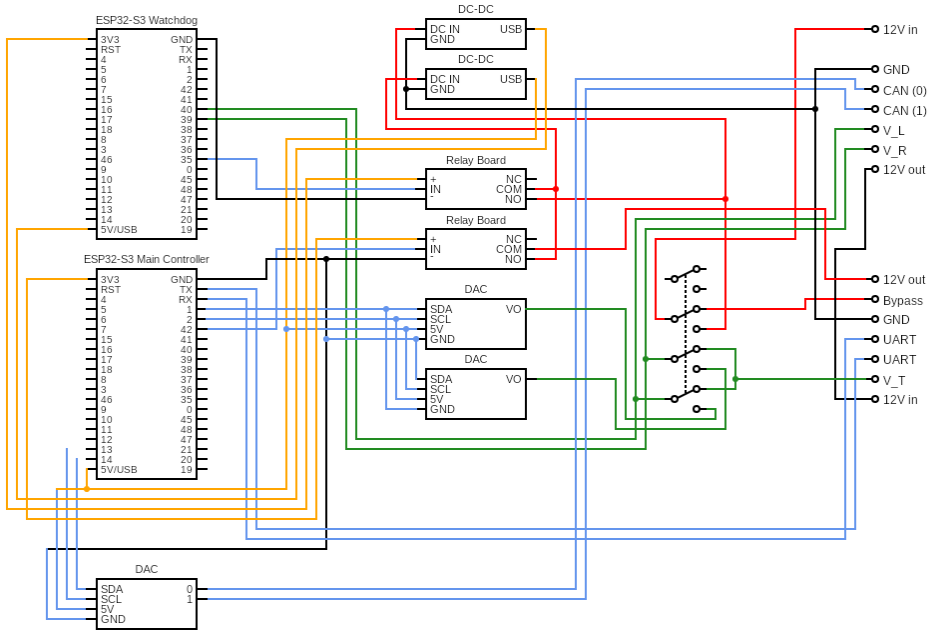
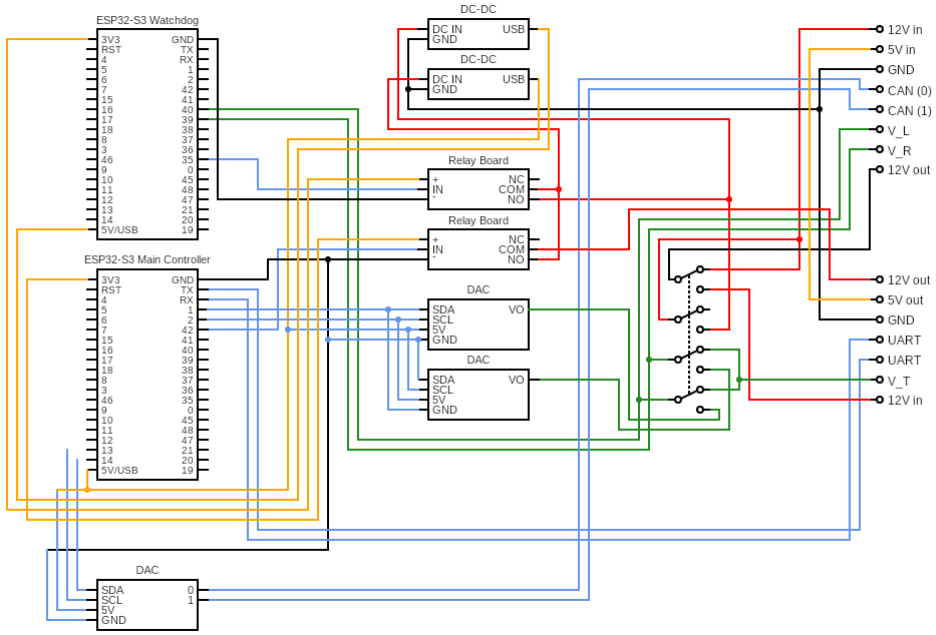
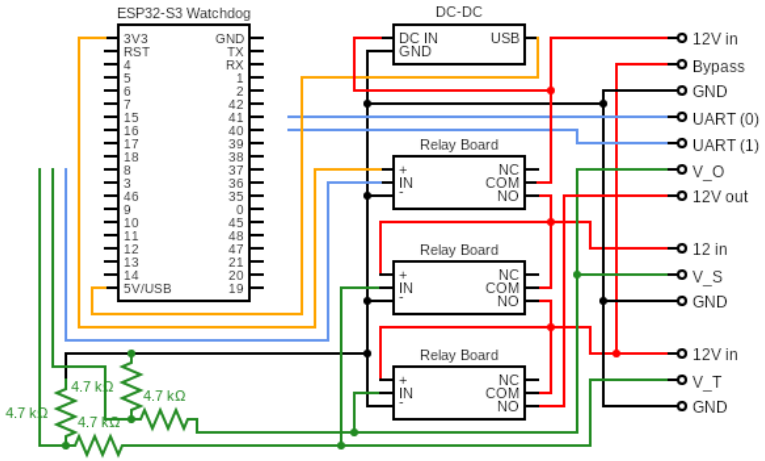
Main Controller, 12V throttle configuration



