.println(IP);

server.on("/", HTTP\_GET, handleRoot);

server.onNotFound(handleNotFound);

wsCarInput.onEvent(onCarInputWebSocketEvent);

server.addHandler(&wsCarInput);

server.begin();

Serial.println("HTTP server started");

}

void loop()

{

wsCarInput.cleanupClients();

}

**Appendix 1.6: Code for Temperature and Humidity Subsystem**

//Include WIFI WebServer and DHT librarys

#include <WiFi.h>

#include <WebServer.h>

#include "DHT.h"

// define sensor

#define DHTTYPE DHT11 // DHT 11

// Put ssid and password of network to connect to

const char\* ssid = "ElectricVisionCam"; // Enter SSID here

const char\* password = "12345678"; //Enter Password here

WebServer server(80);

// DHT Sensor and pin its connected to

uint8\_t DHTPin = 25;

// Initialize sensor.

DHT dht(DHTPin, DHTTYPE);

//Create variables

float Temperature;

float Humidity;

void setup() {

//Set up serial connection and delay

Serial.begin(115200);

delay(100);

//Define pin mode

pinMode(DHTPin, INPUT);

//Start sensor reading

dht.begin();

//Print connection info

Serial.println("Connecting to ");

Serial.println(ssid);

//connect to your local wi-fi network

WiFi.begin(ssid, password);

//check wi-fi is connected to wi-fi network

while (WiFi.status() != WL\_CONNECTED) {

delay(1000);

Serial.print(".");

}

//Print connection info and ip info

Serial.println("");

Serial.println("WiFi connected..!");

Serial.print("Got IP: "); Serial.println(WiFi.localIP());

server.on("/", handle\_OnConnect);

server.onNotFound(handle\_NotFound);

server.begin();

Serial.println("HTTP server started");

}

void loop() ;

server.handleClient();

}

void handle\_OnConnect() {

//Gets the values of the temperature

Temperature = dht.readTemperature();

//Gets the values of the humidity

Humidity = dht.readHumidity();

//Send to HTML temperature and humidity info

server.send(200, "text/html", SendHTML(Temperature,Humidity));

}

//Error check for HTML connectionm

void handle\_NotFound(){

server.send(404, "text/plain", "Not found");

}

//HTML code for interface

String SendHTML(float Temperaturestat,float Humiditystat){

String ptr = "<!DOCTYPE html> <html>\n";

ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr +="<title>Electric Vision Bot</title>\n";

ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";

ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;}\n";

ptr +="p {font-size: 24px;color: #444444;margin-bottom: 10px;}\n";

ptr +="</style>\n";

ptr +="</head>\n";

ptr +="<body>\n";

ptr +="<div id=\"webpage\">\n";

ptr +="<h1>Electric Vision Robot</h1>\n";

//Print temperature in celsius

ptr +="<p>Temperature in Celsius: <p>";

ptr +=(int)Temperaturestat \* 0.86;

ptr +="&deg;C</p>";

//Print temperature in fahrenheit

ptr +="<p>Temperature in Fahrenheit: <p>";

ptr +=(((int)Temperaturestat \* 0.86) \* 1.8) + 32;

ptr +="&deg;F</p>";

//Print humidity in percentage

ptr +="<p>Humidity Percentage: <p>";

ptr +=(int)Humiditystat;

ptr +="%</p>";

//HTML interface code

ptr +="</div>\n";

ptr +="</body>\n";

ptr +="</html>\n";

return ptr;}

**Appendix 1.7: Code for Pan Tilt Arm and ESP32 Cam Subsystem**

//Include needed librarys

#include "esp\_camera.h"

#include <Arduino.h>

#include <WiFi.h>

#include <AsyncTCP.h>

#include <ESPAsyncWebServer.h>

#include <iostream>

#include <sstream>

#include <ESP32Servo.h>

//Need to create 2 dummy servos.

//So the ESP32Servo library does not interfere with pwm channel and timer used by esp32 camera.

