Department of Electrical and Computer Engineering

Drive-by-Wire Vehicle for Autonomous Research

ECE Capstone Series 2022

Sponsored by the Cyber-Physical Systems Research Center and
Hybrid Systems Laboratory at UC Santa Cruz

Team Members

Arturo Gamboa-Gonzalez
Micah Herbert
Sutter Lum
Walter Teitelbaum
Sriram Venkataraman

Table of Contents

I. Abstract

In this paper, a modular Drive-by-Wire system is developed and applied to an experimental test platform which emulates a physical vehicle. We aim to address the lack of open source, low cost, and vehicle-agnostic autonomy conversion kits. The proposed solution surpasses previous methods due to its matured safety systems, intuitive user interface, and ease of implementation. Through custom actuation of a target vehicle's steering, braking, and acceleration, we provide an intermediary between autonomous control algorithms and the vehicle, facilitating safe development and testing. Experimental results prove that this solution satisfies the mechanical and software criteria set by autonomous control researchers. The system is promising for use by control departments to begin rigorous testing of algorithms on a wide variety of vehicles.

II. Introduction

University researchers approaching autonomous driving algorithms lack centralized platforms to apply their research. Their current options include scaled models, simulation, and expensive closed-source solutions from private corporations. This project aims to develop a modular Drive-by-Wire (DBW) system to enable autonomous control of utility vehicles similar to the Polaris Gem e2 for UCSC's Hybrid Systems Laboratory (HSL), which will serve as a platform for research on sensing, control, and networking in autonomous vehicles. The project's short-term goal is to enable gamepad control of the vehicle, and the long-term goal is to prepare the platform for safe and autonomous navigation within the UC Santa Cruz extended campus through autonomous algorithms. The vehicle will be utilized as a testbed to develop practical on- and off-campus applications, such as autonomous delivery and transportation. As autonomous vehicle research is a new advent to the UCSC campus, campus safety administrators have

provided strict design constraints on the Drive-by-Wire system. Within nominal operation, the system will need to enable digital control of the vehicle's steering, braking, and acceleration, while also maintaining the original functionality of the manual driving controls. The system needs to quickly and safely shut off if a problem occurs, and needs to allow for safe and succinct manual override. These unique yet universally desired requirements differentiate our implementation from other solutions.

Projects similar to this one have been completed before, although most involve the complete removal of the vehicle's steering column [1] which goes against the client's specifications of allowing the vehicle to be manually operated by a driver. Additionally, companies such as Nexteer approach software control through "steer-by-wire", which involves mechanical detaching the steering wheel from the steering mechanisms, thereby introducing the potential for software communication failures. One competitor, Dataspeed, offers commercially available products that allow for DBW control of similar vehicles, but these products are cost-prohibitive and closed-source [2]. It is for these reasons that HSL discarded these solutions as options for their vehicle, as they did not align with their research goals or the university safety requirements. Since our team offered HSL a solution that is low-budget, well documented, open source, and can be easily modified in the future, our collaboration quickly became the logical option for them.

Although this vehicle is meant to be a platform for autonomous vehicle research, safety for both the passenger and pedestrians remains the highest priority. Guided by discussions with the Human-Robot Interaction Lab at UCSC [3], our team is responsible for ensuring the safe operation (see Section 5) of our DBW system as well as providing redundant Emergency-Stops (E-Stops) for any potential users of the system. Finally, this includes an intuitive user interface to

facilitate the balance between enforced safety by the DBW system, and the desire by the operator to test their autonomous algorithms.

Since the intent of this DBW system is to be applied on a wide variety of vehicles, we sought to provide a proof of concept through an experimental setup that we call the “System on a Bench”. This will consist of our methods of interfacing with the steering, braking, and acceleration subsystems of the vehicle, as well as peripherals including the turn signals, horn, and headlights. It will also showcase the fully developed safety systems, emulating the control flow of using autonomous algorithms or the gamepad to drive the vehicle. Further, it enables the testing of autonomous driving algorithms through hardware-in-the-loop simulation.

The remainder of this document discusses overall design approaches, implementation, and validation of the DBW system. Additionally, it will serve as a resource to guide research teams in the implementation of our DBW system on candidate vehicles. Section 3 presents the overall system level view of our project, discussing scope and limitations we faced as well as the concept of operations. Sections 4-10 provide further detail into the methods used, reviewing the implementation and operation of each subsystem and how they align with client requirements. Section 11 presents the next steps for completing the project, and lays the framework for future development of the system.

III. System Level View

Modular Drive-by-Wire Representation

Our implementation of the Drive-by-Wire system differs from previous implementations due to its modular design. Since each subsystem is independent of each other, it is easy for future teams to build upon or revise our solutions for each subsystem. Furthermore, the modularity of our solution allows for the Drive-by-Wire system to be easily adapted to other vehicles, instead

of being custom tailored to only the Polaris GEM e2. The safety, user interface, autonomous controller interface, and underlying system will be the same when implementing our system on different vehicles. The added advantage of this approach is that duplicating the DBW system will lead to intuitive use, and all documentation can be consolidated (since multiple versions wouldn't require new training on user interface). The difference in various vehicle implementations is with how the steering, braking, acceleration and peripherals are interfaced. With the future designs outlined in Next Steps (Section), each subsystem can be changed and modified at will, all the while retaining the same functionality. For further information on specific vehicle implementations, see Integration (Section).

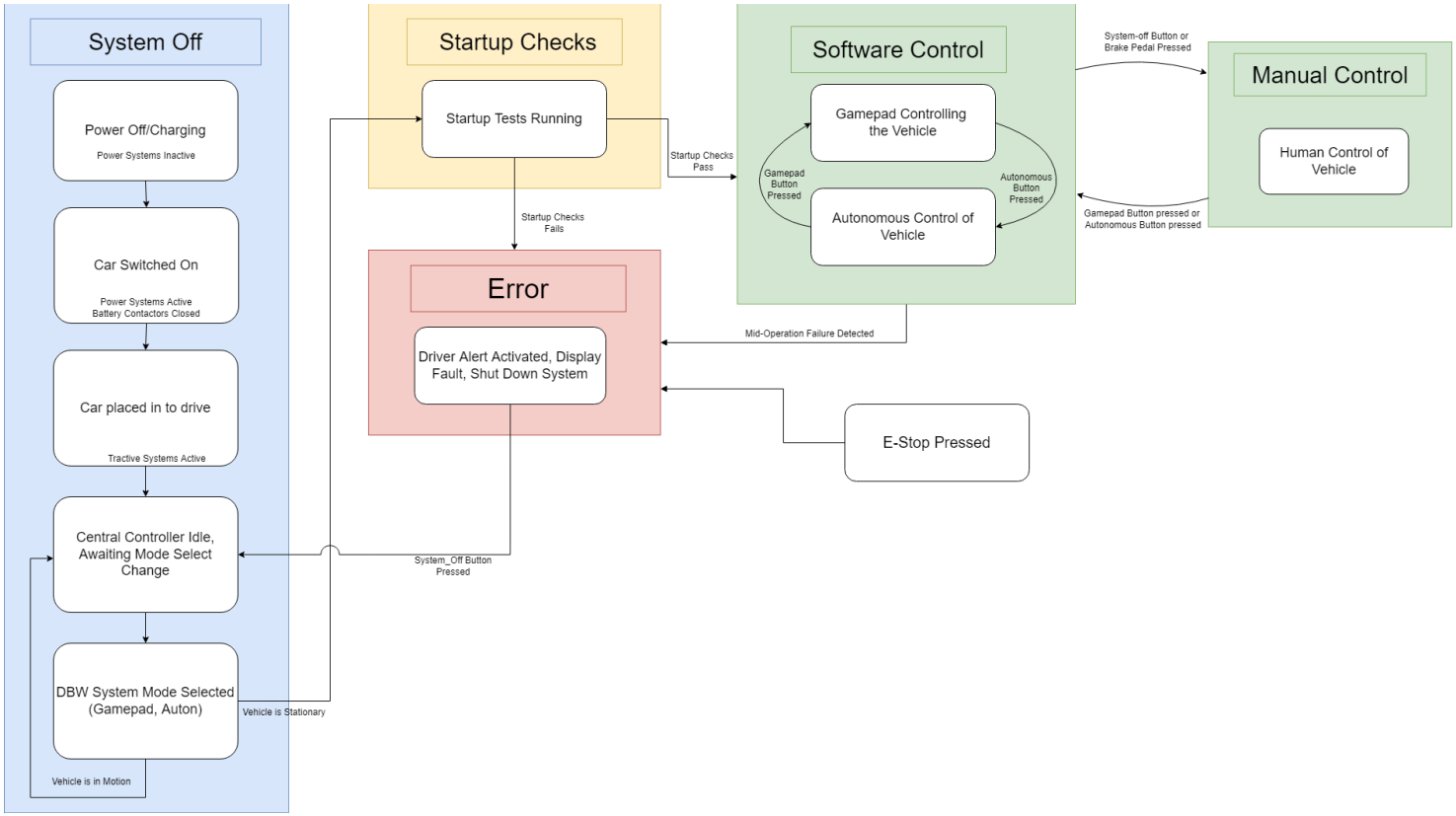
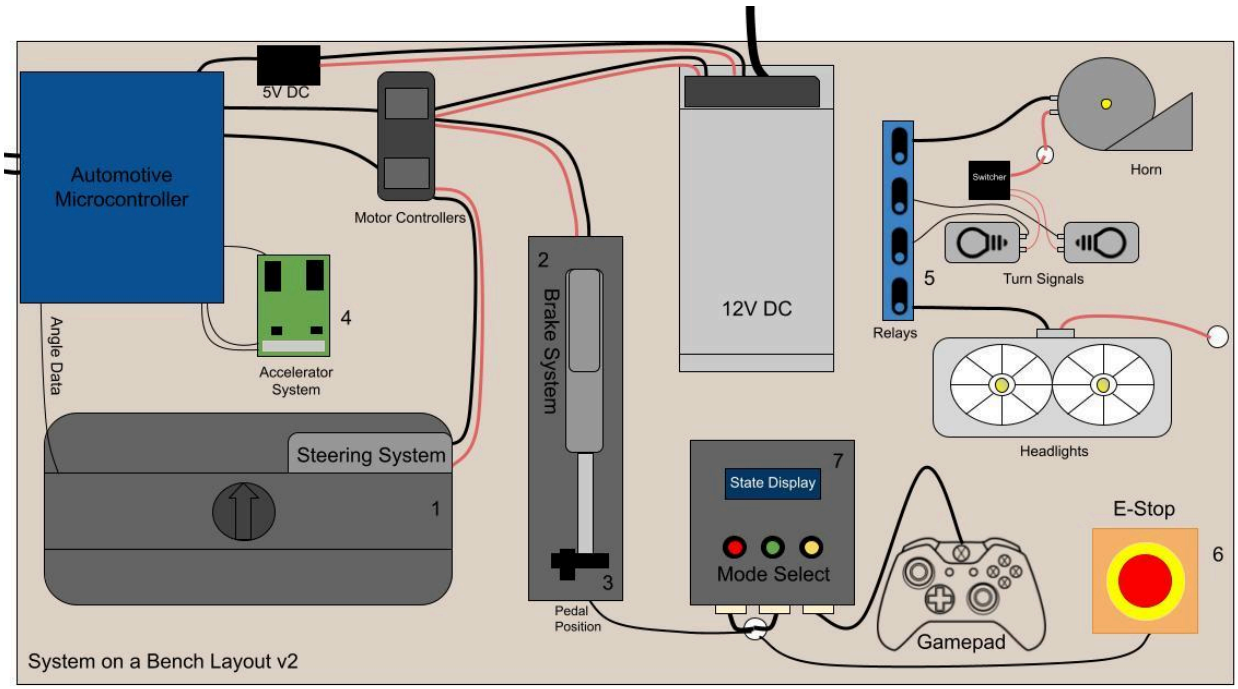


Figure __: *Flow Chart detailing the Concept of Operation.*

Concept of Operations: Figure __ guides a potential user of the Drive-by-Wire system through one of the use cases: testing and verification of control methods, and hardware in the loop simulation for the development of autonomous control algorithms. The user interaction with the system can be abstracted into five major areas: the Drive-by-Wire system is off, the system is performing startup checks, the system is being controlled through software (either with the gamepad or autonomous algorithm), the system has been temporarily bypassed and the vehicle is manually controlled, and the system has observed an error and prevents further operation.

System Functional Overview



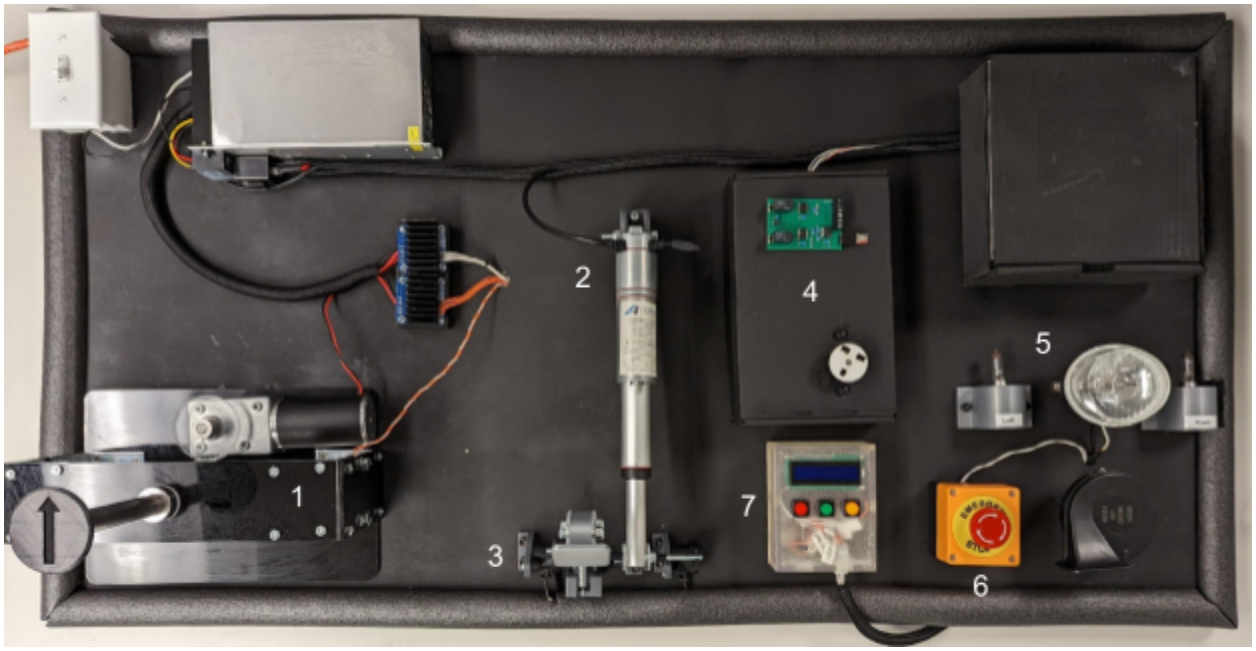


Figure ___: *System on a Bench. (Top) Schematic top-down view. (Bottom) Photograph of the constructed system.*

(1) Steering Subsystem – A 12V DC motor is geared directly to a standard ¼” steering column. A linear feedback potentiometer allows our system to command and control a precise steering angle.

(2 & 3) Braking Subsystem – A 12V DC linear actuator is positioned adjacent to a spring-loaded brake pedal. When the actuator contracts, the brake pedal is coincidentally depressed. Two automotive pedal position sensors provide feedback for the pedal and the actuator independently, allowing our system to accurately control braking intensity as well as detect manual override by the user.

(4) Acceleration Subsystem – A digital to analog converter allows our system to output a precise voltage in order to replicate the output of a standard accelerator pedal. A custom contention circuit allows our synthesized signal to take priority over the vehicle’s accelerator pedal when the Drive-by-Wire system is active.

(5) Peripheral Interface – An array of relays allows our system’s microcontroller to control the vehicle’s horn, headlights, and turn signals independently. These relays are intended to be spliced into an existing vehicle signal harness.

(6) Safety Systems – E-Stops located inside and outside the vehicle allows a user to quickly and safely shut off the Drive-by-Wire system in the case of an emergency. When an emergency stop event is detected, our system will bring the vehicle to a stop.

(7) User Interface – A user is intended to interface with the Drive-by-Wire system either via an autonomous driving computer or via a gamepad controller. Our system’s mode select panel allows the user to easily switch between the methods of input as well as turn our system off.