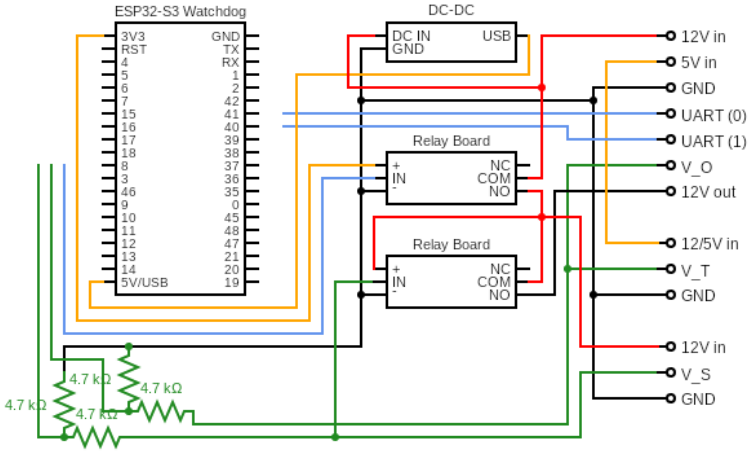
Main Controller, 5V throttle configuration



Auxiliary Controller, 12V Throttle Configuration



Auxiliary Controller, 5V Throttle Configuration

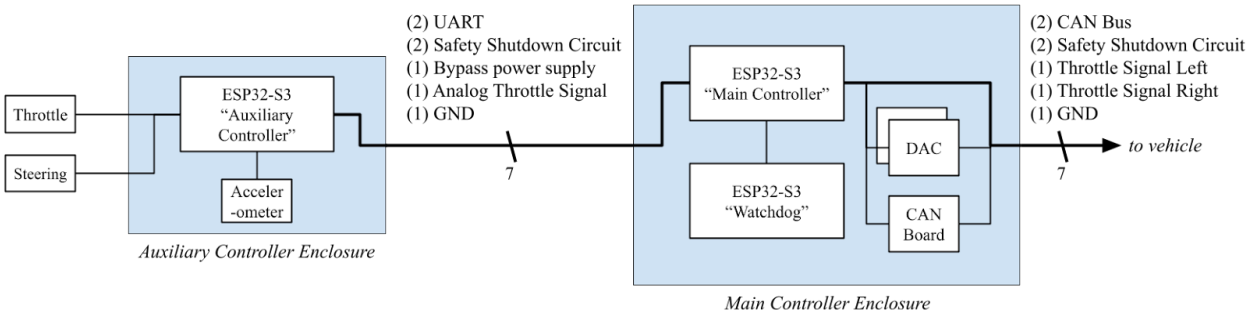


1. *Main Controller ⇔ Vehicle*

6 12V supply	5 GND	4	3	2 CAN (Hi)	1 V_L
12 SSC out	11 5V supply	10	9	8 CAN (Lo)	7 V_R

2. *Main Controller ⇔ [Aux Controller] (same pinout both ends)*

6 12V supply [SSC in]	5 GND	4	3	2 UART TX [RX]	1 V_T
12 SSC in [SSC out]	11 5V supply	10	9	8 UART RX [TX]	7



The system consists of two enclosures, connected to each other by a custom wiring harness, and to the vehicle with a similar wiring harness.

The auxiliary controller enclosure houses the auxiliary controller, which relays input sensor data to the main controller via the Universal Asynchronous Receiver-Transmitter (UART) communication protocol. The enclosure includes our accelerometer sensor as one of our data inputs; the other two sensor inputs come from the steering wheel and the throttle pedal, which are all communicating to the auxiliary controller via Inter-Integrated Circuit (I2C).

The main controller enclosure houses several primary components: the main controller, which is running the torque vectoring algorithm; the watchdog, which monitors system behavior for anomalies; two Digital to Analog Converters (DACs), which output throttle signals; and a CAN Bus controller, which communicates telemetry data with the vehicle.

The output of the overall system is our 3D-printed vehicle demonstration model. Assembled with acrylic boards, printed wheels, and drone motors, the car will respond to the user's inputs coming from the steering, throttle, and accelerometer by taking in the output values produced from the main controller and feeding it through the motor controller attached to the vehicle. In turn, the wheels will spin and/or turn accordingly in real-time.

Torque vectoring employs a computer system that controls the power of automobile differentials more effectively in order to improve grip on road surfaces and allow the vehicle to accelerate more quickly, especially on turns. So, this system will integrate electrical components, a control algorithm, and mechanical implementation.

