ECET365 Course Robotic Car Project

Byron Jordan

DeVry University

Author Note

Professor Toshko Tzvetkov

Contents

[Abstract 3](#_Toc44149885)

[[**Product Development Timeline**] 4](#_Toc44149886)

[[Microcontroller] 7](#_Toc44149887)

[[Visual Subsystem] 9](#_Toc44149888)

[[Motor Subsystem] 10](#_Toc44149889)

[References 21](#_Toc44149890)

Abstract

In this project, I learned to implement product development techniques to research, design, and test the build/operation of a robotic car using an ESP32S microcontroller. I explored several subsystems involving DC motors, H-bridges (motor controls), ultrasonic sensors (obstacle avoidance), and Wi-Fi interfaces (steering control).

Keywords: [DC motors, H-Bridge (DRV8833), Ultrasonic Sensors (HC-SR04), ESP32S]

[**Product Development Timeline**]

We had 8-weeks to purchase, build, and test our robotic car designs. Below is the schedule we followed to ensure that we effectively met our deadlines of the course project. Abiding by the Ghantt chart wasn’t particularly difficult because it coincided with our weekly labs.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Task | Week1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week8 |
| **1.0 Operate DC Motors** | XXXXXXX |  |  |  |  |  |  |  |
| 1.1 Develop Software | XXXXXXX |  |  |  |  |  |  |  |
| 1.2 Check H-Bridge | XXXXXXX |  |  |  |  |  |  |  |
| 1.3 Install Power Wiring | XXXXXXX |  |  |  |  |  |  |  |
| 1.4 Test Drive Motors | XXXXXXX |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| **2.0 Install and Test Visual Sensors** |  | XXXXXXX |  |  |  |  |  |  |
| 2.1 Install Power on Proto-Board |  | XXXXXXX |  |  |  |  |  |  |
| 2.2 Install Sensors on Car |  | XXXXXXX |  |  |  |  |  |  |
| 2.3 Connect Sensors to CPU Ports |  | XXXXXXX |  |  |  |  |  |  |
| 2.4 Test Visual Sensors |  | XXXXXXX |  |  |  |  |  |  |
| 2.5 Test Visual Sensor Subsystem |  | XXXXXXX |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| **3.0 Convert Requirements to Subsystems** |  |  | XXXXXXX |  |  |  |  |  |
| 3.1 Determine Requirements |  |  | XXXXXXX |  |  |  |  |  |
| 3.2 Determine Sensor Alternatives |  |  | XXXXXXX |  |  |  |  |  |
| 3.3 Determine Motor Alternatives |  |  | XXXXXXX |  |  |  |  |  |
| 3.4 Determine CPU Alternatives |  |  | XXXXXXX |  |  |  |  |  |
| 3.5 Determine Power Alternatives |  |  | XXXXXXX |  |  |  |  |  |
| 3.6 Determine System Design |  |  | XXXXXXX |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| **4.0 Build and Test the Power Subsystem** |  |  |  | XXXXXXX |  |  |  |  |
| 4.1 Set up Power Supply on Primary Elevator |  |  |  | XXXXXXX |  |  |  |  |
| 4.2 Check Power to Drive Motors |  |  |  | XXXXXXX |  |  |  |  |
| 4.3 Check Power to Visual Sensors |  |  |  | XXXXXXX |  |  |  |  |
| 4.4 Ensure that System Power is Working |  |  |  | XXXXXXX |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| **5.0 Test Interrupt Software** |  |  |  |  |  |  |  |  |
| 5.1 Load Programs in Appendix A of Construction Notes |  |  |  |  | XXXXXXX |  |  |  |
| 5.2 Observe Effects on Robot Operation |  |  |  |  | XXXXXXX |  |  |  |
|  |  |  |  |  |  |  |  |  |
| **6.0 Explore Possible Internet and Wireless Interfaces** |  |  |  |  |  | XXXXXXX |  |  |
| 6.1 Find Possible Interfaces on the Internet |  |  |  |  |  | XXXXXXX |  |  |
| 6.2 Evaluate Candidates in Terms of Price, Availability, and Capability |  |  |  |  |  | XXXXXXX |  |  |
| 6.3 Present Results to Class |  |  |  |  |  | XXXXXXX |  |  |
|  |  |  |  |  |  |  |  |  |
| 7.0 Final Integration of Subsystems |  |  |  |  |  |  | XXXXXXX |  |
| 7.1 Software Subsystem |  |  |  |  |  |  | XXXXXXX |  |
| 7.2 Drive Motor Subsystem |  |  |  |  |  |  | XXXXXXX |  |
| 7.3 Power Subsystem |  |  |  |  |  |  | XXXXXXX |  |
| 7.4 Sensor Subsystem |  |  |  |  |  |  | XXXXXXX |  |
| 7.5 Project Manual |  |  |  |  |  |  | XXXXXXX |  |
|  |  |  |  |  |  |  |  |  |
| 8.0 Demonstrate Project |  |  |  |  |  |  |  | XXXXXXX |