IV. User Interaction

Mode Selector Panel

The mode selector panel serves as the primary form of user interface with the Drive-by-Wire system. Using the Mode Select Panel, a user can select between modes of vehicle control, read error messages and status updates, and shut off the DBW system. The panel uses three back lit buttons to indicate which control method is currently in use (i.e. the “OFF” button will be illuminated when the system is under manual control, and so on). Also included in the panel is a buzzer used to indicate error states or emergency stops of the system (Figure __).

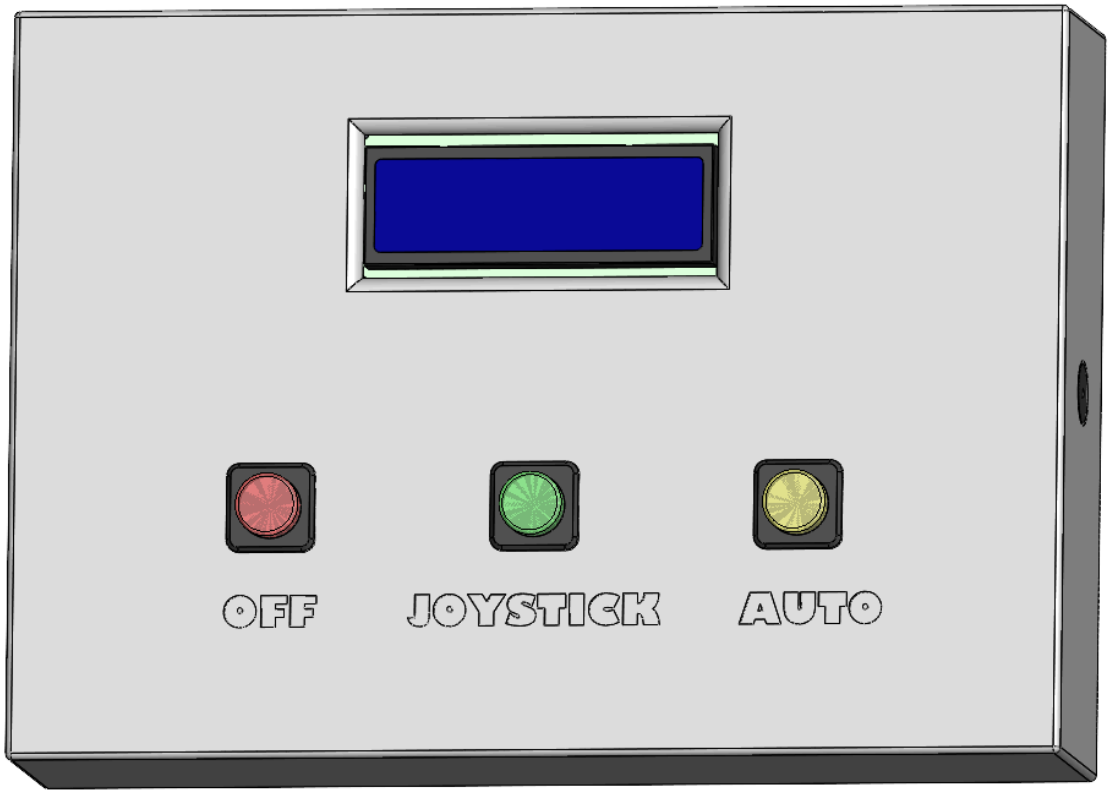


Figure __: Mode select panel CAD Model (Front view).

If the OFF button is pressed during software control of the by-wire system, incoming software commands will be rejected, and the vehicle will only be controlled manually. Additionally, if the user wants to switch from gamepad to autonomous control (or vice versa), they will have to press the off button, then press the desired button. This is to prevent the system

from accidentally switching to autonomous control due to accidental button presses. The display will report the current state of the vehicle (idle, startup, software-operation, manual-operation, or error), and will report what caused the system to enter an error state. There are also areas of the concept of operations in which the user may not know what the system is doing (particularly the startup tests), so the display allows the user to understand the underlying functionality without being required to understand the complete technical functionality of the system. Finally, a 12V buzzer is used to indicate to the user that an error has occurred. Even though this will be shown through the display, these annunciators of failure are vital for safe use of the system.

The mode select panel includes integrated driver circuitry for all button LEDs and the buzzer so that they can be toggled using GPIO pins from the central controller. The schematic for this circuitry is shown below in Figure __.

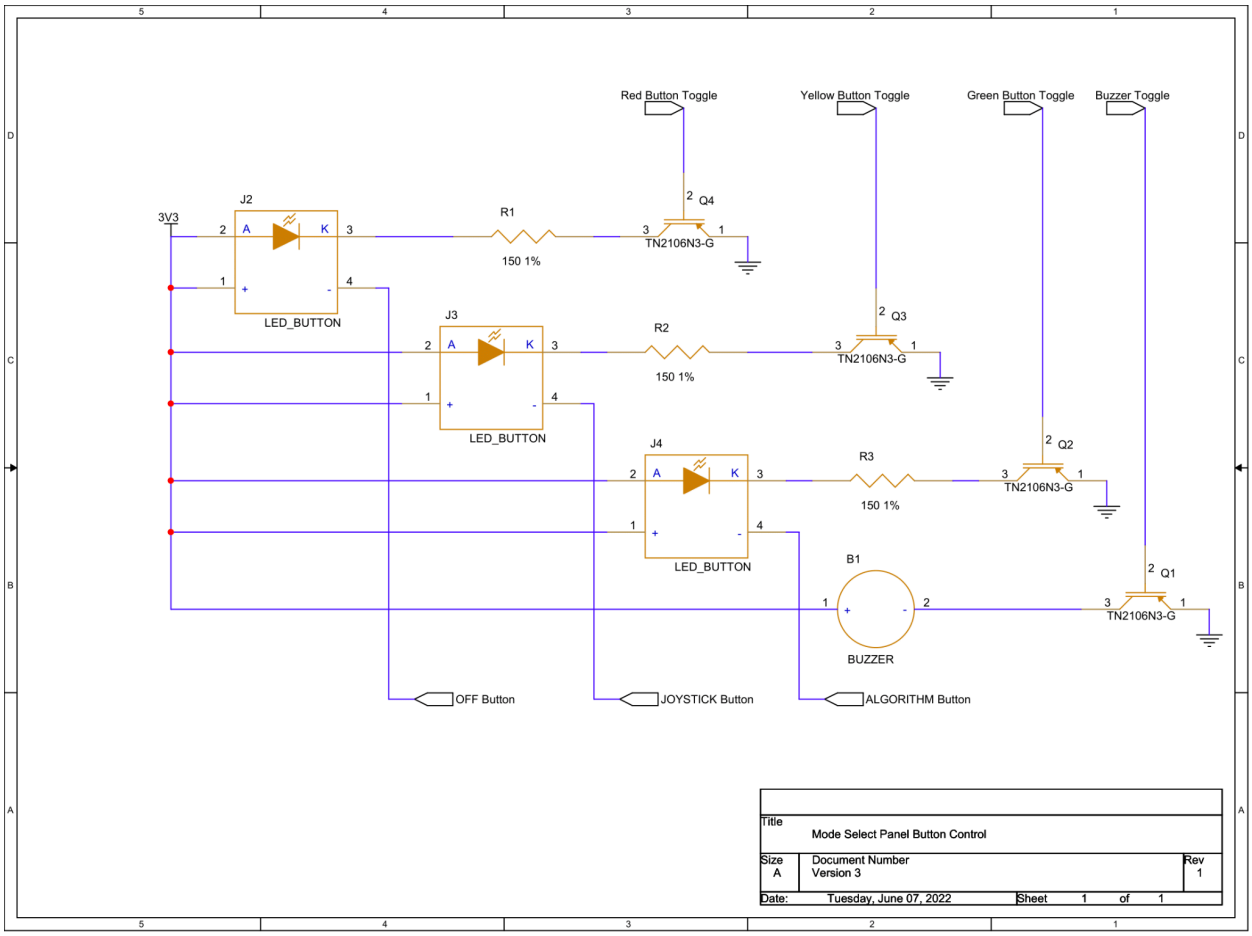


Figure __: *Mode Select Schematic.*

Gamepad Interface

The first method of by-wire control is through an Xbox 360 game controller. In addition to serving as a method of complete vehicle control, the gamepad serves as validation for software control, as input on the gamepad is translated to reference commands for the actuators. Figure __ depicts the way in which the vehicle can be driven by the gamepad; the left joystick x-axis is used to control the steering angle, the left trigger controls what percent the brake actuator is being extended, and the right trigger controls what percent of maximum acceleration we are commanding the vehicle. Button Y, the directional pad, the right joystick, and the middle three buttons (back, guide, and start) are available for use by the implementer of the DBW system for

custom purposes. Since the joysticks and triggers are analog, they provide the user with great precision in vehicle control. This particular scheme (left trigger brake, right trigger throttle, and left joystick steering) mirrors several driving video games [ ], allowing for intuitive control.

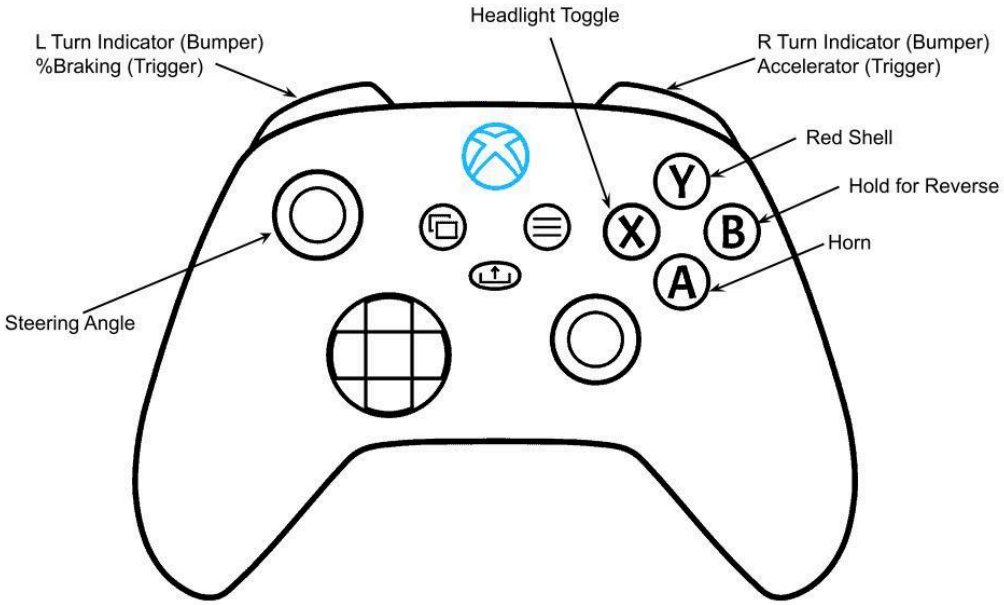


Figure ___: *Joystick button mapping for vehicle control.*

Autonomous Controller

The second method of by-wire control is through the use of a higher-level controller, which uses a future sensor suite to autonomously control the vehicle. All autonomous controllers will send the same type of data to the DBW system, and will receive the same type of data from the DBW system. This will be covered in more depth in the following section, as well as the API calls that a developer can use to command our system. The tools that an autonomous software developer chooses to run on their autonomous controller is completely in their control; they can choose to use whatever visualization software they feel is best suited to their applications.

V. Central Controller

Design Specification

The cornerstone to the DBW system is a controller which facilitates the interaction between software control methods (either through the gamepad or autonomous controller) and the actuators controlling the mechanics of the vehicle. We have aptly named this the “central controller”, although considering its role in ensuring the safety of the passengers and pedestrians, “safety controller” was also a candidate name. This section will cover the requirements for candidate microcontrollers, the overarching software framework, and specific processes which need to be run for desired functionality.

Microcontroller Selection

The main characteristics to look for when choosing an automotive microcontroller are “hardware support for virtualization, quality of service settings, the ability to firewall peripherals, freedom from interference and secure compartmentalization of software functions while supporting concurrent multiple ASIL safety levels” [5]. The platform which best fits these standards is the STM SPC58EC-DISP automotive microcontroller. This board enables the use of a wide variety of communication protocols, including I2C, SPI, Ethernet, and CAN. Thus, we chose to use the SPC58EC-DISP as our central controller, since it fit both our immediate needs and future expectations from the Drive-by-Wire system.

State Machine

Now that the concept of operations has been introduced, as well as the intended user interface, this section can focus on the software architecture that enables the intended behavior. This current section will focus more on the state machine functionality, covering specific software processes and threads which run within these states. This will encapsulate the entire abstraction stack from a high level input command to the underlying software processes that follow. The state machine is shown below in Figure __. The individual states will now be analyzed in greater detail.

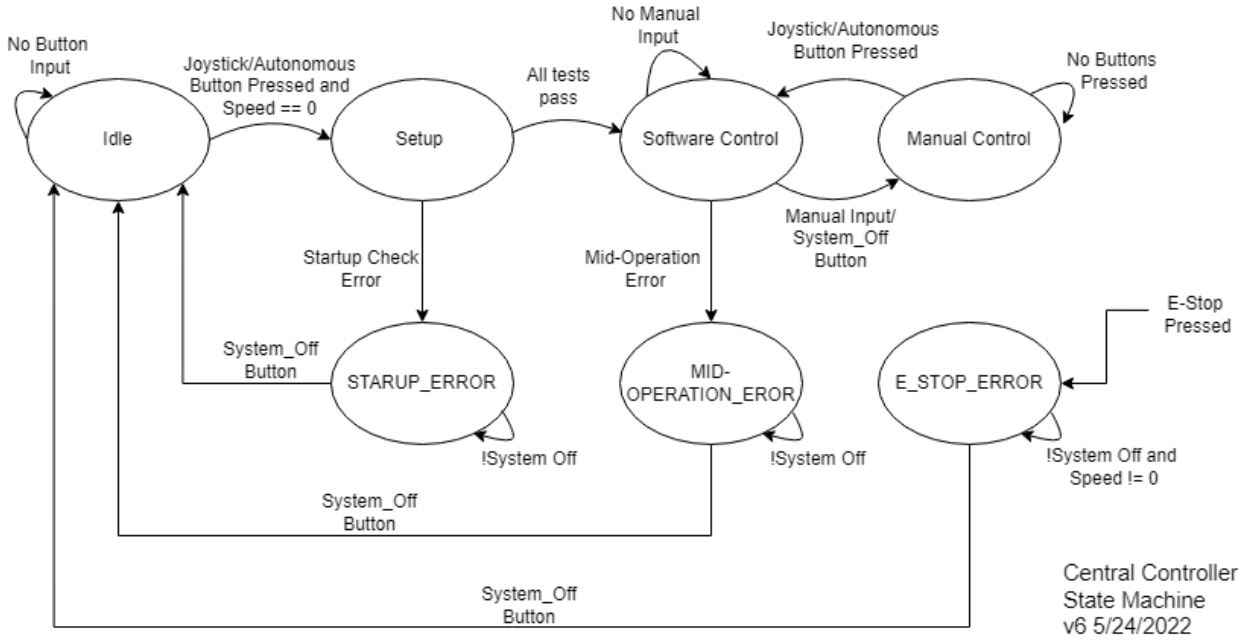


Figure __: *State machine of the central controller.*

Idle State: When the vehicle is initially keyed on and placed into low voltage mode, the central controller will boot up and enter the idle state. While in this state, the DBW system will simply wait for software control to be initiated through the input mode channel (either the gamepad button or autonomous button is pressed). If the vehicle is in motion while the system is idle and one of these buttons is pressed, the button press will be rejected. This is because the setup state requires the vehicle to be idle.

Setup State: The setup state is where the DBW system checks that the system is safe and ready to use. Specific tests are run in this state and are listed as follows.

1. Mode Selector E-Stop Test: The user is required to press the emergency stop button on the mode selector panel, verifying the mechanical and software connection.
2. Brake Test: This test commands the braking linear actuator to 50% and 100% extension, and verifies that the resulting brake pedal angle proportion is 50% and 100%.
3. Steering Test: This test uses the steering motor to command the steering column to ±5°, and using the steering column potentiometer, verifies that this angle is correct. See Section X (Next Steps) for future iterations.
4. Accelerator Test: The accelerator of the Gem is commanded to the "off voltage" (1.08V) using a dedicated circuit (see section IX), so this test verifies the connection by reading the actual output with the central controller's ADC.
5. Joystick Controller Ping Test: The central controller sends a specific message to the joystick USB host controller and awaits a predefined response. If the response is not received within a time frame, an error flag is raised.
6. Autonomous Controller Ping Test: Similar to the joystick controller's test, the central controller sends a specific message to the autonomous algorithm controller. If the predefined response is not received, then the central controller prevents the autonomous mode from being enabled. Operation of the DBW system via joystick is still possible if all other checks are passed.