#define DUMMY\_SERVO1\_PIN 12

#define DUMMY\_SERVO2\_PIN 13

//define pins used on esp32 cam

#define PAN\_PIN 14

#define TILT\_PIN 15

//define servos

Servo dummyServo1;

Servo dummyServo2;

Servo panServo;

Servo tiltServo;

//define camera related constants

#define PWDN\_GPIO\_NUM 32

#define RESET\_GPIO\_NUM -1

#define XCLK\_GPIO\_NUM 0

#define SIOD\_GPIO\_NUM 26

#define SIOC\_GPIO\_NUM 27

#define Y9\_GPIO\_NUM 35

#define Y8\_GPIO\_NUM 34

#define Y7\_GPIO\_NUM 39

#define Y6\_GPIO\_NUM 36

#define Y5\_GPIO\_NUM 21

#define Y4\_GPIO\_NUM 19

#define Y3\_GPIO\_NUM 18

#define Y2\_GPIO\_NUM 5

#define VSYNC\_GPIO\_NUM 25

#define HREF\_GPIO\_NUM 23

#define PCLK\_GPIO\_NUM 22

//defince wifi ssid and password

const char\* ssid = "ElectricVisionCam";

const char\* password = "12345678";

//websocket info

AsyncWebServer server(80);

AsyncWebSocket wsCamera("/Camera");

AsyncWebSocket wsServoInput("/ServoInput");

uint32\_t cameraClientId = 0;

#define LIGHT\_PIN 4

const int PWMLightChannel = 4;

//HTML setup

const char\* htmlHomePage PROGMEM = R"HTMLHOMEPAGE(

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">

<style>

.noselect {

-webkit-touch-callout: none; /\* iOS Safari \*/

-webkit-user-select: none; /\* Safari \*/

-khtml-user-select: none; /\* Konqueror HTML \*/

-moz-user-select: none; /\* Firefox \*/

-ms-user-select: none; /\* Internet Explorer/Edge \*/

user-select: none; /\* Non-prefixed version, currently

supported by Chrome and Opera \*/

}

//HTML slider setup

.slidecontainer {

width: 100%;

}

.slider {

-webkit-appearance: none;

width: 100%;

height: 20px;

border-radius: 5px;

background: #d3d3d3;

outline: none;

opacity: 0.7;

-webkit-transition: .2s;

transition: opacity .2s;

}

.slider:hover {

opacity: 1;

}

.slider::-webkit-slider-thumb {

-webkit-appearance: none;

appearance: none;

width: 40px;

height: 40px;

border-radius: 50%;

background: red;

cursor: pointer;

}

.slider::-moz-range-thumb {

width: 40px;

height: 40px;

border-radius: 50%;

background: red;

cursor: pointer;

}

</style>

//HTML interface

</head>

<body class="noselect" align="center" style="background-color:white">

<!--h2 style="color: teal;text-align:center;">Wi-Fi Camera &#128663; Control</h2-->

<table id="mainTable" style="width:400px;margin:auto;table-layout:fixed" CELLSPACING=10>

<tr>

<img id="cameraImage" src="" style="width:400px;height:250px"></td>

</tr>

<tr/><tr/>

<tr>

<td style="text-align:left"><b>Pan:</b></td>

<td colspan=2>

<div class="slidecontainer">

<input type="range" min="0" max="180" value="90" class="slider" id="Pan" oninput='sendButtonInput("Pan",value)'>

</div>

</td>

</tr>

<tr/><tr/>

<tr>

<td style="text-align:left"><b>Tilt:</b></td>

<td colspan=2>

<div class="slidecontainer">

<input type="range" min="0" max="180" value="90" class="slider" id="Tilt" oninput='sendButtonInput("Tilt",value)'>

</div>

</td>

</tr>

<tr/><tr/>

<tr>

<td style="text-align:left"><b>Light:</b></td>

<td colspan=2>

<div class="slidecontainer">

<input type="range" min="0" max="255" value="0" class="slider" id="Light" oninput='sendButtonInput("Light",value)'>

</div>

</td>

</tr>

</table>

//Websocket script

<script>

var webSocketCameraUrl = "ws:\/\/" + window.location.hostname + "/Camera";

var webSocketServoInputUrl = "ws:\/\/" + window.location.hostname + "/ServoInput";

var websocketCamera;

var websocketServoInput;

function initCameraWebSocket()