# [Microcontroller]

There are tons of choices when choosing a microcontroller on the market. Some more capable than others. In previous courses, we learned to design products based on the Arduino ATmega328 and ATmega2560. For ECET365, the microcontroller of choice was the ESP32S. ESP32 is a single 2.4 GHz Wi-Fi-and-Bluetooth combo chip designed with the TSMC ultra-low-power 40 nm technology. It is designed to achieve the best power and RF performance, showing robustness, versatility, and reliability in a wide variety of applications and power scenarios. It is equipped with 34 programmable GPIOs. Up to 18 channels of 12-bit SAR ADC and two 8-bit DACs. It supports motor PWM and UART Bluetooth interfaces up to 4Mbps.

A close up of a calculator

Description automatically generated

Figure Functional Block Diagram

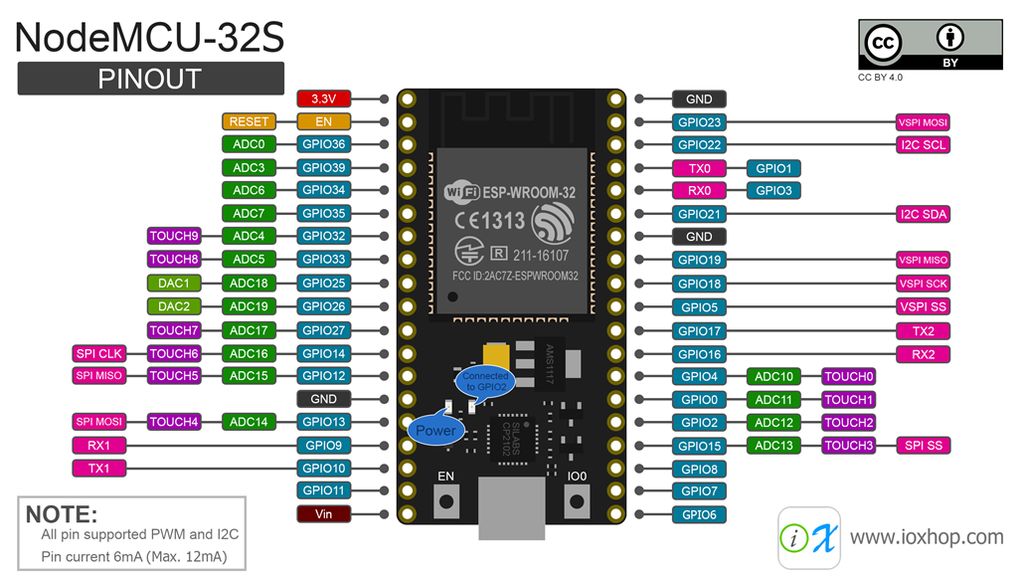


Figure NodeMCU-32S Pin Map

I ran into an issue uploading the source which required a little troubleshooting to resolve. I kept receiving an error after the code would compile. I was instructed to wait for the Arduino IDE to show a “Connecting” message. I was then supposed to push the IO0 button to boot the microcontroller. However, the button