Software Control: This state is the main feature of the DBW system; it receives input from the gamepad and autonomous controller, polls for a human override, commands the actuators, and sends state data back to the autonomous controller. The multitude of tasks performed in this state

necessitated the creation of the computation graph shown in Figure ( ). This diagram details every software process, function, and thread which the central controller performs during the software operation state, and the following discussion in this section will mirror this diagram.

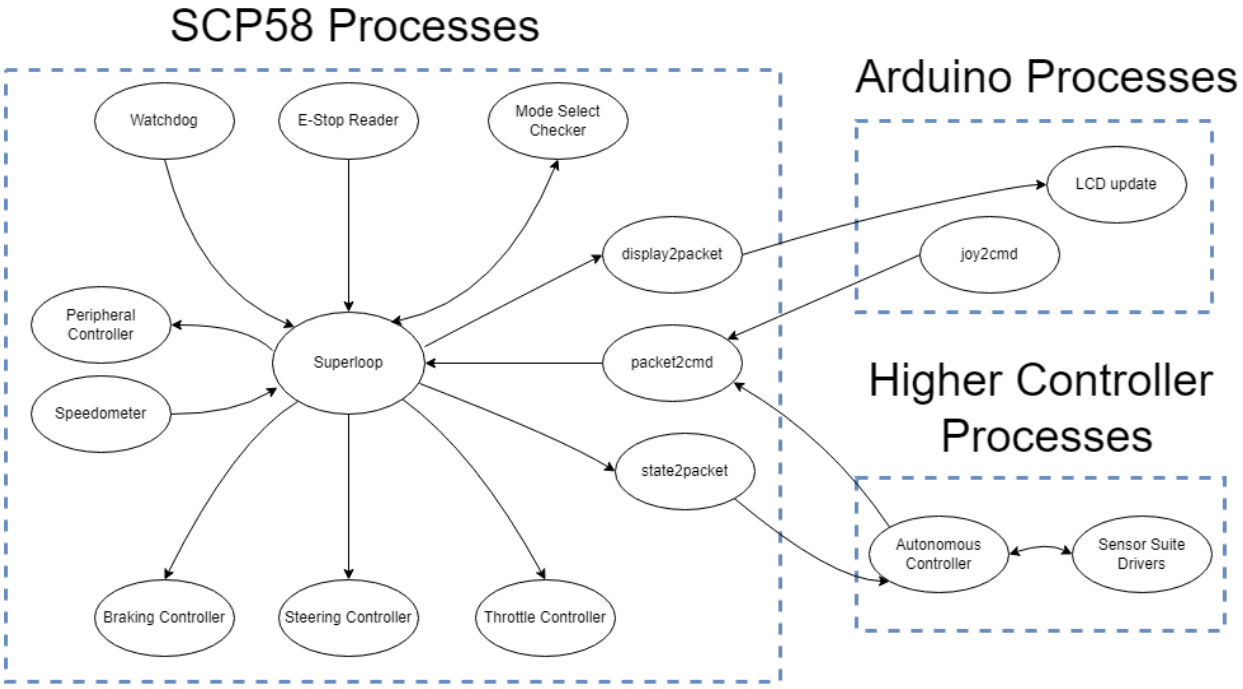


***Figure* :** *Graph of Active Processes in Software-Operation State*

The central controller can be controlled via two methods, a top-level controller running autonomous algorithms, or through a gamepad. The central controller lacks the capabilities to act as a USB host for the gamepad, so an intermediary microcontroller (the Arduino) is used as an interpreter for the gamepad. Therefore, the central controller uses UART (Universal Asynchronous Receive Transmit) serial communication protocol with both the top-level controller and the interpreter controller. Upon receiving a packet, the messages are parsed into (i) steering command, (ii) braking command, (iii) accelerator command, and (iv) additional peripherals commands. (i) The steering command differs based on the input priority source. The top level controller sets the turn rate of the steering column, whereas the joystick directly commands the angle of the wheels. (ii) The brake command sets the depression of the brake

pedal via a linear actuator from not pressed to fully depressed. (iii) The accelerator command sets the ACC (Accelerator Contention Circuit) voltage level to the equivalent accelerator pedal depression. (iv) The fourth part of the message is a bit array that toggles the different peripherals discussed in Section X.

The braking and steering subsystems use DC actuators to control position, which are powered via the high current 12V power supply. Connecting the actuators directly to the power supply would mean that the actuators are continuously active at full throttle in a single direction, therefore, H-Bridge Motor Drivers were used to control the actuators. H-Bridge Motor drivers allow for the central controller to adjust the polarity and power of each actuator using two PWM (Pulse Width Modulation) channels (one for each direction) by changing the duty cycle of each PWM signal.

When a position is commanded to the braking or steering subsystem, the value is passed as the reference input of a control loop, where the current position is determined with potentiometers. A simple discrete PID controller with saturation limits was designed for each subsystem, where the output of the controller (or the control effort) was scaled to be the duty cycle of the PWM signal pair.

In order to read logic inputs, two methods are employed: periodic polling and edge interrupts. A global timer is configured to raise a flag at a periodic rate, which triggers the central controller to read and store the logic level of several GPIO pins. This method is used for non-time-sensitive inputs, such as the mode select panel. For cases where transitions are important or time-sensitive, interrupts are used. Interrupts are used to read the speedometer rotary encoder and detect E-Stops, as the system needs to respond as quickly as possible to those inputs.

There are 4 cases that will cause the Drive-by-Wire system to disengage during operation. Any of these failures will result in the DBW system entering the MID-OPERATION_ERROR state:

1. Serial Communication Failure between SPC58 & Joystick/Autonomous Controller: This error case covers two different types of failure. The first type of failure involves a bit-level desync between the two communication members, which is detected by analyzing the head and tail of each message to ensure proper placement. In this case, the partial message is disregarded and the system begins to parse incoming serial data bit-by-bit until the contents of the RX buffer align with the expected packet structure. The second type of failure occurs when a valid message is passed in with a checksum that fails to validate the checksum calculated using the rest of the message.
2. Braking Subsystem Failure: This is detected through an analysis of the error between commanded and actual brake actuator extension. Additionally, if the brake pedal is less depressed than what the actuator is commanding, an error will also be generated. This test can determine both mechanical failures of the actuators, and software failures in communication between the actuators and the central controller.
3. Steering Subsystem Failure: This is identical to the braking subsystem failure, but with the error being observed between commanded and actual steering column angle. See Section XI.A (Next Steps) for future iterations.
4. Accelerator Failure: The error between the commanded voltage on the ACC (Accelerator Contention Circuit), and the actual (as measured by the ADC) exceeds a threshold, indicating a potential electrical failure of the ACC assembly.

If one of these errors occurs, entering E_STOP_ERROR state would be suboptimal; for example, if there is an issue with the brake actuator, the shutdown routine would be rendered useless. This necessitated the creation of MID-OPERATION_ERROR state, in which the DBW system plays a tone which alerts the driver to take manual control of the vehicle.

Manual Control: In terms of functionality, this state operates similarly to the idle state; the DBW system rejects software input and allows the user to manually drive the vehicle. The distinction lies in the transition from manual control to software control: rather than requiring the startup tests to be run, the DBW system can transition directly to software control. This is because it is assumed that the startup tests have already been run once and that the mid-operation checks can catch any mechanical or software failures that have arisen since the startup of the vehicle. A further distinction is that in this state if any emergency stops are pressed the DBW system will enter the shutdown routine. This is not the case with the idle state.

Error Modes: There are three broad categories of errors; errors that arise from the detection of system failures during startup, errors detected during system operation, and the pressing of an emergency stop. These three error states are discussed below.

1. STARTUP_ERROR: This state is entered following a startup check error. The DBW system issues a loud tone and prevents the DBW system from being started. To leave this error state, the user will need to press the “System Off” button on the mode select panel, after which the DBW system will return to its idle state.
2. MID-OPERATION_ERROR: This state is entered following a mid-operation error. The DBW system issues a loud tone and prevents the DBW system from being re-started. To leave this error state, the user will need to press the “System Off” button on the mode select panel, after which the DBW system will return to its idle state.

3. E_STOP_ERROR: This state is entered following any E-stop being pressed. The brake is pressed fully until the vehicle speed is 0, after which the DBW system remains in the error state. To leave this error state, the user will need to press the "System Off" button on the mode select panel, after which the DBW system will return to its idle state.

Communication Methods:

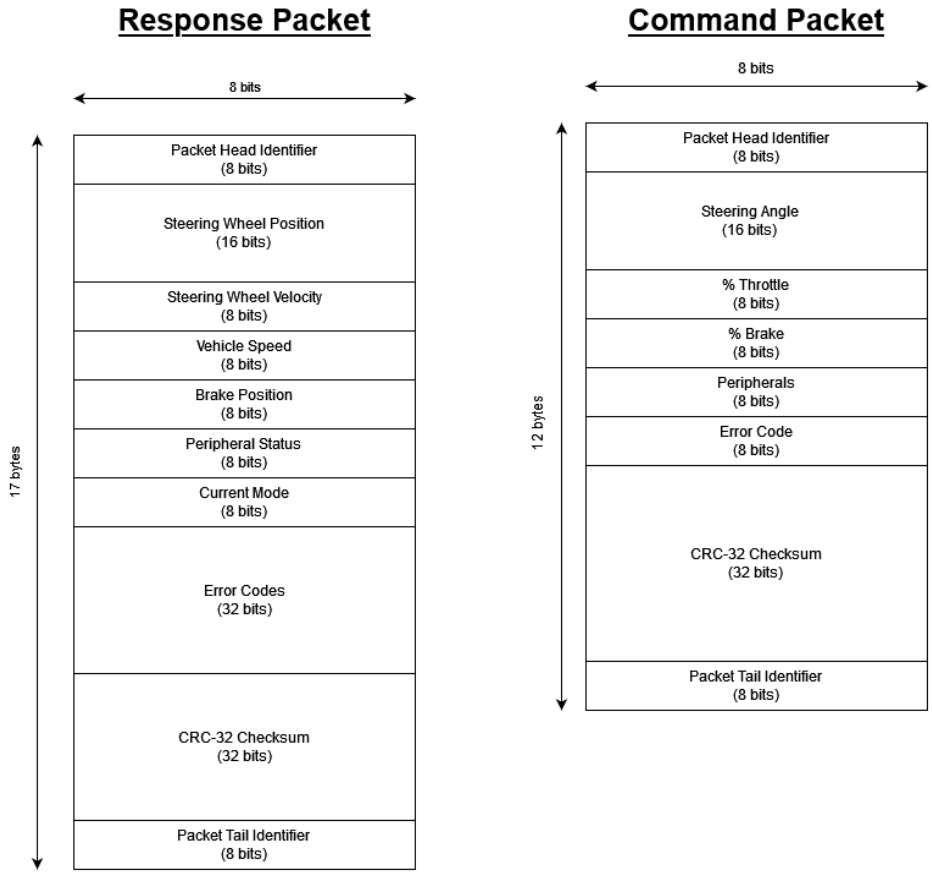


Figure __: *Command Packet to- and Response Packet from- Central Controller*

As shown in Figure X, the central controller is capable of taking input from either the top-level controller or a connected joystick. Regardless of the connection, the central controller expects to receive a twelve-byte command packet containing the vehicle's desired steering angle, percentage of brake applied, percentage of throttle applied, and an error code (if applicable); as

well as toggles for the headlights, left and right turn signals, horn, windshield wipers, gear shift, and emergency stop. From this data, a checksum is calculated and appended to the end of the packet. The packet then has head and tail indicators attached, which allow the system to detect when a set of data fits into a valid packet structure. When the central controller sends a report packet upstream to the top level, the same process is repeated with much of the same data, save for the addition of steering velocity, the % throttle statistic being swapped for the vehicle's current speed, and space reserved for the error code being expanded. After these alterations, the size of the response packet comes out to seventeen bytes

Design Verification

Since there are several edge cases that cause the state machine to exhibit errors, prolonged testing was done with the intent to break the software functionality of the system. By having individuals who weren't familiar with the project use the DBW system, areas of unintended behavior and unhandled streams of execution were uncovered. The feedback received was that the system is intuitive to use, and would be easy for a developer to test their algorithms. Since the design of the state machine was driven by the concept of operations, which was again designed around the desires and requirements of an intended user (a graduate researcher), the feedback received by users of our experimental setup proves that the design was successful and fulfills its purpose. The final version of the command-packet communication protocol was tested by connecting the gamepad and converting its output into command packets. These packets were received and interpreted by the central controller, which was analyzed for any discrepancies. The communication checks (both bit-level and byte-level) proved to be successful in receiving data, as communication failures arising from the physical layer are detected immediately.

VI. Safety Systems

Design Specification

The UCSC campus administration has mandated that the DBW system needs to be able to yield manual control to the user at any point. As a further constraint, the system needs to be intuitive such that any user would be able to operate the system without having to understand the underlying state machine. There is a balance that needs to be achieved between the expected functionality specified in the concept of operation, and the technical aspects of the state machine. Thus, the goal is to abstract the technical minutiae in a way that is clear to any user, whilst also maintaining strict safety requirements. The following sections will delve into the design choices made toward this goal, covering both the underlying technical details and the intended user interaction with the safety systems.

Signal Blending

As requested by the Hybrid Systems Laboratory, the DBW system should not be selecting a singular source of software input to the vehicle, but rather commanding the priority of the different inputs. For example, if an autonomous lane-following algorithm drifts to one side of the road, the user should be able to easily pick up the gamepad and readjust the vehicle, and once the vehicle is in a better position, resume autonomous control. The following table dictates the control scheme for facilitating this signal blending for priority, covering both overrides and emergency stops. In the table, **DC** (meaning don't care) represents that the detected input will result in no state transitions. In the event that braking input is detected, during manual operation, this signal is meaningless since the vehicle is already being manually controlled. If the vehicle is being controlled by either of the software modes, then detected braking will result in the

disengagement of software control and yield vehicle control to the driver. If the user is driving manually or with the gamepad, the detection of the gamepad input will result in no transitions. However, if the vehicle is being autonomously controlled, like the lane-following case mentioned above, detected gamepad input will yield control over to the gamepad (until the autonomous button on the mode select panel is pressed again). If any of the three emergency stops are pressed, regardless of the DBW control mode, the shutdown routine will be engaged.

	Manual Driving (system on)	Gamepad Mode	Autonomous Mode
Braking Input Detected	**DC**	Switch to manual	Switch to manual
Gamepad Input Detected	**DC**	**DC**	Switch to gamepad
Mode Select E-STOP	Shutdown routine	Shutdown routine	Shutdown routine
External E-Stop	Shutdown routine	Shutdown routine	Shutdown routine
RC E-Stop	Shutdown routine	Shutdown routine	Shutdown routine

Table ___: *Priority Truth Table of Software Inputs*

Emergency Overrides

There are 2 forms of emergency override:

1. The user presses the brake while the gamepad or autonomous system has the highest priority, which yields control to the driver.
2. The user commands the gamepad while the autonomous system has the highest priority, which yields control to the gamepad.

In the case of the first override, the DBW system will ignore software input and will enter the “Idle” state. Once the gamepad or autonomous button is pressed, software control of the

vehicle will resume. In the second case, the gamepad will take control if an input greater than a threshold is detected, and the gamepad will yield control back to the autonomous controller after the autonomous button is pressed.

Emergency Stops (E-Stops)

There are three redundant full emergency stops:

1. RC transmitter/receiver
2. External E-Stop
3. Mode Select Panel E-Stop

Any of these E-Stops being triggered will transition the DBW system to *Error Mode 3*, in which the brakes are applied until the vehicle reaches a complete stop. The DBW system will then sit idle in this error state until the reset button is pressed. It is important to note that once the vehicle is in the error state, the full state transition from start → setup → operation will need to be performed.

Design Verification

Through the system on a bench implementation of our DBW system, we have achieved functional validation of our safety systems. We can prove that the state machine reacts accordingly to input from the emergency stops and detected input from the gamepad and brake pedal. Without a physical vehicle to test on, however, it is difficult to judge the efficacy of certain aspects such as the shutdown routine. We can prove that we are in fact extending the brake actuator, but without experimental observation of how long it takes the fully extended

actuator to bring the vehicle to a halt, we cannot provide accurate performance specifications to our client.

VII. Steering System

Design Specifications

In order to successfully actuate the steering system of the vehicle, the Drive-by-Wire system needs to act on the vehicle's steering column with the level of speed, strength, and precision that a human driver would. These specifications are outlined below:

- ➢ Maximum Error from Commanded Steering Angle: 1 degree
- ➢ Minimum Time to Reach Commended Angle: 2 seconds
- ➢ Minimum Precision to Commanded Angle: 0.25 degrees

Additionally, the steering subsystem must not hinder the user's ability to control the vehicle using the steering wheel when the Drive-by-Wire system is inactive.