{

websocketCamera = new WebSocket(webSocketCameraUrl);

websocketCamera.binaryType = 'blob';

websocketCamera.onopen = function(event){};

websocketCamera.onclose = function(event){setTimeout(initCameraWebSocket, 2000);};

websocketCamera.onmessage = function(event)

{

var imageId = document.getElementById("cameraImage");

imageId.src = URL.createObjectURL(event.data);

};

}

function initServoInputWebSocket()

{

websocketServoInput = new WebSocket(webSocketServoInputUrl);

websocketServoInput.onopen = function(event)

{

var panButton = document.getElementById("Pan");

sendButtonInput("Pan", panButton.value);

var tiltButton = document.getElementById("Tilt");

sendButtonInput("Tilt", tiltButton.value);

var lightButton = document.getElementById("Light");

sendButtonInput("Light", lightButton.value);

};

websocketServoInput.onclose = function(event){setTimeout(initServoInputWebSocket, 2000);};

websocketServoInput.onmessage = function(event){};

}

function initWebSocket()

{

initCameraWebSocket ();

initServoInputWebSocket();

}

function sendButtonInput(key, value)

{

var data = key + "," + value;

websocketServoInput.send(data);

}

window.onload = initWebSocket;

document.getElementById("mainTable").addEventListener("touchend", function(event){

event.preventDefault()

});

</script>

</body>

</html>

)HTMLHOMEPAGE";

void handleRoot(AsyncWebServerRequest \*request)

{

request->send\_P(200, "text/html", htmlHomePage);

}

void handleNotFound(AsyncWebServerRequest \*request)

{

request->send(404, "text/plain", "File Not Found");

}

void onServoInputWebSocketEvent(AsyncWebSocket \*server,

AsyncWebSocketClient \*client,

AwsEventType type,

void \*arg,

uint8\_t \*data,

size\_t len)

{

switch (type)

{

case WS\_EVT\_CONNECT:

Serial.printf("WebSocket client #%u connected from %s\n", client->id(), client->remoteIP().toString().c\_str());

break;

case WS\_EVT\_DISCONNECT:

Serial.printf("WebSocket client #%u disconnected\n", client->id());

panServo.write(90);

tiltServo.write(90);

ledcWrite(PWMLightChannel, 0);

break;

case WS\_EVT\_DATA:

AwsFrameInfo \*info;

info = (AwsFrameInfo\*)arg;

if (info->final && info->index == 0 && info->len == len && info->opcode == WS\_TEXT)

{

std::string myData = "";

myData.assign((char \*)data, len);

Serial.printf("Key,Value = [%s]\n", myData.c\_str());

std::istringstream ss(myData);

std::string key, value;

std::getline(ss, key, ',');

std::getline(ss, value, ',');

if ( value != "" )

{

int valueInt = atoi(value.c\_str());

if (key == "Pan")

{

panServo.write(valueInt);

}

else if (key == "Tilt")

{

tiltServo.write(valueInt);

}

else if (key == "Light")

{

ledcWrite(PWMLightChannel, valueInt);

}

}

}

break;

case WS\_EVT\_PONG:

case WS\_EVT\_ERROR:

break;

default:

break;

}

}

void onCameraWebSocketEvent(AsyncWebSocket \*server,

AsyncWebSocketClient \*client,

AwsEventType type,

void \*arg,

uint8\_t \*data,

size\_t len)

{

switch (type)

{

case WS\_EVT\_CONNECT:

Serial.printf("WebSocket client #%u connected from %s\n", client->id(), client->remoteIP().toString().c\_str());

cameraClientId = client->id();

break;

case WS\_EVT\_DISCONNECT:

Serial.printf("WebSocket client #%u disconnected\n", client->id());

cameraClientId = 0;

break;

case WS\_EVT\_DATA:

break;

case WS\_EVT\_PONG:

case WS\_EVT\_ERROR:

break;

default:

break;

}

}

void setupCamera()