never seemed to work. I then unsoldered the button and tried to short the SMD pad with a wire. Still no luck. Assuming the microcontroller was defective, I tried one more time to short the pads of the button by making a solder bridge across the pads. Still no luck. Agreeing with Professor Tzvetkov I decided to order several more ESP32S. I then realized, there are several flavors of the microcontroller, from different manufacturers, with different layouts and pin assignments. SparkFun, in my opinion, makes the most user-friendly version. This is because the board is labeled and has a connector for a Li-PO battery and there are tutorials on how to connect and use it. I decide to continue using my replacement of the original NodeMCU-ESP32S.

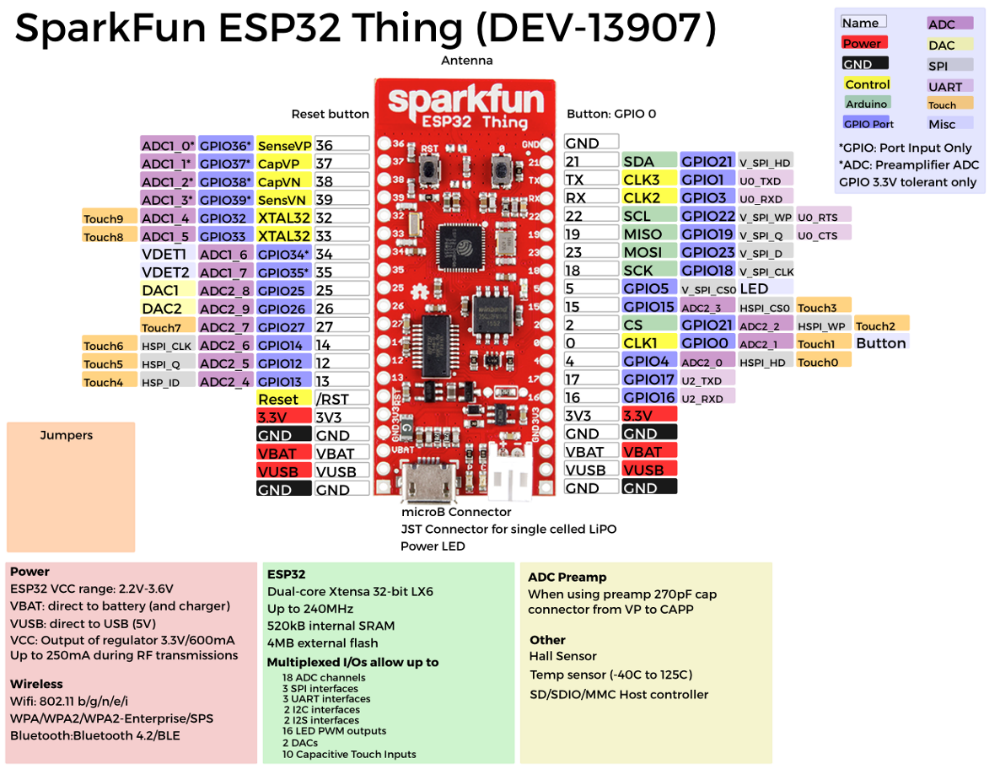
[](https://learn.sparkfun.com/tutorials/esp32-thing-hookup-guide)

Figure SparkFun ESP32S Thing Pin Map

## [Visual Subsystem]

For this project, there were several choices for a visual subsystem. We chose to use an ultrasonic distance sensor for obstacle avoidance. This sensor works by sending 8-40kHz ping signals from the trigger pin of the module. It then listens and detects whether there is a pulse signal back.