Functional Overview

System-on Behavior: When the Drive-by-Wire system is active, the central controller constantly receives serial data from an input device such as a gamepad. From that data, the central microcontroller passes the commanded steering angle (see Section 5) into a feedback control loop. The microcontroller receives feedback from the steering demo in the form of a linearized voltage from a 10-KΩ potentiometer that is mechanically attached to the steering column. The linearized voltage is fed into one of the central microcontroller's built-in 12-bit analog-to-digital converters. The microcontroller outputs two PWM signals (for directionality) to an IBT_2 dual H-Bridge motor controller (BTS7960) which controls the speed of a brushless 12V motor that mechanically turns the steering column.

System-off Conditions: While the Drive-by-Wire system is active, if the user presses the “OFF” button on the mode select panel or depresses the brake pedal of the vehicle with their foot, the central microcontroller will shut off all actuators and return to the idle state, where control of the vehicle’s steering wheel is yielded to the driver.

Emergency Stop Behavior: If one of the Emergency-Stop buttons is pressed while the system is active, the central microcontroller will apply the brake actuator for five seconds in order to bring the vehicle to a stop, and an alarm will sound to notify the driver that a critical event has occurred, and the system will then shut down into the idle state and control of the vehicle’s steering wheel will be yielded to the driver. The user must restart the Drive-by-Wire system if they desire functionality to be resumed.

Subsystem Error Behavior: As described in Section #5, the central microcontroller constantly checks for system errors, if one is detected, the system will shut down as if it had been turned off manually. The appropriate error message will be displayed on the mode selector’s built in LCD display, and an alarm will sound to alert the driver that control has been yielded to them.

Mechanical design

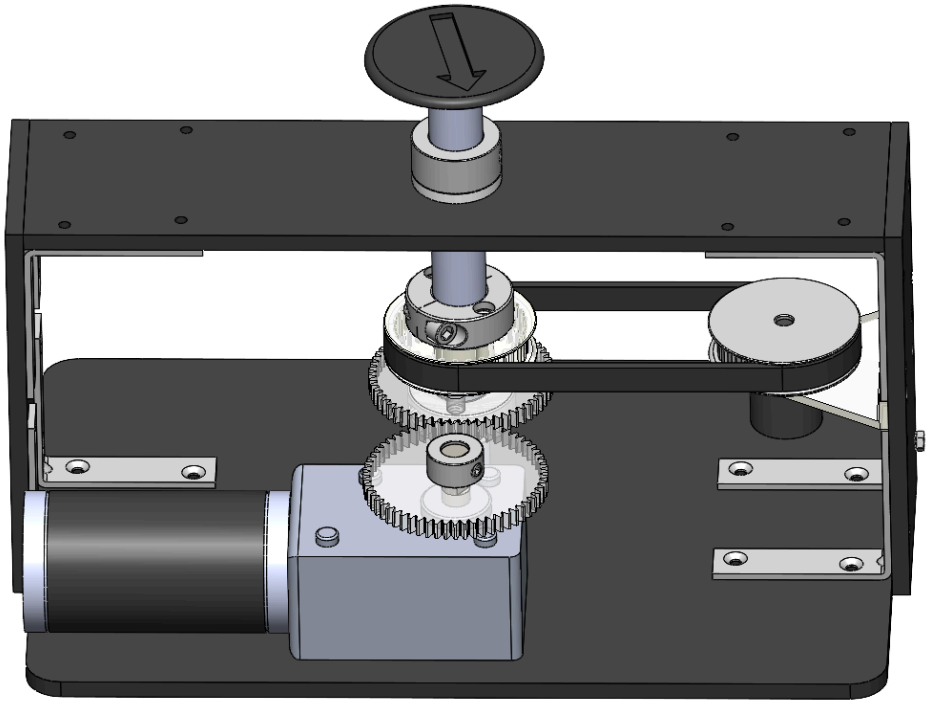


Figure __: *Steering Demo CAD model (isometric view)*

A demo was created to visualize the mechanical authority that the Drive-by-Wire system has on the steering column of the vehicle (Figure ___).

An SLA Resin printed belt sprocket is mounted to a ¾" steering column using a mountable shaft collar. A belt sprocket is mounted to the output of the potentiometer with a set screw. An XL Series Timing belt from McMaster Carr links the potentiometer to the column, and a resin printed bracket tensions the belt sprocket and feedback potentiometer against a frame member.

The structural frame members, gears, and belt sprockets have been laser cut from ¼" black acrylic or resin printed with clear V4. Number 6 countersunk machine screws and steel locknuts fasten the acrylic to eight corner brackets.

In order to simulate an analog, bipolar DC source, an H-Bridge motor drive was used between the 12V DC supply and the rotary actuator. A dual PWM input from the central

controller adjusts polarity, while the duty cycle controls magnitude (as described in section 5). The IBT_2 driver is fastened to our system in a dedicated 3D printed enclosure with the intent to protect it from vibration and ingress.

Software Control

The 5-turn rotary potentiometer attached to the steering column via a belt feeds an analog voltage to a 12-bit ADC (4096 quantization levels) if using the SPC58EC board, or 10-bit ADC (1024 quantization levels) if using the Raspberry Pi 4. However, since the Polaris GEM e2 has a four-turn steering column, the range of motion is restricted to be in the range $[410,\ 3686]$ if using the SPC58EC and $[102,\ 922]$ if using the Raspberry Pi 4. The steering position is commanded in the form of ± 25.00 degrees and scaled to be in the range of motion.

A discrete PID (Proportional Integral Derivative) loop is then implemented in software on the Raspberry Pi 4 to control the position of the column with a sampling rate of 20Hz, with the dynamic model (plant) being approximated as a simple integrator (Figure). The gains were tuned, which resulted in the values of $P = 10,\ I = 1,\ D = 0.2$. Additionally, to account for the limitation of the motor, saturation limits of one revolution per second are implemented.

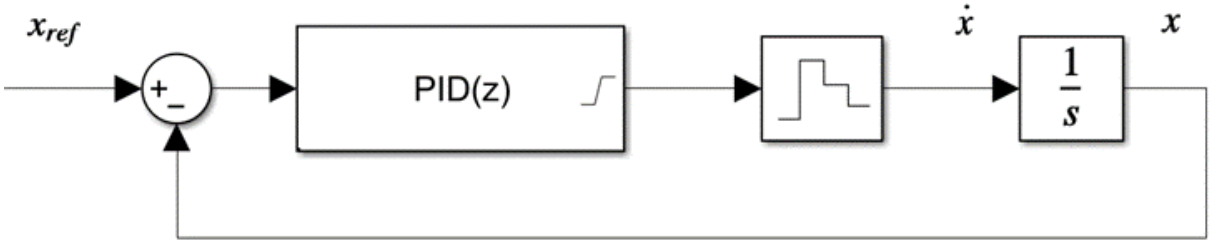


Figure : Discrete PID loop of an integrator

When implemented in software, $\dot{x}$ is scaled to be in the range $[-\ 100,\ 100]$ and used as the duty cycle for the PWM pair controlling the H-Bridge, and x is the potentiometer reading. Figure __ shows the step response of the system when commanded one revolution. The response has a rise time of 1.00s and an overshoot of 1.91%.

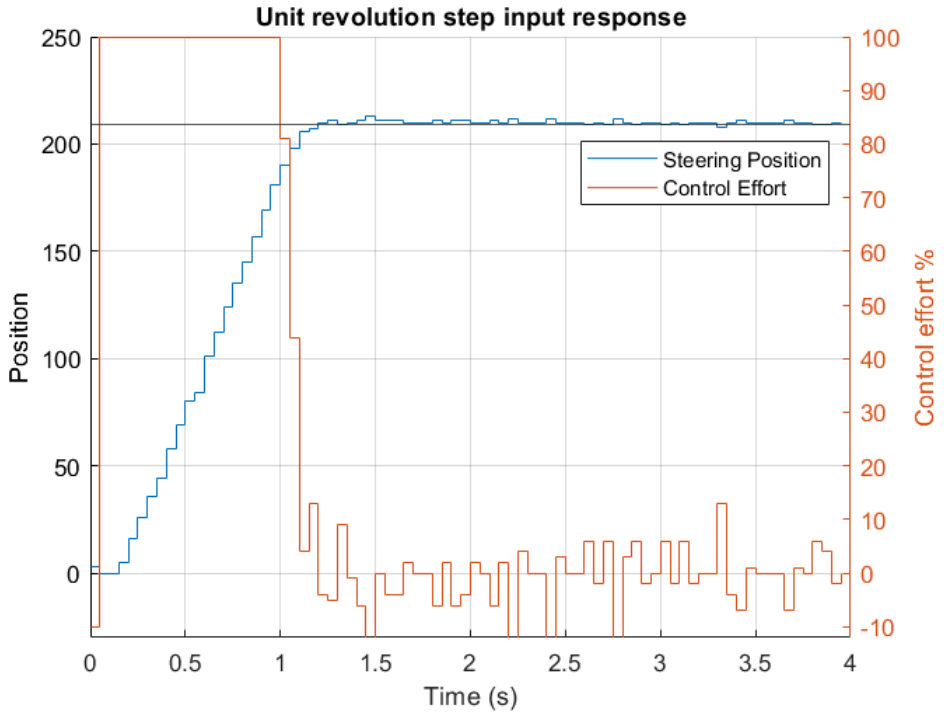


Figure __: *Response of the steering column control loop to a unit revolution input.*

The frequency response of the system was also analyzed to determine how responsive the system was to oscillations in the input. Figure __ shows the tracking of the system for a sine wave input of 0.25Hz, 0.5Hz, and 1Hz. The system had a gain of 0dB, -0.34dB, and -1.06dB, and a phase loss of 13.5°, 27°, and 54° respectively.

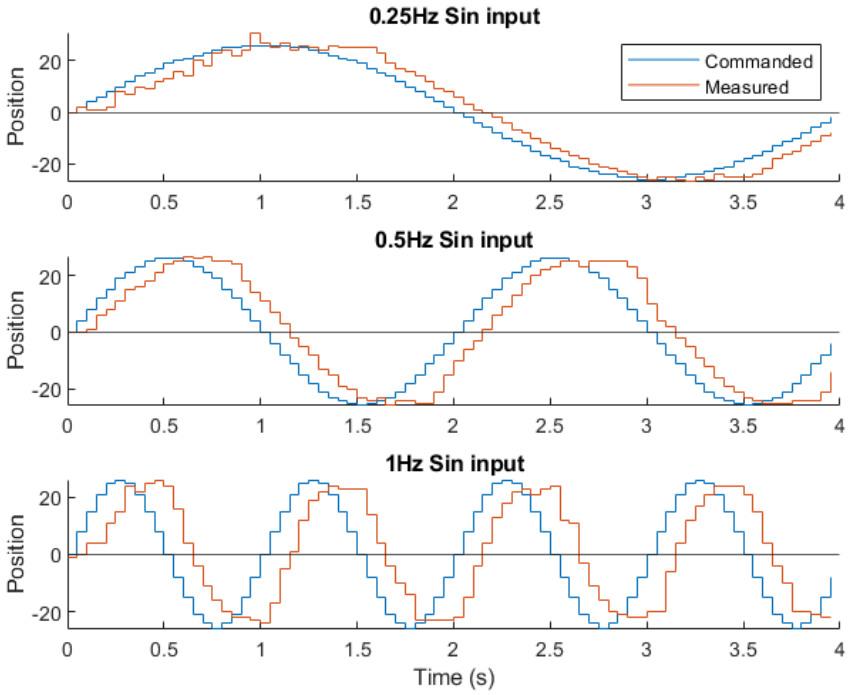


Figure __: Frequency response of the steering system to three different frequency inputs.

The frequency response of the system as the input approached 1 Hz was not desirable, and could be improved with a lead compensator, as the DC gain beyond 1 Hz is negligible. However, the easiest way to improve performance would be to increase the sample rate. On the Raspberry Pi 4, this cannot be achieved due to hardware limitations. The Raspberry Pi 4 has a maximum PWM frequency of 100 Hz, and changing the PWM duty cycle at a rate close to or faster than that frequency would change the RMS voltage that the actuator receives, greatly reducing the stability and performance of the control loop. Additionally, specifying a better actuator would mean that the system could track faster.

Furthermore, while this model may work for the System on a Bench, it is not representative of the actual steering column. The torque to turn steering columns depends on a myriad of variables, such as stiction, speed of the vehicle, and terrain, among other factors. It is

extremely difficult to simulate all of the parameters without an actual vehicle. PID loops have the benefit that they tend to remain stable even if the plant varies from the model. However, the performance of the control loop will vary depending on the dynamics of the actual plant.

Circuit Design

Comprehensive circuitry documentation for the steering system is included in the system wiring diagram (See Appendix)

Design Verification

In order for a vehicle implementation of our system to be considered a success, it must meet several criteria. It must be able to turn the wheels of the vehicle to 25 degrees in each direction. It also has to be able to hold the commanded position to less than 1% error. Figure demonstrates a steady state deviation of 0.48% of the total range, which is within our margin of error. Additionally, it is desirable for the column to be able to rotate one revolution (12.5° change in the wheel angle) in less than 1.5s in order to enable for smooth and decisive steering. With the actuator we chose for the System-on-a-Bench, the steering column can turn at 1rpm. Furthermore, a frequency response of -3dB gain and 30° phase loss are desired at 1 Hz. This system was able to meet the gain margin with a gain of 1.06dB, but the phase loss was not met as the system suffers from 54° phase loss (Figure). Fortunately, at 0.5 Hz, the system did meet the phase requirement.

Next Steps

Manual override for the steering subsystem was not implemented in the System-on-a-Bench. Implementing this functionality on a vehicle would require the use of a

rotary torque sensor in order to detect a user attempting to steer the vehicle. On many vehicles with an Electronic Power Steering System (EPS), this hardware is already present. In theory, the steering subsystem could be fully implemented on a vehicle by simply interfacing with an existing EPS system (likely over CAN), using the pre-existing actuator to perform steering actions when the Drive-by-Wire system is active. Configuring the steering subsystem to function as an EPS module in the System Off state is a potential solution if the vehicle does not have an existing EPS or the EPS module cannot be interfaced with. In order to achieve this, a torque sensor is needed on the steering column to determine the torque and direction the user is putting into the system to determine the amount of compensation needed to help the user. Additionally, a current sensor to the built in EPS module would further aid in estimating the compensation. Otherwise, the steering actuator for the Drive-by-Wire system would need a mechanical disconnect from the steering column of the vehicle so that a user of the system would not have to resist the force of the steering actuator when the system is inactive.

VIII. Braking System

Design Specifications

In order to successfully actuate the braking system of the vehicle, the Drive-by-Wire system needs to act on the vehicle's brake pedal with the level of speed, strength, and precision that a human driver would. These specifications are outlined below:

- ➢ Error from Commanded Pedal Position: 1 percent
- ➢ Minimum Time to Fully Depress Pedal: 0.5 seconds

Additionally, the braking subsystem must allow for a user to press the brake pedal and any time during operation in order to override the Drive-by-Wire system. The system must

relinquish control of the vehicle to the driver within 200 milliseconds of the pedal being depressed. Lastly, the braking subsystem must not hinder the user's ability to control the vehicle using the brake pedal when the Drive-by-Wire system is inactive.

Functional Overview

System-on Behavior: When the Drive-by-Wire system is active, the central controller constantly receives data from the input device. The central microcontroller passes the commanded brake pedal position (see Section 5) into a feedback control loop. The microcontroller receives feedback from the braking demo in the form of a linearized voltage from an automotive standard 4-KΩ brake pedal position sensor that is mechanically attached to the brake pedal. The linearized voltage is fed into one of the central microcontroller's built-in 12-bit analog-to-digital converters. The microcontroller outputs two PWM signals (for directionality) to another IBT_2 dual H-Bridge motor controller (BTS7960) which controls the speed of a feedback linear actuator that controls the depression of the brake pedal.

System-off Conditions: While the Drive-by-Wire system is active, if the user presses the “OFF” button on the mode select panel or depresses the brake pedal of the vehicle with their foot, the central microcontroller will shut off all actuators and return to the idle state, where control of the vehicle’s brake pedal is mechanically yielded to the driver.

Emergency Stop and Subsystem Error Behavior: The user of the vehicle will always have mechanical control of the brake pedal and will be able to override or even overpower the Drive-by-Wire system if necessary.