{

camera\_config\_t config;

config.ledc\_channel = LEDC\_CHANNEL\_0;

config.ledc\_timer = LEDC\_TIMER\_0;

config.pin\_d0 = Y2\_GPIO\_NUM;

config.pin\_d1 = Y3\_GPIO\_NUM;

config.pin\_d2 = Y4\_GPIO\_NUM;

config.pin\_d3 = Y5\_GPIO\_NUM;

config.pin\_d4 = Y6\_GPIO\_NUM;

config.pin\_d5 = Y7\_GPIO\_NUM;

config.pin\_d6 = Y8\_GPIO\_NUM;

config.pin\_d7 = Y9\_GPIO\_NUM;

config.pin\_xclk = XCLK\_GPIO\_NUM;

config.pin\_pclk = PCLK\_GPIO\_NUM;

config.pin\_vsync = VSYNC\_GPIO\_NUM;

config.pin\_href = HREF\_GPIO\_NUM;

config.pin\_sscb\_sda = SIOD\_GPIO\_NUM;

config.pin\_sscb\_scl = SIOC\_GPIO\_NUM;

config.pin\_pwdn = PWDN\_GPIO\_NUM;

config.pin\_reset = RESET\_GPIO\_NUM;

config.xclk\_freq\_hz = 20000000;

config.pixel\_format = PIXFORMAT\_JPEG;

config.frame\_size = FRAMESIZE\_VGA;

config.jpeg\_quality = 10;

config.fb\_count = 1;

// camera init

esp\_err\_t err = esp\_camera\_init(&config);

if (err != ESP\_OK)

{

Serial.printf("Camera init failed with error 0x%x", err);

return;

}

if (psramFound())

{

heap\_caps\_malloc\_extmem\_enable(20000);

Serial.printf("PSRAM initialized. malloc to take memory from psram above this size");

}

}

void sendCameraPicture()

{

if (cameraClientId == 0)

{

return;

}

unsigned long startTime1 = millis();

//capture a frame

camera\_fb\_t \* fb = esp\_camera\_fb\_get();

if (!fb)

{

Serial.println("Frame buffer could not be acquired");

return;

}

unsigned long startTime2 = millis();

wsCamera.binary(cameraClientId, fb->buf, fb->len);

esp\_camera\_fb\_return(fb);

//Wait for message to be delivered

while (true)

{

AsyncWebSocketClient \* clientPointer = wsCamera.client(cameraClientId);

if (!clientPointer || !(clientPointer->queueIsFull()))

{

break;

}

delay(1);

}

unsigned long startTime3 = millis();

Serial.printf("Time taken Total: %d|%d|%d\n",startTime3 - startTime1, startTime2 - startTime1, startTime3-startTime2 );

}

void setUpPinModes()

{

//pin setup

dummyServo1.attach(DUMMY\_SERVO1\_PIN);

dummyServo2.attach(DUMMY\_SERVO2\_PIN);

panServo.attach(PAN\_PIN);

tiltServo.attach(TILT\_PIN);

//Set up flash light

ledcSetup(PWMLightChannel, 1000, 8);

pinMode(LIGHT\_PIN, OUTPUT);

ledcAttachPin(LIGHT\_PIN, PWMLightChannel);

}

void setup(void)

{

//start serial connection

setUpPinModes();

Serial.begin(115200);

//start wifi connection

WiFi.softAP(ssid, password);

IPAddress IP = WiFi.softAPIP();

Serial.print("AP IP address: ");

Serial.println(IP);

//server connection get command

server.on("/", HTTP\_GET, handleRoot);

server.onNotFound(handleNotFound);

//send camera events to server

wsCamera.onEvent(onCameraWebSocketEvent);

server.addHandler(&wsCamera);

//send servo events to server

wsServoInput.onEvent(onServoInputWebSocketEvent);

server.addHandler(&wsServoInput);

//start server

server.begin();

Serial.println("HTTP server started");

setupCamera();

}

void loop()

{

wsCamera.cleanupClients();

wsServoInput.cleanupClients();

sendCameraPicture();

//Serial.printf("SPIRam Total heap %d, SPIRam Free Heap %d\n", ESP.getPsramSize(), ESP.getFreePsram());

}