[](https://www.amazon.com/SainSmart-HC-SR04-Ranging-Detector-Distance/dp/B004U8TOE6)

Figure HC-SR04 Ultrasonic Sensor $4.95 [Amazon]

Testing this subsystem was simple. For a quick test, I used the serial monitor to view the reading from the distance sensor. I used my hand as an obstacle and was able to witness the sensor reading as far as 45 inches away. Although the specifications of the sensor say its capable of reading as far as 500cm or as close as 2cm, with the effective range being less than 15cm (approx. 5.91 inches). Embedded within our source code is a switch case interrupt.   
The ESP32S uses the switch case and the ultrasonic sensor to halt the operations of the car. Switch cases are decision making commands based on predefined conditions. In this case, the microcontroller was designed to sense danger if the car comes within 4-inches of an obstacle. In the figure below, you can see the distance readings as I block the sensor with my hand, then release.

A screenshot of a cell phone

Description automatically generated

Figure Serial Monitor: Distance Sensor

Figure DRV8833 Motor Driver Module

[Motor Subsystem] [A screenshot of a cell phone

Description automatically generated](https://www.amazon.com/HiLetgo-Stepper-Controller-Tb6612fng-Replace/dp/B00UYIFYCW)

[H-bridge technology was used to control the two DC motors of our robotic car. Using PWM (Pulse Width Modulation) on the “En” pin or “stby” on the DRV8833 we can control the speed and direction of the motors. The best thing about the DRV8833 is that it’s a dual h-bridge. This allow us to individually control each of the DC motors. Pin “VM” or “VMotor” is where we supply the voltage that drivers the DC motors. The DRV8833 can deliver up to 1Amp of current for the operation of the motors. Using a Fluke clamp-on meter and a Fluke multimeter, I measured 35mA using a 9V battery as the “VM” source voltage. I purposely seized the rotation of the wheels of the car and measured 66mA of current draw.

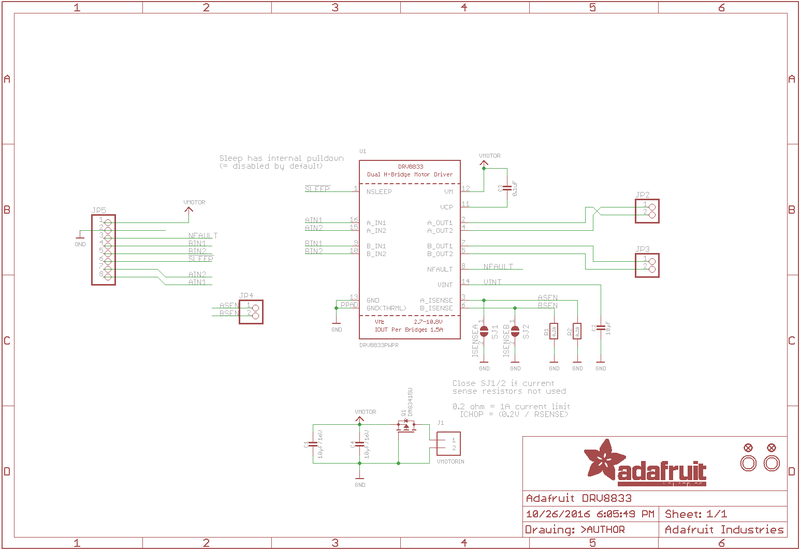
[](https://learn.adafruit.com/adafruit-drv8833-dc-stepper-motor-driver-breakout-board/downloads)

Figure Adafruit DRV8833 Motor Driver Schematic

Controlling the DRV8833 is a little different using the ESP32S than with Arduino microcontrollers. This is because it uses a different library. PWM on Arduino boards use analogWrite() and digitalWrite() for controlling DC motors like we are for this robotic car. However, the ESP32S has its own LED library to do similar functions. We use commands like,

ledcSetup(md1\_ain2, frequency, 8), which allow us to set the frequency and the resolution for the PWM. The frequency was set to 250Hz with 8-bit resolution.

Testing the motor system was interesting. It made this project appear more technical to onlookers who weren’t familiar with the project. Using the IDE serial monitor, my cellphone mobile hotspot, and an oscilloscope, I performed magic to unfamiliar onlookers as I explained what they were witnessing.