Mechanical design

Figure __:

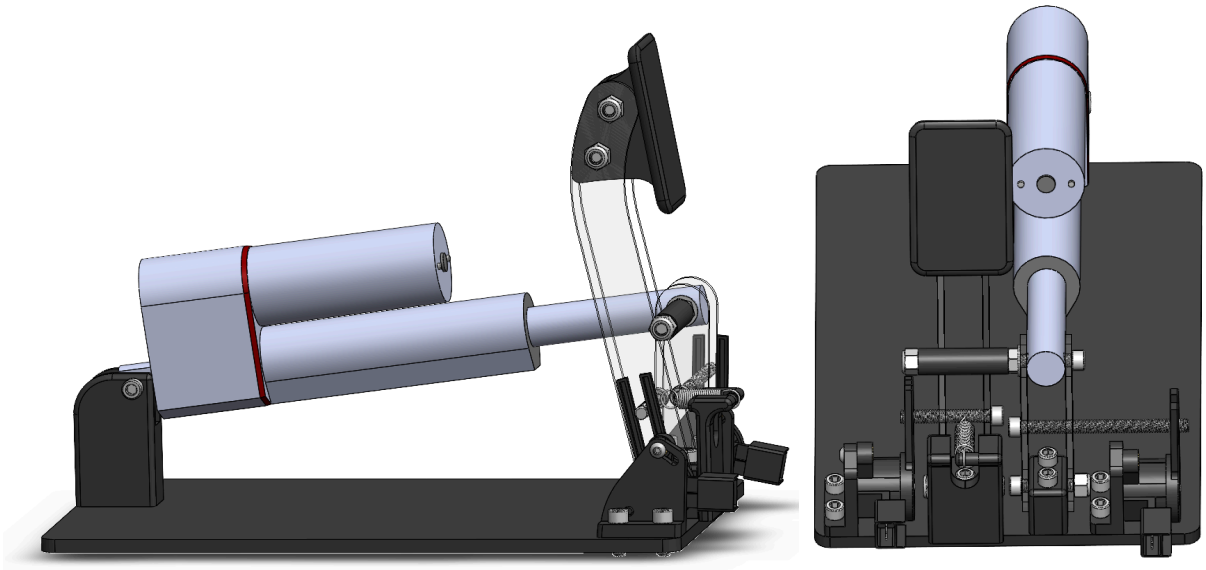


Figure __: Braking demo CAD model. (Left) Side view. (Right) Isometric front view.

A demo was created to visualize the mechanical authority that the Drive-by-Wire system has on the brake pedal of the vehicle (Figure ___).

A placeholder ‘brake pedal’ is made from two pieces of laser cut acrylic and a PLA printed pedal body which is bonded together with epoxy resin. The pedal is mechanically held together by ¼-20 bolts and nylon locking fasteners. The ‘brake pedal’ is mounted to a base plate with resin printed mounts and has a steel compression spring that returns the pedal to an upright position after being depressed.

A feedback linear actuator is also mounted to the base using ¼-20 hardware and resin brackets. An arm of similar construction to the ‘brake pedal’ is attached to the output rod of the actuator so that a contraction of the actuator results in a depression of the brake pedal, but the pedal can also be pressed independently from the actuator (this is important for manual override capabilities).

Two pedal position sensors are mounted to the base and attached to both the linear actuator arm as well as the ‘brake pedal’ so that the two positions can be read independently by the central microcontroller with on ADC input.

The setup for powering the linear actuator is identical to the steering column (Section VII.C). Two PWM channels are used to control another IBT_2 H-Bridge motor driver.

Software Control

Through testing, the limits of the position are determined to be $[0,\ 344]$. The brake position is commanded as an integer from 0 to 100, and scaled to be in the aforementioned range. A second discrete PID loop is implemented in software (Figure __), estimating the plant to be an integrator with saturation limits determined through motor specifications and a plant of with gains of $P = 10,\ I = 1,\ D = 0.2$ determined through manual testing.

Figure __ shows the step response of the brake pedal when commanded to full extension from rest. The response has a rise time of 1.80s with a negligible overshoot and settling time.

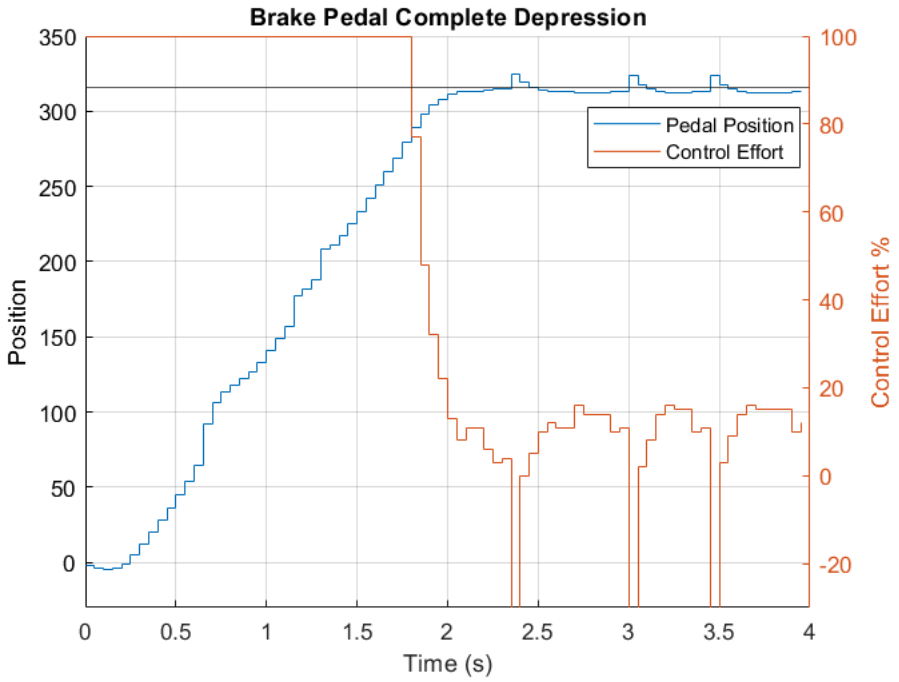


Figure __: *Step response of the brake pedal to full compression.*

Figure __ shows the response of the system from full compression to full extension. The rise time of this command was 1.55s, which is 0.25s faster than the compression command despite traveling the same distance.

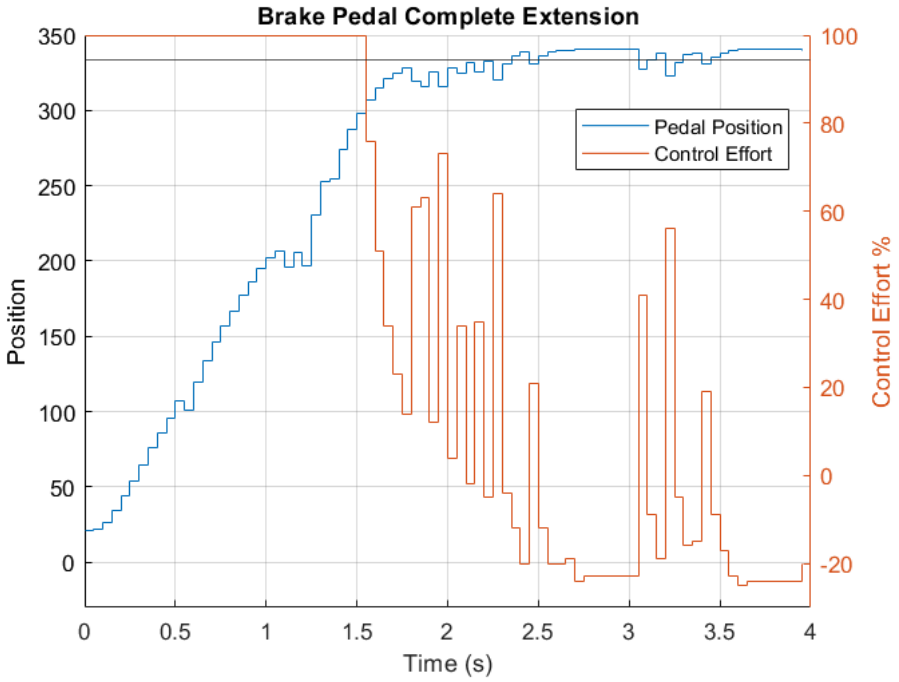


Figure __: Step response of the brake system to full extension.

The difference between extension and compression can be attributed to the dynamic of the spring at the end of the linear actuator. For the system on a bench, a more accurate plant would have been $\dot{x} = C_1 u - (C_2 u)kx$, where u is the input control effort, $C_1 u$ is the velocity at the given control effort with no load and $(C_2 u)kx$ is loss in velocity due to the load caused by the spring. Actual vehicle brake pedals have a much more complex dynamic, as the pedal system consists of multiple spring damper systems [SOURCE NUM]. This system could be approximated as $\dot{x} = C_1 u - (C_2 u)(kx + b\dot{x})$, where $b\dot{x}$ is the load due to the damper and velocity of the system. Since the velocity lost to just the spring (which has a spring constant smaller than anything found in a real vehicle) is nontrivial (i.e $C_1 - C_2$ is not approximately C_1), a better actuator must be sourced before being implemented in an actual vehicle.

Manual override detection was implemented into this subsystem. A second brake pedal sensor is attached to the brake pedal, which is also read with an ADC. A linear relationship between the pedal sensor on the linear actuator and the brake pedal is calculated. If the difference between the estimated position of the brake pedal and the ADC position of the brake pedal exceeds a threshold value, the system determines that the user is manually depressing the brake and cedes control back to manual inputs.

Circuit Design

Comprehensive circuitry documentation for the braking system is included in the system wiring diagram (See Appendix E)

Design Verification

This system does not quite meet the specifications for a vehicle implementation of Drive-by-Wire as outlined in the Criteria for Success Matrix. The system fails to reach full depression in the required time of 0.5 seconds, and cannot hold the commanded position to a maximum error of 1% (Figures __ & __). The “Control Effort” in both figures is at 100% for at least the first 1.5 seconds. This means that the actuator is attempting to move at maximum speed; therefore achieving a faster response with a more aggressive PID controller would yield minimal improvement to the response time. The slow response is an inherent issue with the linear actuator chosen for the system. Additionally, the steady state response of the system fails to converge to the commanded position within the acceptable range of 1%. Noise may cause uncertainty in the position of the brake read by the ADC, which in turn causes the controller to overcorrect. Applying filtering to the brake position signal or isolating the low voltage and high voltage

system better may correct the unwanted noise. The system is able to detect user input on the brake pedal and return to the manual control state. If the user depresses the brake pedal further than the current actuator position, the central controller detects the difference in positions and transfers control of the entire vehicle back to the user.

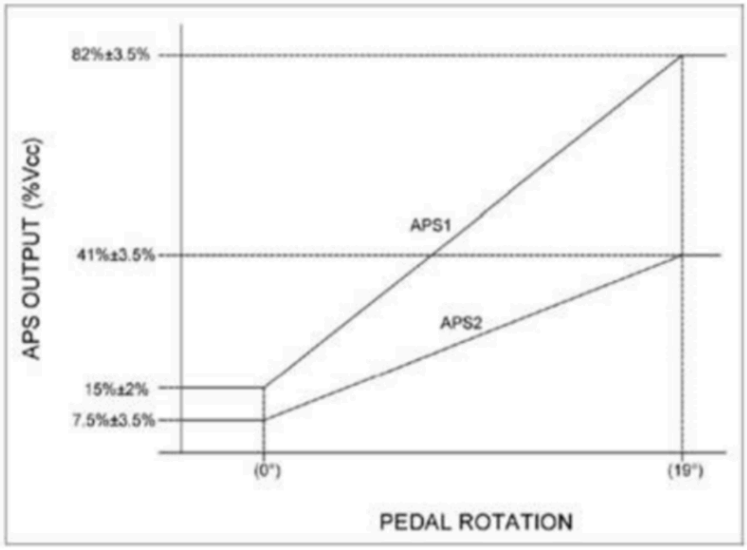
Next Steps

The design for the braking subsystem works for the bench simulation, but due to the actuator speed, is not recommended for use on an actual vehicle. One potential solution would be adding an additional actuator which can fully depress quickly (either pneumatic or hydraulic). The existing linear actuator could continue to be used for precise braking motions, and the secondary actuator would be used for emergency braking. A more expensive and vehicle specific option would use a hydraulic control unit, which could directly control the amount of pressure in the brake systems. This solution would also be able to multiplex input from the braking pedal, allowing for fully integrated manual override. This approach would also contribute to the modular design, since the entire braking system is abstracted from the central controller, meaning it isn't vehicle specific. The issue faced by our team when attempting this approach is the difficulty in acquiring an HCU as well as the complexity of installing and interfacing with one; there are several layers of legislative hurdles that need to be passed in order to implement this solution on a vehicle. Although this is the most technically proficient solution, opting for the secondary fast-actuator is more feasible for implementation.

IX. Throttle System

Design Specifications

Most cars nowadays have a method of digitally controlling the throttle from the accelerator. The Polaris GEM e2 has two accelerator pedal sensors (APS) to control the throttle of the vehicle. Figure __ displays the relationship between the pedal and the APS output. The APS scales linearly from at rest to fully depressed; additionally, the two sensor outputs are scalar multiples of each other, where APS2 is half of APS1 with a margin of error of 3.5%.



Position	APS 1	APS 2	Digital (1)	Digital(2)
0 % pedal	1.08 V	.52 V	27	13
100 % pedal	4.20 V	2.09 V	108	54

Figure __: *The relationship between the depression of the accelerator pedal and the APS output [1].*

In a vehicle, the APS1 and APS2 signal lines leaving the accelerator carry position data (in the form of an analog voltage) directly to the vehicle central controller, which processes this data into the motor / engine throttle. For our modular drive by wire system to control the inputs to the vehicle controller directly, a circuit was devised which can set the output of APS1 and

APS2 to any value in the range shown in Figure __. In order to follow the safety and manual control requirements outlined in section __, the devised circuit needed to have the additional feature that, when disabled, connection of the APS1 and APS2 lines are reconnected to the vehicle controller and the accelerator pedal function is restored.

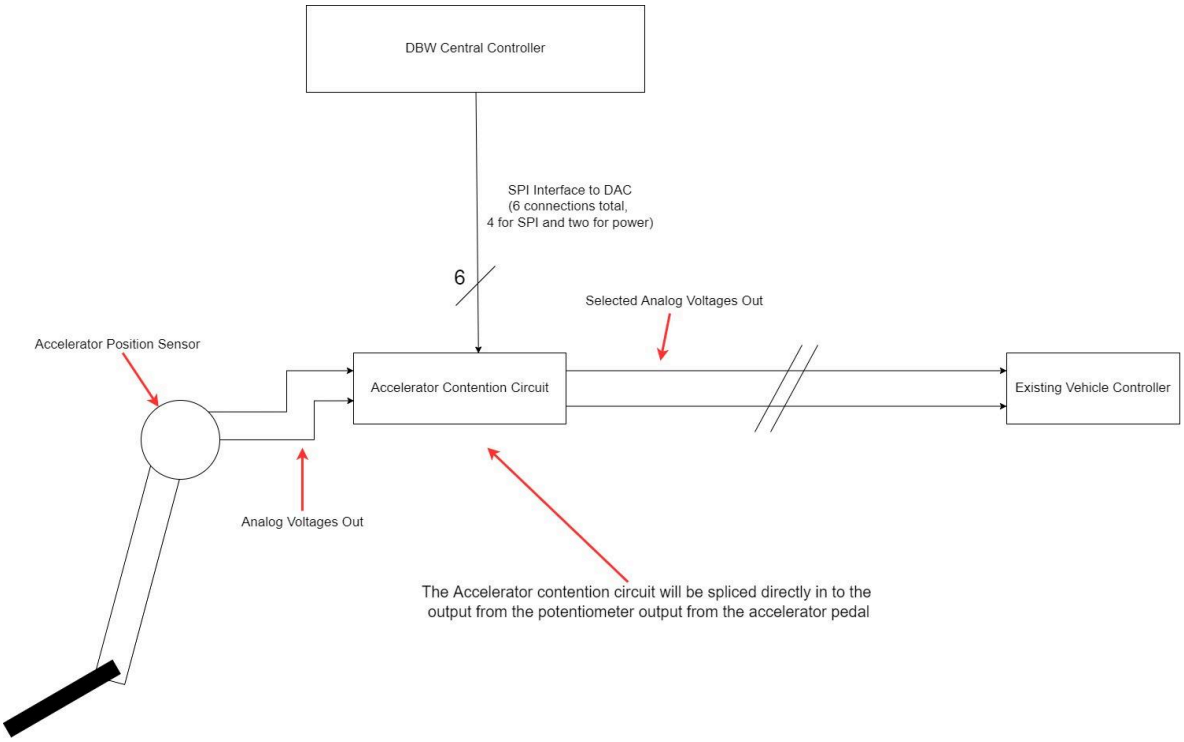


Figure__: *Simplified Block Diagram of the Accelerator Control Methods*

By disconnecting the accelerator pedal output (shown above in Figure __) and instead using control signals sent directly to the vehicle controller, the system will also have the ability to detect manual override by the depression of the accelerator pedal. The APS1 and APS2 signals can be read using an ADC channel of the central controller, and changes in position of the pedal resulting from manual override can be detected as per the safety requirements in Section __.