The main foundation of the hardware/software integration lies within our choice of microcontrollers: the ESP32-S3 developed by Espressif. We chose these parts to drive the main, motor, and watchdog controllers of our system due to the wealth of communication features they provide for us to integrate our components, including two discrete 10-channel ADCs, integrated UART/Serial, I2C, and Serial Peripheral Interface (SPI) controllers, and two logical cores. Other features that were of vital importance included 512kB SRAM, ideal for lookup tables, and its dual-core design allowing multi-task programs. By using the ESP32-S3, our primary software functions were automatically encompassed by ESP-IDF, Espressif's official IoT Development Framework. ESP-IDF contains libraries and configurations within their SDKs to compile and run programs on the ESP32 chips. ESP-IDF is completely free and open source on Github, making it easily accessible to use example ESP-IDF code and modify it to our project deliverables.

Initially, the auxiliary controller was also based on the ESP-IDF functionality as well. However, integrating the Adafruit ISM330DHCX accelerometers with the ESP32-S3 proved to be a difficult challenge, given the complexity of the data byte register mapping on the datasheet and the lack of Internet resources. However, this exact accelerometer had plentiful guidance for integration with an Arduino-based setup. Realizing that the auxiliary controller's main function was transferrable to the Arduino as well, our team decided to switch gears and use a spare Arduino Nano instead of an ESP32-S3 to read sensor data from the accelerometer, steering, and throttle inputs, then package that data to the main controller over UART.

Communication between multiple ESP32-S3s and the Arduino Nano is fundamental to our system design, and we decided to stick with the available UART protocols. UART has a simple data transmission wiring procedure (generally just two wires) that facilitates low hardware complexity and a one-to-one connection between two microcontrollers to send an active digital signal. UART protocol also features reliability for long-distance communication, which aligns with our system design separating the auxiliary and main controllers. Located on different sections of the car, the auxiliary controller can easily feed the input data through packets free of interference to the main controller.

In terms of sensors, our suite collection originally encompassed the following three inputs:

- Adafruit ISM330DHCX accelerometers
- Honeywell RTY360HVNAX rotary steering sensor
- Honeywell RTY050HVNAX throttle sensor

However, due to functional issues with the rotary steering sensor, we could not proceed with using our originally defined steering and throttle sensors. Instead, our team 3D-printed a steering wheel component and collected a formula car throttle pedal from our client, then wired those to the input channels of the auxiliary controller. Our accelerometers were chosen due to the compact packaging and high precision performance, especially with I2C communication to the controllers. Like UART, I2C communication involves a simple hardware setup of two wires: one for the data channel (SDA- serial data) and the other for clock signal (SCL- serial clock). It also supports multi-master and multi-slave buses for high-speed, device connections within a short distance from the ESP32-S3s and Arduino Nano, allowing multiple functionalities and devices to be supported simultaneously.

To design a control algorithm for the torque vectoring system, a variation of PID, a PI controller, was determined to be the easiest to implement, as the team had previous experience with this and there are ample resources online pertaining to PIDs. A PI controller is a PID controller without the derivative term, as it can introduce instability in the controller, especially since our system will have a user input (steering encoder), which can be inherently noisy due to vibrations and the user themselves being unstable. Once the PI controller was determined to be the control algorithm, developing a method of tuning the controller is necessary. To do so, a linear model will be developed to use a root locus method of tuning the controller to streamline the tuning. Due to the nature of the linear model, the PI controller will need to be further linearized at different longitudinal velocities.

Project Costs for Production of Beta Version (Next Unit after Prototype)				
Item	Quantity	Description	Unit Cost	Extended Cost
1	4	Adafruit ESP32-S3 DEVKITC Microcontroller	$15	$60
2	2	Adafruit ISM330DHCX Accelerometer	$20	$40
3	1	Grayhill Rotary Switch	$50	$50
4	2	Adafruit MCP4725 12-Bit DAC with I2C Interface	$5	$10
5	1 (pack of 8)	AZKO 5v Relay Board Relay Module 1 Channel Opto-Isolated High or Low Level Trigger	$12	$12
6	1 (pack of 5)	Solderable Breadboards GK1007	$12	$12
7	1 (pack of 2)	Waveshare SN65HVD230 CAN BUS Control Board	$18	$18
8	2 (pack of 2)	DC-DC Buck Converter Module 12V to 5V Micro USB Power Adapter	$12	$24
9	2	Polycase WC-24F Outdoor Enclosure with Clear Cover (including pack of 100 screws)	$32	$64
10	3	Molex MX150 12 pin male panel mount	$5	$15
11	3	Molex MX150 12 pin female inline plug	$5	$15
12	36	Molex MX150 blade	$0.10	$3.60
13	36	Molex MX150 socket	$0.10	$3.60
14	6	TE Connectivity AMP Connector Blade	$0.15	$.90
15	6	TE Connectivity AMP Connector Socket	$0.40	$2.40
16	2	TE Connectivity AMP Superseal 1.5 3-pin	$1.50	$3
17	42	TE Connectivity Single Line Seal	$0.03	$1.26
18	1	Hilitchi Professional Pin Crimping Tool	$17	$17
19	1	TUOFENG 22 AWG Wire Solid Core Hookup Jumper Wires (6 colors)	$15	$15
20	2	McMaster Cast Acrylic Sheet	$16	$32
			Beta Version-Total Cost	$398.76