Testing the “Forward” direction, I first wanted to show the operation of the microcontroller without the output to the motors. I pressed the Forward button on the display (see Figure \*), watching the serial monitor (see Figure \*) to verify the function. The DC motors have been code with 16-duty cycle increments in the forward direction. The initial value measured 39.1% duty cycle. I didn’t think to record it at the time, but I believe the car was drawing 26mA at the time?

|  |  |
| --- | --- |
| **Increment** | **Duty Cycle** |
| 0 | 0.0% |
| 1 | 39.1% |
| 2 | 40.7% |
| 3 | 42.2% |
| 4 | 43.8% |
| 5 | 53.2% |
| 6 | 46.9% |
| 7 | 48.5% |
| 8 | 50.0% |
| 9 | 51.6% |
| 10 | 53.2% |
| 11 | 54.7% |
| 12 | 56.3% |
| 13 | 57.8% |
| 14 | 59.4% |
| 15 | 60.9% |
| 16 | 62.5% |

Figure Forward Motion Increments

A screenshot of a cell phone

Description automatically generated

Figure Wi-Fi Display on a Cellphone

A screenshot of a social media post

Description automatically generated

Figure IDE Serial Monitor over Wi-Fi (Forward)

Figure Forward 39.1% Duty Cycle w/Motors Disabled

A screen shot of a monitor

Description automatically generated **A picture containing monitor

Description automatically generated**

Figure Forward 62.5% Duty Cycle w/Motors Disabled

After cycling through each of the increments, verifying the PWM. I need to verify the proper operations of the motor driver circuit under load. Enabling the DC motors, I was able to see the noise created by the motors and how they are affected by the PWM signal. I proceeded to cycle through the previous steps, this time enabling the motors by pressing “Go”.

Figure Forward 39.1% Duty Cycle w/Motors Disabled

A screenshot of a computer

Description automatically generated A close up of text on a black background

Description automatically generated

Figure Forward 62.5% Duty Cycle w/Motors Enabled

I went on to repeat the process for “Left”, “Right”, and “Reverse”. “Left” and “Right” were similar, except there was a PWM difference between the two motors. Depending on the direction, the duty cycle of the opposite wheel would be shorter that the other. At higher percentages, the opposite motor would stop as the other wheel kept accelerating. The purpose, essentially to make the car spin faster.

Figure Right 50% Duty Cycle

A picture containing monitor

Description automatically generated A screenshot of a computer

Description automatically generated

Figure Left 50% Duty Cycle

[Power Subsystem]

[In terms of power, there were many alternatives to choose from. First, I needed to supply enough power to the ESP32S, which operates on an input voltage of 3.3V – 5V. I also needed approximately 5V to power the HC-SR04 (ultrasonic sensor). Understanding that the ESP32 has an onboard 5V regulator, I realized that I could potentially supply 3.3V to the pin designated for this voltage and receive 5V at the 5V input. That is correct, the 5V input is virtually and input/output. Supplying 3.3V to the 3.3V input pin was in fact the input to the onboard 5V regulator. This means the ESP32S is capable of supplying 5V in ultra-low power mode.

The next issue was supplying enough current to power the two DC motors. Although, the ESP32S was capable of powering the motors, it didn’t perform as well as I hoped. I began to notice that I didn’t have as much control of the motor rotation speed as I’d preferred? The motors seemed to either move slowly or fast (not as many increments). Additionally, I noticed that I seemed to frequently lose Wi-Fi connection as well? From my work experience, reinforced by knowledge gain from weekly lectures, I figured the motors were probably drawing too much current. This was potentially one of the causes of my defective ESP32S and DRV8833. Professor Tzvetkov mentioned in a lecture regarding this issue, that the large draw in current would drag the voltage just low enough to cause the ESP32S to reset. This was potentially the cause of the lost internet connections? Therefore, using an oscilloscope, I noticed the motors indeed produced an in-rush voltage spike that potentially may have initially damaged my original DRV8833, but possibly my original ESP32S as well? It was suggested by Professor Tzvetkov to place capacitor near the input voltages to filter these spikes. This essentially reduces the ripple voltages produced by the PWM of the DC motors.