The circuit which dealt with the ‘point of contention’ caused where the two input signals met and needed to be selected was developed following the above requirements and implemented in the System-on-a-Bench. The circuit has outputs which are settable over an SPI

connection, and the global enable. When the global enable is off, the output of the circuit will be the APS1 and APS2 signals passed through directly, with the rest of the ACC electrically isolated from the vehicle signal lines.

Circuit Design

See the next page for our detailed circuit design for the accelerator contention circuit.

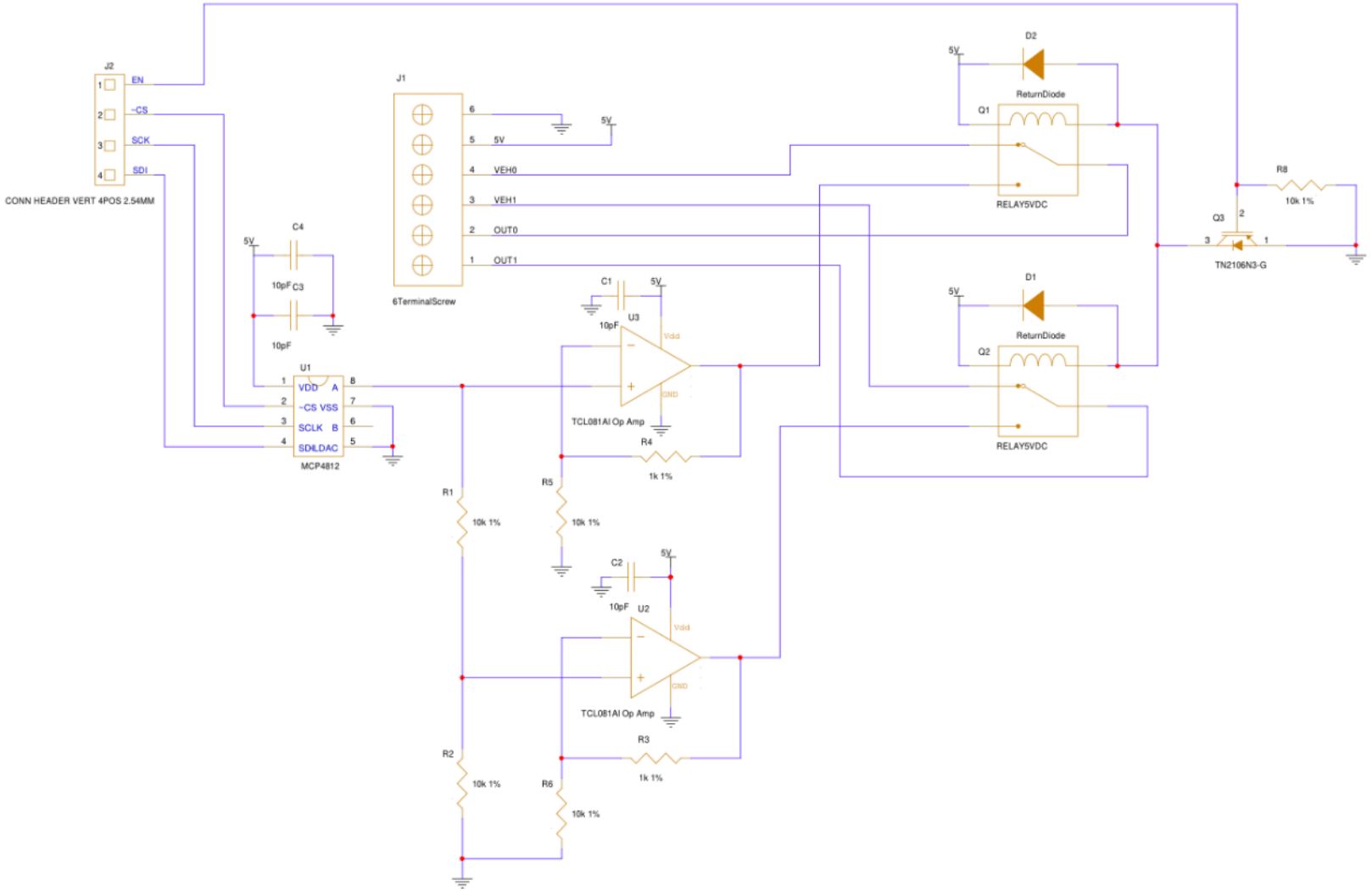


Figure __: *ACC Circuit Schematic*

In the final implementation of the ACC, an MSP4812 10 bit DAC (U1 in Figure) is used to produce an output signal mimicking APS1. This is followed by a 50% voltage divider (R1 and R2), which will produce APS2. In order to isolate the DAC output from the rest of the system, two single supply TCL081 operational amplifiers were used in a non-inverting configuration. Since the maximum output of the DAC that was selected for the final implementation was V_{dd}-1V=4.00V, the op amps implement a 110% gain, using R3-6, so that the maximum reachable value at the output is 4.40V. In practice, any value above 4.2V is ignored by the vehicle controller, since all larger values correspond to a 100% throttle.

The output of the circuit is selected using two signal relays which are powered from the same 5V rail and the DAC and op amps, and can be switched on and off using the global enable. Toggling the global enable gates on the TN2106 MOSFET and allows the relays to switch on. When the global enable is set to a logic low, the relays disengage and the signal from APS1 and APS2 (VEH0 and VEH1 in the diagram) are allowed through the circuit without interruption. The use of signal relays in this manner allows the ACC to be electrically isolated from the vehicle when powered off, so that no errors are detected by the vehicle controller.

The final implementation of this device was routed to a PCB which was used for the SoB testing of the design. The layout includes an unused additional filtering capacitor on the DAC power line, which can be implemented to reduce noise in a specific application or use case (see Appendix L).

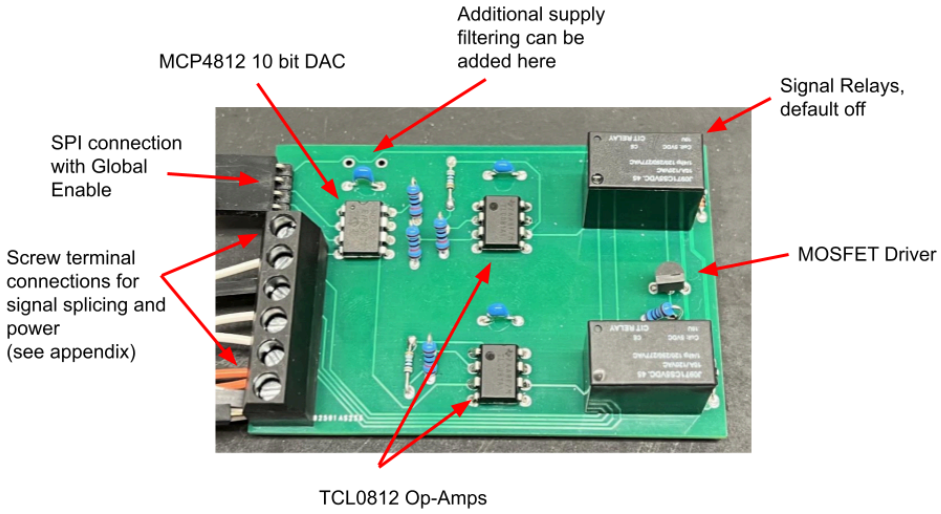


Figure ___: *Final ACC implementation*

Part selections for the final accelerator contention circuit PCB are outlined in Appendix B.

Functional Overview

System on behavior: When the DBW system is engaged by any of the methods outlined in the User Interface section, the global enable will be set to a logic HIGH, and the relays will be engaged. Once this happens, the DAC can be set to any value within the APS output range. Since the DAC has 10 bits of precision, and the actual range of valid output values (from the APS ranges in Figure ___) corresponds to digital values of 244 and 950. Each value in between these two corresponds approximately linearly to a voltage output between 1.08V and 4.20.

System off behavior: When the system is disengaged by the user in any of the methods outlined in the User Interface section, Safety section, or in the event of a power failure, the global enable pin will become logic LOW. In this case, the relays will not be powered, and will be restored to their mechanical equilibrium state, allowing the APS1 and APS2 channels from the vehicle accelerator pedal to pass through to the output of the ACC unimpeded. The relays also provide

isolation between the ACC when it is in the off state and the vehicle electrical system, because there is no linkage between the two systems.

Error reporting: Errors involving the DAC and the ACC can be detected by the central controller by the use of several ADC channels to read the inputs and outputs of the ACC. This enabled the central controller to detect changes in the accelerator pedal position, which was represented using a potentiometer, detecting manual override. It also allowed for the DBW system to run its mid operation checks to ensure that the output of the ACC was actually the desired output. This was determined by the team to be the best method of mid operation checking, because it was representative of the way that the vehicle controller would read the APS1 and APS2 inputs. The DAC connection cannot be checked over SPI interfacing because the DAC does not respond over serial, but does change it's output according to input it receives.

Software Control

The MCP4812 is a 10 bit digital to analog converter (DAC) which uses SPI communication with the central controller to set the output voltage. As the APS signal is constrained to the range [1.08, 4.20] volts, the full range of the DAC is not used. If the system is in software control mode, the global enable gets switched on and the contention circuit selects the DAC's output as the signal to pass to the vehicle. The central controller receives a desired throttle percentage in the range [0, 100], scales the value to be in the APS range, converts the voltage level to quantization levels, and sends a message to the MCP4812 to update the output voltage. If the system is in manual mode, the global enable is off and any output of the DAC is overwritten by the APS.

Due to the possibility of corruption due to noise on the communication channel or a physical disconnect of the ADC from the central controller, a mid operation check is employed. In software operation mode, the output of the contention circuit is read by an ADC. If the read value of the ADC differs from the commanded value of the DAC for multiple consecutive reads, then a mid-operation error event is triggered.

Additionally, the voltage of the APS output is sampled periodically with and ADC. If the value of the APS signal rises above the no depression voltage of 1.08V, then the central controller interprets this as the user attempting manual override and cedes control to the user.

Design Verification

Functional testing of the ACC included observing the output in response to a commanded a ramp, and verification of the Global Enable switching behavior. Applying a regulated 5V to the power input of the ACC and toggling the global enable, we were able to verify that the relay driver circuit worked as designed. We also ran the circuit with the enable toggled HIGH for several hours, but the power draw remained stable and the relays did not overheat.

In order to determine that the output of the ACC was valid through the whole digital input range, we additionally applied a ramp input and observed the output voltage using an oscilloscope. Shown below in Figure __, where the ramp is set to run from 0.52V to 4.25V. The initial downward spike was a result of the scope looping, and the measurements before -0.5s can be ignored.

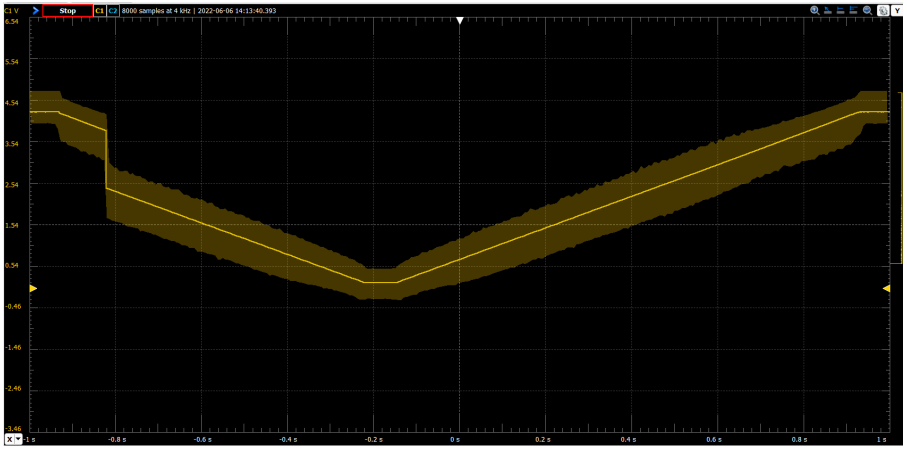


Figure ___: *Initial ramp testing with ACC*

As can be seen in the initial test, there is consistent non-negligible noise on the output channel of the ACC, which might cause control issues when the channel is being read. To determine the source of the noise so that it can be prevented in the future, further testing was performed with and without the system on a bench power supply.

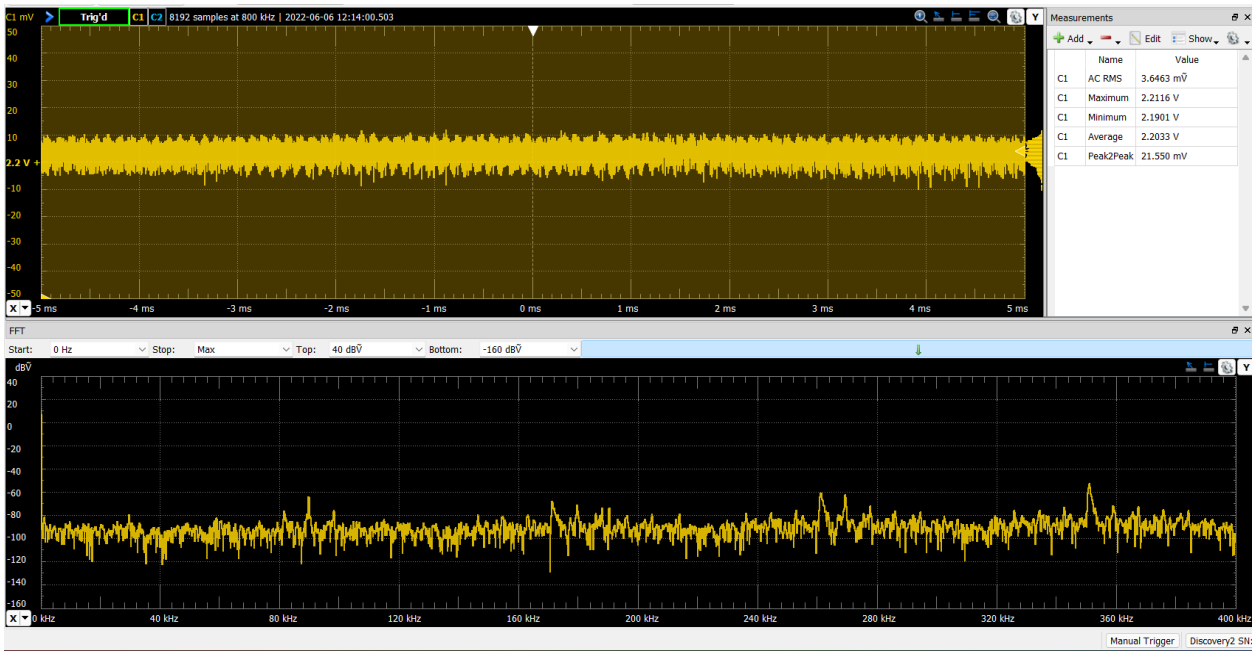


Figure ___: *ACC Output Commanded to 2.20V with Power Supply On*

In this view, with a constant commanded value at the output of the ACC, the content of the noise on the output line becomes more clear. In the figure above, the Fourier analysis of the signal shows a major 60Hz component of noise, followed by several other peaks at 90kHz, 170kHz, 260kHz, 270kHz, and the largest peak at about 350kHz. The 60Hz fundamental is likely a residual from the 5V supply used to power the central controller, which in turn regulates the voltage and outputs it to the ACC, but the other high-frequency components are unaccounted for. The peak to peak of 21.55mV that was measured is a very large variation on a system with little inherent noise. Disabling the larger SoB power supply, but leaving the power line to the ACC intact had a major impact on the output voltage stability, shown in the figure below.

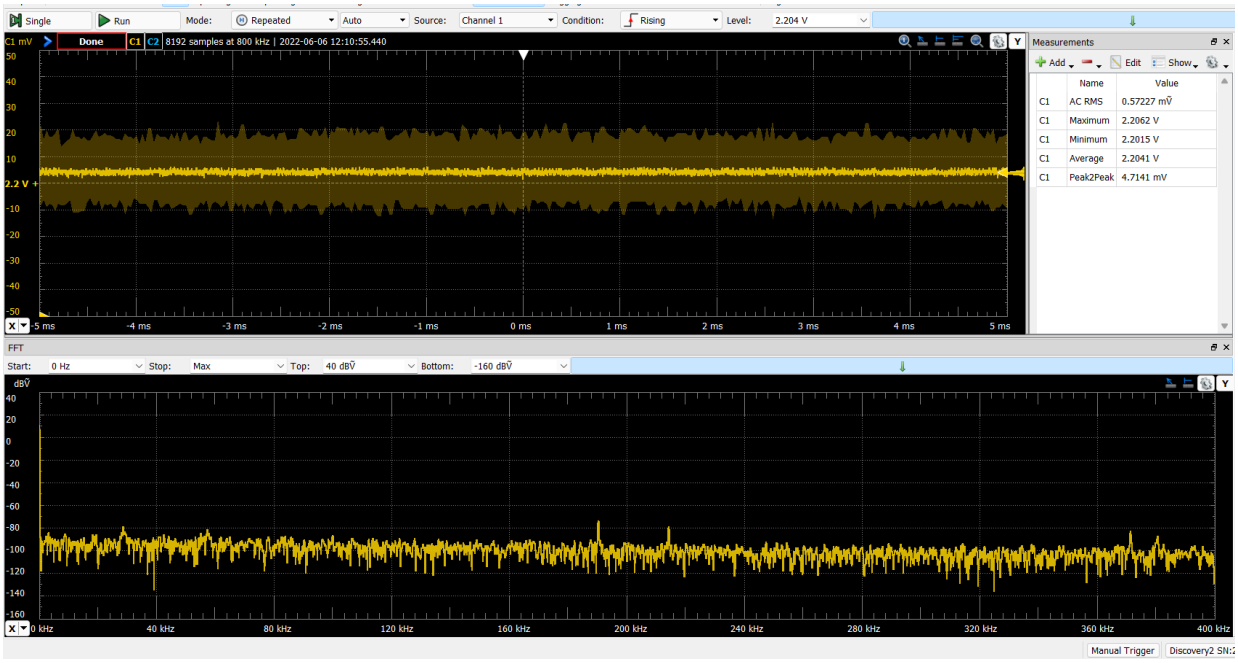


Figure __: *ACC ACC Output Commanded to 2.20V with Power Supply Off*

In this case, with the commanded output unchanged but the 12V SoB power supply disconnected, the noise is grammatically reduced. The peak-to-peak variation changes from 21.55mV to 4.71mV. In both cases, the output variation peak to peak was less than our

requirement of 3.5% error, but simple filtering could greatly reduce the output noise. Disconnecting the SoB supply alone reduces the peak-to-peak noise by 78.1%, and also entirely removes the 90kHz, 260kHz, 270kHz, and 350kHz harmonics. The only conclusion to draw from this is that the power supply itself is producing (or causing other systems to produce) the excess noise. The output of the ACC could be improved with a low pass filter to remove the high frequency noise, followed by a band reject filter to remove the 60Hz component as well.

The ACC is, without a doubt, not noise resistant and will need to be redesigned with these findings taken into account before it can be considered safe to implement on a vehicle.

X. Peripherals

In addition to being able to drive the vehicle, the Drive-by-Wire needs to be able to interface with different vehicle peripherals. Vehicle peripherals are designed to enhance vehicle safety and improve functionality, so it is important for the Drive by Wire system to be able to interface with them. Both the top level controller and the joystick can control various peripherals with the peripheral bus portion of the command packet.

The vehicle's peripherals are controlled by independent 12V electrical connections running throughout the vehicle. Our system uses a number of power relays that are connected to ground and controlled by various GPIO pins on our central microcontroller (outlined in Appendix F). These relay circuits will be spliced into an existing vehicle wiring harness in parallel so that each peripheral can be controlled by our DBW system or by standard vehicle control.

Peripherals Interfaced:

- Horn
- Wipers
- Reverse
- Turn Signals
- Headlights

Design Verification:

As mentioned earlier in this section, having proper access to the peripheral systems of a vehicle is essential to the safe operation of a vehicle. As each of the peripherals on the system-on-a-bench is controlled by a GPIO toggle, testing was conducted via simple audio and visual checks. Unfortunately, the system-on-a-bench does not include representations of the windshield wipers or the reverse-toggle, so these systems have yet to be verified.

XI. Conclusion

Success Criteria Verification

In the Criteria for Success Matrix (Appendix ___), metrics are set for considering what a successful implementation should be able to accomplish. The central controller can successfully communicate with the joystick controller and a top-level controller to parse commands. However, the central controller was not able to display the current state to the user due to corruption in the display message. Additionally, returning states and errors to the top level controller was never tested as an autonomous controller was never designed.

The central controller is able to run startup and real time tests to ensure that the subsystems are operating as intended. The controller checks the braking, steering, and throttle subsystems in the startup phase, and is currently capable of checking the throttle and communication subsystems during operation. Braking and steering mid-operation checks are not implemented on the System on a Bench.

The steering subsystem is able to track step inputs of unit revolutions within one second and less than 1% error in steady state. For a sinusoidal signal at 1Hz, the steering subsystem is able to keep a gain greater than -3dB, but fails to keep the phase loss to less than 30°. Additionally, manual override is not implemented in the system on a bench.

The braking system is not able to meet the command input response specifications in the Success Criteria Matrix. The system takes longer than 0.5 seconds to fully depress and does not hold the desired position to less than 1% error. However, the braking subsystem successfully detects user input and adjusts the central controller's state respectively.

The throttle subsystem is capable of commanding the output of the simulated APS to well below the 3.5% margin outlined for the Polaris GEM e2. Additionally, the user can press on the accelerator pedal, and the system is capable of detecting the manual override.

Finally, the System on a Bench is able to digitally control various peripherals such as the headlights, horn, and individual turn signals.

Letter to Future Teams

The system-on-a-bench implementation of the DBW system was valuable in designing the firmware and safety systems for the central controller, which is the layer in between autonomous control algorithms and the actual vehicle. These systems have been thoroughly tested and likely shouldn't have to change from here on out. The steering and braking actuation

needs to be improved; the current solutions are slow, unsafe, and don't meet the criteria of success required to place them on an actual vehicle.

When starting work on the system-on-a-bench, teams should begin by thoroughly understanding *how* our system works: this includes thoroughly running through the state machine, paying close attention to what is happening in each stage. The state machine diagram and software-process graph are a valuable resource for gaining this understanding.

The first step in addressing our team's shortcomings should be transitioning our codebase from the Raspberry Pi 4a to the SPC58EC-DISP microcontroller. To learn more about why we were unable to use the SPC58, see the Lessons Learned (Section). Since the software suite (state machine, safety checks) are all written in C, this transition should only really require work to be done on the driver level. This means that all SPI, input capture, timers, and serial I/O will need to be written using SPC5 Studio to program the SPC58 board. We have provided resources to familiarize future teams with the basic workflow of SPC5 studio in Appendix , and with instructions on how to set up drivers in Appendix . On our Gitlab page, the project "Onboarding for SPC5 Studio" provides sample workspaces which test out various drivers including SPI, PWM, serial, and input capture.

Once the code base has been shifted from the Raspberry Pi to the SPC58 (this should hopefully take a month at most), the previous functionality of the system on a bench should be verified based on observations made at the beginning of the project. Converting wiring from the RPi to the SPC58 will take time, so teams should be cautious, careful, and methodical. When the functionality of the system bench post-SPC58 is the same as pre-SPC58, take a moment to relax and take pride in your accomplishment.

The next step is to implement dual software control with both the gamepad and autonomous controller. Since the Raspberry Pi only had a single serial channel, we were unable to achieve this, and thus the gamepad override during autonomous control was never fully tested. The SPC58 provides several serial channels, and adding a second channel after implementing the first should be fairly trivial. Be careful of pin placement, and consistently document which pins you are using on the pinout spreadsheet.

The team should now be free to take this project wherever they or the client wants: the SPC58 automotive microcontroller provides a great deal of power. Should they decide to focus on an autonomous controller, they can use the API calls described in packet_builder.h, which abstracts the work and allows them to focus only on the autonomous capabilities. Should they decide to implement hardware in the loop (HIL) simulation, they can run the steering and braking potentiometer data through physics simulators to emulate actual autonomous driving. Should they decide to redesign the steering subsystem to implement electric powered steering, the interface with the central controller should be trivial due to the sheer number of IO pins that the SPC58 has. We hope that the platform we have developed incites creativity, and provides the opportunity for future students to learn about autonomous driving, perception, planning, and control. The following section consists of specific ways in which we think the system could be improved.

Implementation-Level Next Steps

Subsystem Microcontrollers: As discussed in previous sections, it has been highlighted that abstracting the vehicle-specific embedded functionality away from the central controller allows for improved user functionality with the system, and also allows the safety systems to be set to a higher standard. For the steering subsystem, this would mean a microcontroller is dedicated to

commanding the angle of the steering column, and receives this steering data from the central controller. This subsystem microcontroller would also take care of the mid-operation checks to verify that all electromechanical components are functioning as expected. This would be done similarly with the linear actuator of the braking subsystem. This approach would mean that the central controllers can be duplicated, and then individual teams could design methods of actuating the steering, braking, and acceleration of their particular vehicle. After this setup, they would simply need to connect their subsystem microcontrollers with the central controller, and the DBW system is complete. The issue with this approach would be latency from increased communications: an additional layer of serial communications is added between the autonomous controller and the actuators, and delay is extremely costly in control systems. Additionally, it adds another layer of liability; there is potential for the subsystem microcontrollers themselves to have errors, and the central controller must poll for these errors. It cannot be decided yet whether subsystem microcontrollers would truly benefit the DBW system.

CAN (Controller Area Network) Based System: Rather than designing custom methods of actuation for each subsystem, research groups could alternatively control the vehicle using actuators already on the car, namely the electric power steering and built-in hydraulic control units of the vehicle. This would require understanding how the vehicle's main microcontroller (which is referred to as the Vehicle Control Module (VCM) on the Polaris Gem), communicates with these units over the CAN bus. In order to accomplish this, intensive testing and analysis must be performed to understand the message structure, frequency, and other communication properties. Once this has been completed, these signals can be produced by the central controller (or subsystem microcontrollers capable of CAN communications), and sent to the EPS and HCU. This would then allow for control of the steering and braking systems of the vehicle without the

need for external actuation. The constraint of this solution is that a target vehicle is *required* to have EPS and an HCU (which limits the range of target vehicles). Additionally, implementing a safe method of multiplexing manual and software input to the steering column is tricky, and may not get approval by the UCSC safety administrators.

Ethernet Communications: Early in the project, it was identified that ethernet is the best method of communication for an autonomous controller. Many sensors use ethernet to communicate (the Hokuyo LiDAR for example uses Ethernet 100 Base-TX for data transmission), so adding the central controller to the ethernet bus simplifies communication. Additionally, ethernet is capable of much faster transmission rates than UART (10-40,000 Mbits/sec vs 300-230400 bits/sec). Therefore, the decision to use ethernet was driven by expectations of the client's needs later in the project (when they are implementing sensors for autonomous driving). Due to difficulties in establishing ethernet communications, we opted to use UART for the proof of concept of our communication systems. Switching to ethernet should be a priority in progressing further with this project.

Multi-Controller Input: It is common for autonomous vehicle researchers to test multiple algorithms at a time, toggling between these algorithms mid-operation. The current DBW system doesn't support this capability, both through software and the mode select panel. Adding the software implementation is fairly trivial, this just requires serial pins for each controller. One potential design for a next iteration of the mode select panel would add several more buttons, and have them connected via software. These buttons could then be used for whatever an implementer desires, whether it be switching to a different autonomous controller, or engaging a peripheral.

Graphical User Interface: The current system-on-a-bench relies on a joystick connected to the system via an arduino as the sole source of command inputs. While a basic graphical user interface (GUI) had been identified as a developer-friendly means of testing user commands to the system, as well as reading in feedback from the system, it had been deprioritized when the team decided to shift from ethernet to UART communication protocol. Finishing the implementation of a GUI for testing has the potential to save future developers a significant amount of time when it comes to debugging communication-related issues.

Lessons Learned

Issues our team ran into

- SPC board exploded due to bad parenting (not our fault)
- SPC board jr exploded due to ethernet voltage level difference (our fault)
- SPC debugger exploded due to interference by God

What can be improved

- Better noise resistance
- Improved layout separating HV and LV systems

Personal opinions/recommendations for future teams

XII. References

[1] M. B. Salfer-Hobbs, “Acceleration, Braking, and Steering Controller for Polaris GEM e2,” thesis, 2019.

[2] Dataspeed Inc, *By-Wire Kit for Autonomous Development*. Dataspeed Inc, Rochester Hills, MI, 2020.

[3] N. Bender, S. McGuire, et al, “Projects–Smartbases,” *Build Things*. [Online]. Available: https://nikolaasbender.github.io/. [Accessed: 31-Mar-2022]

[4] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4456887/

[5] https://newsroom.st.com/media-center/press-item.html/p4269.html

XIII. Appendices

A. Block Diagrams

B. Parts Selection

   a. Datasheets

C. Budget

Below is a proposed budget for the Drive-by-Wire Project, including vehicle implementation:

MECHANICAL			ELECTRICAL			OPERATIONS		
Section	***Name***	***Cost***	***Section***	***Name***	***Cost***	***Section***	***Name***	***Cost***
Steering			Control System			Tools		
	Motorized Actuator	$600.00		Main Microcontroller	$355.00		Soldering Station	$80.00
	Chain and Sprocket	$100.00		PSOC Controllers	$60.00		Crimpers	$100.00
Braking				Joystick Controller	$30.00		Multimeter	$30.00
	Hydraulic Actuator	$600.00		IMU Module	$45.00		Digital Calipers	$30.00
	Hydraulic fittings	$200.00	Brakes				Crane Scale	$40.00
	Hydraulic Tubing	$100.00		Pedal Position Sensor	$20.00		Hand Tools	$300.00
Hardware				Additional Connectors	$10.00		PPE	$100.00
	Fasteners	$500.00	Steering				Analog Discovery 2	BELS
	Positive Retention Nuts	$50.00		Wheel Position Sensor	$60.00		Hot Knife	$50.00
Stock	1/8" Aluminum Sheet Stock	$70.00	Accelerator				Heat Gun	$50.00
	Misc Mounting Stock	$200.00		Additional Connectors	$10.00		Torque Wrenches	SLab
Misc			Hardware Switch			Consumables		
	PLA Filament	$50.00		Switch	$20.00		Butt Splice Connectors	$50.00
	Misc Mounting Hardware	$100.00	Misc				Heat Shrink	$100.00
				Waterproof Connectors	$300.00		Tape	$20.00
				Digital Cables and Adaptors	$50.00		Cable Ties	$20.00
				Gauged Copper Wire	$200.00		Cable Sheathing	$50.00
				E-STOP Button	$40.00		Dielectric Grease	$10.00
	TOTAL	$2,570.00		TOTAL	$1,200.00		TOTAL	$1,030.00
							PREDICTED PROJECT TOTAL	$4,800.00

The actual project budget for this project was tracked an an expense tracking document managed by HSL, the link to which is attached below, it also includes information on issued reimbursements.

https://docs.google.com/spreadsheets/d/12uSbW8P5HtIJn0ZiAjUDST9MpDI8SHByi_CQQ3cgVDg/edit?usp=sharing

D. Mechanical Drawings and CAD Files

All mechanical drawings and CAD files are included in the CAD section of the project GIT repository (see Appendix J for details and access).

https://git.ucsc.edu/drive-by-wire/cad.git

E. System Wiring Diagram

F. Power Budget

HSL Drive by Wire project power budget

This document is representative of the power budget and requirements for the full DBW system to be designed by the capstone team. It should be updated and adjusted when new parts are added or finalized in the design, and a new version created each day during which changes are made.