Figure DC Motor Unfiltered

**A picture containing clock, screen

Description automatically generated A screenshot of a computer

Description automatically generated**

Figure DC Motors Filtered

Finally, I decided to separately give power to the DC motors by adding a 9V battery. This seemed to improve performance. The issues of the lost Wi-Fi connections were gone. I had more incremental control over the PWM of the motors. I noticed the microcontroller LED shined brighter, as it wasn’t bogged down by the DC motors. When testing the power subsystem, I chose to time how long the DC motors would run on a new 9V Duracell battery, using the full 62.5% PWM duty cycle in “Forward” mode. Throughout our simulations and labs we estimated 2 hours of drive time. With my setup, I measured approximately 1 hour and 43 minutes of drive-time before the motors seized. With the low power consumption of the ESP32S, it may have run forever as far as we know? This is because there was no longer any large current being drawn from the 5.1V battery pack that I used to power it. Essentially, if I decided to configure it in ultra-low power mode, it may’ve seemed like it would run forever?

A circuit board

Description automatically generated

Figure ESP32 LED w/Separate 9V DC Power

[**System Operation**]

As previously mentioned, the ESP32S is Wi-Fi and Bluetooth capable. For this application, we utilized the Wi-Fi system. Using the local IDE, we only needed to write a few lines of code to establish a connection. Uploading the source code to the microcontroller with the proper Wi-fi credentials via the IDE was as simple as clicking “Upload”. Waiting for the IDE to connect to the ESP32S. Then, holding the IO0 (Reset) button when a connection is made. At that time, I like to open the serial monitor of the IDE to verify the Wi-Fi connection.

A screenshot of a social media post

Description automatically generated

Figure Wi-Fi Configuration

A screenshot of a social media post

Description automatically generated

Figure IP Address Given Verifying Wi-Fi Connection

With a customizable web server interface established within the source code, we able to enter the provided IP address into the browser of our choice to control the robotic car as long as the device was logged into the same Wi-Fi network. When successful, I used the mobile hotspot of my cellphone to pair the ESP32S and my laptop to my cellphone for controlling the robotic car. Using the serial monitor on IDE of my laptop, I was able to verify the operations of the car as I cycled through each of the buttons of the web interface.

Figure Web Server Interface

A screenshot of a cell phone

Description automatically generated A screenshot of a social media post

Description automatically generated

Figure Verification of Web Server Commands

I tested the distance sensor by blocking it with my hand. I needed to verify that the car would indeed stop if it came within 4 inches of and obstacle. It worked! Next, I put the car on the floor to check the performance again. It worked as previously tested. However, I noticed the controls were a little slow? I feel this is something I will continue to work to improve in the future. Overall, the project was a success.

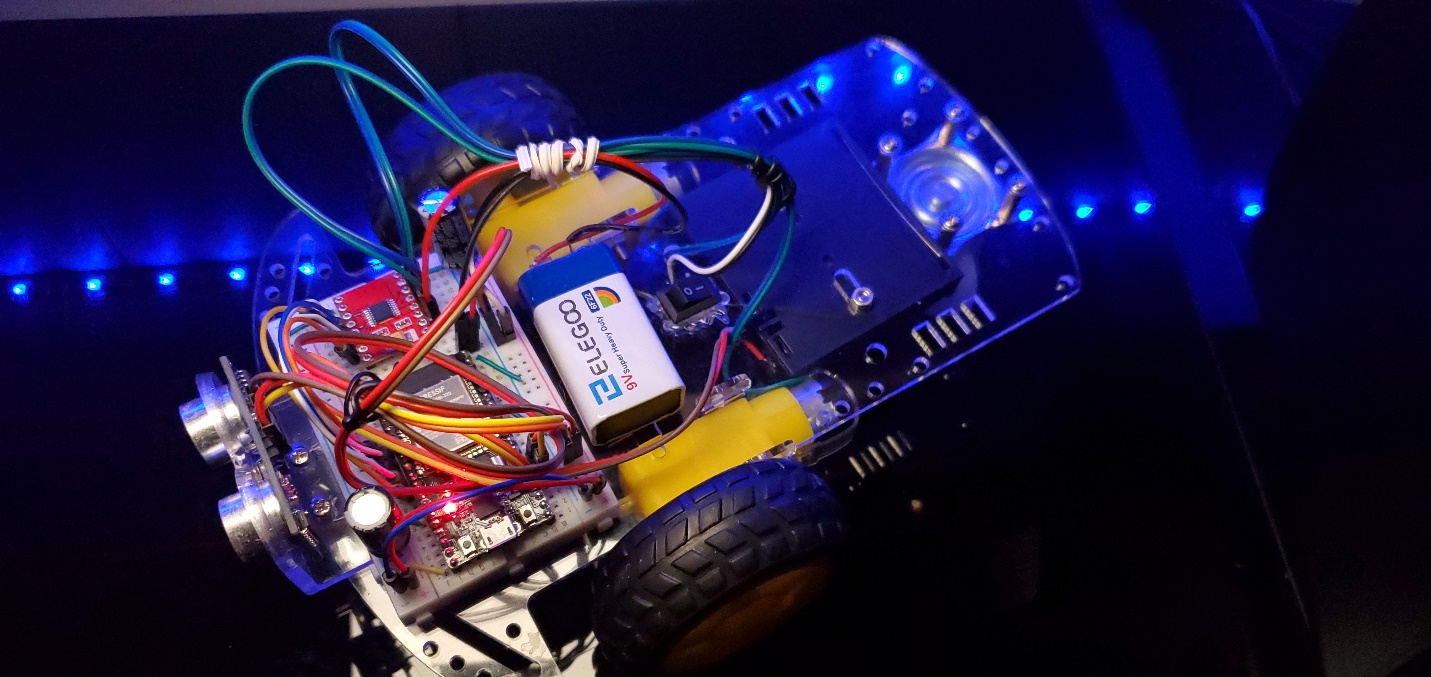


Figure Robotic Car Project

##### Conclusion

The ESP32S is a powerful microcontroller for the price. I has tons of capabilities and has personally sparked interest as a new hobby after I complete my studies. Since I have yet to begin my senior project, it is possible that I may suggest to my group projects involving the ESP32S. While researching the ESP32S, I found several YouTube channels with interesting tutorials for projects based on this microcontroller.

I’d like to thank the Professors of my 3-course series for teaching me what I needed to know. I’d like to thank Professor Tzvetkov, for sharing his knowledge, experience, and assistance as I initially struggled to adjust to this microcontroller, as it was different from the Arduino controllers that I was previously familiar.

References

Adafruit DRV8833 DC/Stepper Motor Driver Breakout Board. (2020). Retrieved 27 June 2020, from https://learn.adafruit.com/adafruit-drv8833-dc-stepper-motor-driver-breakout-board/downloads

AdaFruit DRV8833 Stepper Motor Driver. (2020). Retrieved 27 June 2020, from https://www.amazon.com/HiLetgo-Stepper-Controller-Tb6612fng-Replace/dp/B00UYIFYCW

HC\_SR04 Ultrasonic Sensor. (2020). Retrieved 27 June 2020, from https://www.amazon.com/HC-SR04-Ranging-Detector-Distance-Arduino/dp/B08177P94G/ref=sr\_1\_8?dchild=1&keywords=HC\_SR04&qid=1593273899&s=electronics&sr=1-8

Koyanagi, F. (2020). ESP32: Internal Details and Pinout. Retrieved 27 June 2020, from https://www.instructables.com/id/ESP32-Internal-Details-and-Pinout/

Shenzhen Anxinke Technology CO;LTD. (2016). *ESP-32S Datasheet* [Ebook]. Retrieved from http://www.ai-thinker.com