CHANGELOG

V.1 2/23/22: Implemented version control, entered datasheet values and time in state estimations

V.2 2/24/22: Added description, ACDC and DCDC power supplies along with efficiency estimates for preliminarily selested components

V.3 2/24/22: Clarified purity, filled out Time in State as best as possibe with current understanding
1- Measurements for all parts beginning on 2/24
2- Waiting for time in state to be finalized
3- Power supplies are not yet finalized, other items in power budget will guide selection. Estimates shown are for preliminarily selected items operating at minimum and maximum efficiency

V.4 3/5/22: Added System on a Bench Specific budget for power supply selection
4- Added second row for supply voltages to clarify confusion regarding devices with multiple supplies.
5- Alarm buzzer and Steering motor do not have any outputs

V.5 6/3/22: Fixed erronious calculations. Removed approximations made for external sensor systems. Added Arduino Uno used for joystick reference commands

Legend:	Finalized and selected part	Powered directly from microcontroller	Measurement TBD
	Missing datasheet	***Part not yet finalized, see 3***	

Specific to the system on a bench:

Item	Title	Quantity	Supply Voltage (V nominal)	Supply Voltage (V range)	Secondary Supply Voltage (V nominal)	Secondary Supply Voltage (V range)	Output Voltage (V range)	Spectral Purity	State Based Current from datasheet (mA)			State Based Current Measured (mA)			Time In State (duty cycle or % of operational time)			Total Consumption	Max Power (mW)
									Peak	Rated	Low	Peak	Low	Off	High	Low	Off		
Rpi4B	Central Controller	1	5	4.75-5.25	***see 4***	***see 4***	3.3-5		3000	2400	480								
MCP3008	Analog to Digital Converter	1	3.3	2.7-5.5	3.3	3.3-7	3.3-5		0.3	0.2	0.1								
3459 10 Turn Potentiometer	Steering Angle Sensor	1	5	5	***see 4***	***see 4***	0.05-5	High, may need input filtering	0.5	0.5	0.5								
RP3508	LED Backlit button	3	5	0-125	5	0-125	0-125		15	15	0								
TN0702	Power MOSFET	1	12	0-20	***see 4***	***see 4***	0-19.3												
PKM17EPP-2002-B0	Alarm Buzzer	1	12	0-25	***see 4***	***see 4***	***see 5***		45	45	0								
FA-GM-82-12-96	Steering Motor	1	12	unlisted	***see 4***	***see 4***	***see 5***		37130	8083	1869								
FA-PO-35-12-XX	Braking Actuator	1	12	unlisted	5	0-12	0-3.3		5000	2850	900								
BTS7960	Motor Controller	2	3.3	3.3-5	12	6-27	6-27		2000	1000	2								
LYKREE24053.5	12V-5V step down	1	12	8-40	***see 4***	***see 4***	5		250	150	2								

	Total Power:	12V supply			5V supply			
		Peak (W)	Rated (W)	Low (W)	Peak (W)	Rated (W)	Low (W)	
		557.1	157.536	33.3	15	12	2.4	

Item	Title	Quantity	Supply Voltage (V nominal)	Supply Voltage (V range)			Output Voltage (V range)	Spectral Purity	State Based Current from datasheet (mA)			State Based Current Measured (mA)			Time In State (duty cycle or % of operational time)			Total Consumption	Max Power (mW)
									High	Low	Off	Peak	Low	Off	High	Low	Off		
Microcontroller	Raspberry Pi	1	5				3.3	<15mV variation	3000	480	unspecified	***see 1***			100%	0%	0%	***see 2***	15000
	SPC58 (incomplete datasheet)	1	5	3-5.92			5	<15mV variation	2500	350	unspecified				100%	0%	0%		12500
	Arduino Uno	1	5	5-12			5		200	200	unspecified	200	100	~10	100%	0%	0%		1000
Steering subassembly	Steering Motor	1	13.5	not specified			N/A	<15mV variation	29620	7407	unspecified			0	5%	45%	50%		411000

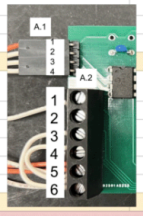
	Steering Motor Controller	1	5	3.3-5			3.3 or 5	<15mV variation	unspecified	unspecified	unspecified			1	100%	0%	0%		
Braking subassembly	Braking Actuator (linear)	1	13.5	not specified			N/A	<15mV variation	4440	2660	unspecified				25%	25%	50%		60000
	Braking Actuator (secondary method)	1																	
	Braking Motor Controller	1	5	3.3-5			3.3 or 5	<15mV variation	unspecified	unspecified	unspecified			1	100%	0%	0%		
Accelerator subsystem	DAC	1	5	5			0-5	<15mV variation	80	20	0				<1%	>99%	0%		400
	Accelerator Contention (74HC4053D,653)	1	5	not specified			-5 to 5	<15mV variation	unspecified	unspecified	unspecified				<1%	>99%	0%		
Sensors	Braking position sensor	1	13.5	0-500			0-12	HIGH, may need additional filtering	135	~1	unspecified				50%	50%	0%		2000
	Steering angle sensor	1	13.5	0-500			0-12	HIGH, may need additional filtering	135	~1	unspecified				50%	50%	0%		2000
	Gear Shift Contention	1	13.5				12	<15mV variation	14	14	14				100%	0%	0%		182
Power Supplies	DCDC step down (48->13.5), vehicle only	2	48	45-52			13.5	<15mV variation	844	210	0				dependant on other duty cycle	dependant on other duty cycle	dependant on other duty cycle		40500
	DCDC step down (13.5->5) vehicle only	1	13.5	10-15			5	HIGH, may need additional filtering on output	110	28	0				dependant on other duty cycle	dependant on other duty cycle	dependant on other duty cycle		1485
	ACDC 13.5v supply	1	120vAC	105-125vAC			13.5	<15mV variation											input
Peak Power:	505567	mW																	
Items below are to be provided and selected by HSL, but will need additional power. If there is no set list at next meeting on 3/1, a total budget of 100W sustained (at 13.5V) or 50W (at 5V) will be allowed.																			
Top level (no design, just power)	High level controller	1		5		not specified	N/A		4400	2000	~0								22000
	Additional sensors	1		unknown		unknown	N/A												
	Router	1		unknown		unknown	N/A												

G. Power and Signal Harnesses

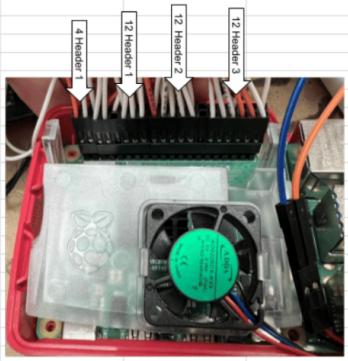
a. Harness Diagram

b. PinOut Table

Connector Title	Description	Pin number	Pin names	Pin diagram
	Accelerator contention SPI	1	Enable	
		2	~CS	
		3	SCK	
A.1		4	SDI	
	Accelerator Contention Screw Terminals	1	OUT1	
		2	OUT0	
		3	VEH1	
		4	VEH0	
		5	5V	
A.2		6	GND	
	Central controller power	1	12VDC	
C1		2	GND	Unimplemented



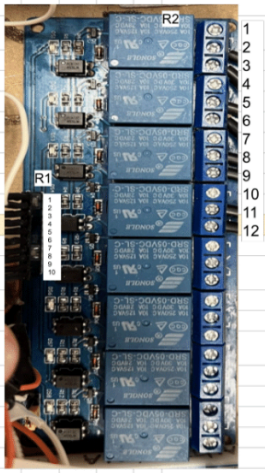
		1	3V3		wPi	BCM	Inum	mum	BCM	wPi	Function	
		2	5V		--	3V3	1	2	5V	--		
		3	SDA	LED 1 (out)	8	SDA	3	4	5V	--		4 Header
		4	5V	LED 2 (out)	9	SCL	5	6	GND	--		
		5	SCL	LED 3 (out)	7	4	7	8	TX	15	TX	
		6	GND		--	GND	9	10	RX	16	RX	
		7	4	Mode Select 1 (in)	0	17	11	12	18	1	E-Stop 1 (in)	
		8	TX	Mode Select 2 (in)	2	27	13	14	GND	--		
		9	GND	Mode Select 3 (in)	3	22	15	16	23	4	Headlights (out)	12 Header 1
		10	RX		--	3V3	17	18	24	5	LEFT_TURN (out)	
		11	17	MOSI	12	MOSI	19	20	GND	--		
		12	18	MISO	13	MISO	21	22	25	6	RIGHT_TURN (out)	
		13	27	SCLK	14	Sck	23	24	CE0	10	ADC	
		14	GND		--	GND	25	26	CE1	11	DAC	
		15	22		30	EED	27	28	EEC	31		12 Header 2
		16	23	Alarm (out)	21	5	29	30		--		
		17	3V3	PWM1	22	6	31	32	12	26	GLOBAL ENABLE (out)	
		18	24	PWM2	23	13	33	34	GND	--		
		19	MOSI	PWM3	24	19	35	36	16	27	Encoder 1 (in)	
		20	GND	PWM4	25	26	37	38	20	28	Encoder 2 (in)	
		21	MISO		--	GND	39	40	21	29	HORN (out)	12 Header 3
		22	25									
		23	Sck									
		24	CE0									
		25	GND									
		26	CE1									
		27	EED									
		28	EEC									
		29	5									
		30	NC									
		31	6									
		32	12									
		33	13									
		34	GND									
		35	19									



A note on the use of the breadboard: The breadboard included in the system on a bench allows the sharing of all ground and 3V3 or 5V connections across all pins of the RPI. All of the 3V3 connections are tied together and put in to the breadboard, and all 3V3 on the harness side are also breadboarded. The same is true of the 5V and GND connections. The breadbaord also allows the connection of the ADC, which has its own pin out in the appendix.

		36	16	
		37	26	
		38	20	
		39	GND	
C2		40	21	
STEERING H BRIDGE				
	H bridge 12V IN	1	12VDC	B+ on device
H1.1		2	GND	B- on device
	H Bridge Signal	1	PWM1	Labeled on device
		2	PWM2	
		3	3V3	R_EN
		4	3V3	L_EN
		5	3V3	
H1.2		6	GND	
	H Bridge 12V OUT	1	M+	Labeled on device
H1.3		2	M-	
BRAKING H BRIDGE				
	H bridge 12V IN	1	12VDC	B+ on device
H2.1		2	GND	B- on device
	H Bridge Signal	1	PWM3	Labeled on device
		2	PWM4	
		3	3V3	R_EN
		4	3V3	L_EN
		5	3V3	
H2.2		6	GND	
	H Bridge 12V OUT	1	M+	Labeled on device
H2.3		2	M-	
MODE SELECT PANEL				
	Display signal		SDI	blue
			SCL	brown
			5V	red
M.A			GND	black
	Buttons and Drivers		3V3	Labeled on harness, see RPI Pin table
			GND	
			Mode Select 1	
			Mode Select 2	
			Mode Select 3	
			LED 1	
			LED 2	
			LED 3	
M.B			Alarm	
	5V Buck converter input	1	12VDC_IN	
P1		2	GND_IN	
		1	3.3V	Top
		2	GND	Middle
S1	Steering Angle Sensor	3	Signal	Bottom

S2	Brake Pedal Angle Sensor	1	3V3	Top
		2	Signal	Middle
		3	GND	Bottom
S3	Brake Actuator Angle Sensor	1	3V3	Top
		2	Signal	Middle
		3	GND	Bottom
R1	Relay Board Control	1	GND	
		2	Right Turn Signal	
		3	Left Turn Signal	
		4	Horn	
		5	Headlight	
		6	Unused	
		7	Unused	
		8	Unused	
		9	Unused	
		10	5V	
R2	Relay Board Output	1	NC	
	Note: There are more pins, but they are not used.	2	Right Turn Signal In	
		3	GND	
		4	NC	
		5	Left Turn Signal In	
		6	GND	
		7	NC	
		8	Horn In	
		9	GND	
		10	NC	
		11	Headlight In	
		12	GND	



H. List of Error Code Meanings

I. Online Documentation

J. Gitlab Repo

This project's codebase is located in Gitlab. This is a group that contains several projects, spanning both the embedded side and the higher control side. The *CAD* project contains STL files for the various components that were developed. *Onboarding for SPC5 Studio* contains sample SPC5 projects that configure and implement various lower level drivers. *Higher Controller Interface* is where we intended to place sample scripts for the HSL to autonomously control the DBW system, however we failed to address this due to time constraints. *System on a Bench v 1-3* contains the code run on the SPC58. As a result of our SPC58 braking, and having to switch to the Raspberry Pi, the most current codebase is actually found in the *System on a Bench vπ* project. As mentioned in our Letter to Future Teams, teams should begin by converting the *System on a Bench vπ* codebase back to the SPC58.

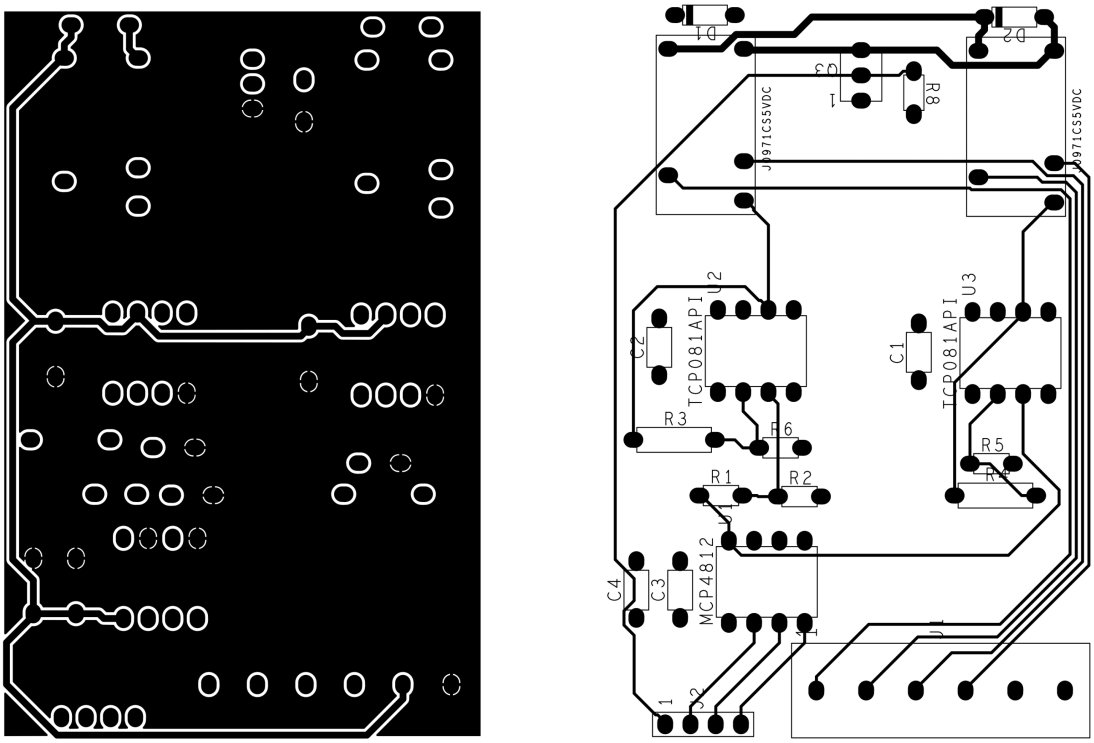
System on a Bench vπ consists of a number of header files, as we believed abstraction was the key to maintaining a clean codebase. Drive_by_Wire.c contains the state machine implementation, and every other file exists to support this. Main.h contains definitions for enums used for the state machine and current input mode (gamepad, manual control, autonomous). CarValues.c/h contain definitions for the struct we use to keep track of the current car data, and certain functions to interface with these structs. Setup.c/h contains the functions which run the system setup tests. Operation.c/h contains the function which updates the subsystems during DBW operation. CircularBuffer.c/h, packet_builder.c/h, and Serial.c/h are all used in the I/O between central controller, joystick controller, and autonomous controller. Leds.h contains files to control the mode select panel leds.

K. SPC5 Studio Guide

L. SPC58 Driver Configuration Guide

M. PCB Specifications

The Accelerator Contention Circuit (ACC) is the only custom PCB implemented on the system on a bench. This section includes each layer of the board for visualization purposes only. The actual files necessary for production of the board are included in the shared Google Drive folder under [Electrical Systems→Accelerator Contention→Version 3 → GERBERv3.1.zip].



It should be noted that these drawings are not to scale, and are included for reference only. In the top layer image (white, right side) C4 is an unused additional filtering capacitor for the power line to the MCP4812 DAC. If noise issues are encountered on the power supply line to the ACC, this capacitor should be added based on the needs of the specific application.

Necessary Components for the ACC			
Part Name	Reference Designator	Quantity	DigiKey Part Number
5V Relay 450mW	Q1, Q2	2	2449-J0971CS5VDC.45
10k 1% Resistor	R1, 2, 5, 6, 8	5	10.0KXBK
1k 1% Resistor	R3, 4	2	BC3916CT
10pF Capacitor	C1, 2, 3	3	445-173471-1
Unimplemented filter	C4	1	N/A

DAC	U1	1	MCP4812 10 bit DAC
Single Supply Op Amp	U2,3	2	TCP081API
6 Terminal Screw Connector	J1	1	WM13968
4 Pin Vertical Header	J2	1	CONN HEADER VERT 4POS 2.54MM
Diode	D1,2	2	1N4148FSCT
N-channel FET	Q3	1	TN2106N3-G