IGVC Drive by Wire Project
Critical Design Report

Team SAS (Semi-Autonomous Sedan)

Spring 2022

Table of Contents

Problem Description

Drive by Wire

Computerize system regulating vehicle speed and direction through actuating the throttle, steering angle & velocity, braking force, and other functions of the car.

Competition

Develop DBW kit in preparation for the Intelligent Ground Vehicle Competition (IGVC)

Basic Requirements:

The requirements for this project include rules from a variety of different sources including government regulations, IGVC rules, and compliance with Autoware.AI. The biggest government regulation that we have had to keep at the forefront of this project is keeping the car a legal Low Speed Vehicle or LSV. The Polaris Gem e2 comes as an LSV so as long as no major changes are made that may take the vehicle out of compliance, this requirement is easy to manage. By focusing our designs on minimal changes to the Polaris and adding in a separate system for the drive by wire, the vehicle will stay in compliance.

Not only does this project need to remain street legal, it more importantly needs to follow the rules and guidelines set out in the Intelligent Ground Vehicle Competition (IGVC) rules handbook. The main concerns we have when it comes to the IGVC regulations are their safety tests and safety measures that are required in order to compete. The safety tests that IGVC provides are also great test cases for the vehicle in general.

Finally, there is the matter of supporting Autoware.AI version 1.4 to make the Polaris CARMA3 compatible. This is more of a luxury than a necessity but being able to use the Polaris for research related to self-driving cars.

EHS

Road Safety and Permits

The vehicle must be able to function safely on the roads so all signal lights, headlights and seatbelts must be kept operational in our design.

Electrical Safety

The vehicle is powered by a 48v battery bank with enough current to be a serious safety hazard, any wiring done needs to be done with the batteries disconnected. All wiring should be inspected for exposed and loose hanging wires and correctly covered and secured.

Mechanical Safety

Having mechanical moving parts can create a possible crush or pinch point, any such areas should be identified and properly protected or covered.

Fabrication Safety

During Fabrication and installation safe working practices should be followed by every team member. Only individuals who have had the training for that task should be the ones performing it.

Vehicle operation

The vehicle should be designed to be safe to operate, for the team testing it and for the general public who may interact with the vehicle. We must also ensure that there is a viable method of stopping the vehicle at all times.

Engineering Standards

Engineers shall hold paramount the safety, health, and welfare of the public. Engineers shall give credit for engineering work to those to whom credit is due and will recognize the proprietary interests of others. Not only does this project intend to act as professional engineers we also intend to build a product like one

NEC 70

- Article 712 DC Microgrids – Using DC circuits that may or may not be connected to the grid.
- Article 725 Class 1, Class 2, and Class 3 Remote-Control, Signaling, and Power-Limited Circuits – Wiring and proper use of wires for different loads.
- Article 727 ITC (Instrumentation Cable Tray) guidelines – Running wires around the cabin of the car and how to do it properly and safely

Knowledge Acquisition

We have gathered knowledge from a variety of sources, including the published manuals for our vehicle, other relevant literature, and by examining autonomous vehicle designs such as those that are available commercially and those produced by other teams. We worked with Allied motion to get all the relevant information we need to work with the EPS.

Detailed Design

PCB Part Selection

Test Board

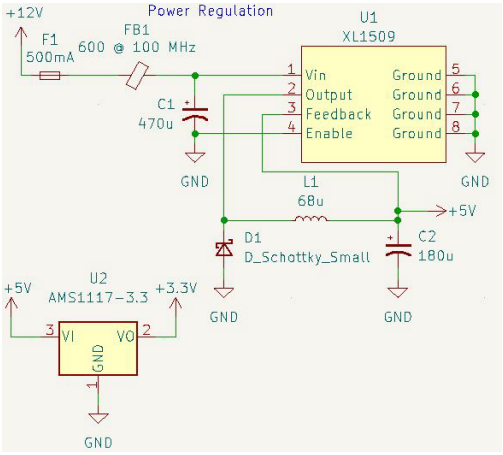
The test board was designed and ordered first in order to speed up our programming on this project. The simple test boards allowed us to ensure that code would compile and test some basic functionality like CAN and DAC code. Going through part selection the one common theme is that all the parts are available on Digikey or Mouser. At first we looked at parts available at JLCPCB because they take care of the manufacturing and soldering of almost all of the parts on the PCB. When ordering parts during the second phase of the project we had to switch from JLC as our parts provider as they had some shortages with key components.

Processor – STM32F405RTG6

When considering what processor we would use in order to communicate and do computations for the various subsystems on the vehicle we first looked at what kind of support there was for CAN communications. Since these processors would have to communicate with both our network and the vehicles CAN network, two separate CAN interfaces was a must. Not only was CAN necessary but we also needed some computing power in order to calculate different things like the acceleration curve for the throttle or reading the quadrature encoder of the linear actuator. We went an STM32 M4 processor that would be able to do all we needed and have plenty of room for future expansion. When looking at what processors were available, we decided since we were upgrading the auxiliary battery on the Polaris power would not be an issue so we went with the high performance STM32F405. This processor had a 64-pin package that gave us plenty of flexibility when designing the PCB. The one downfall to using this processor, or any STM32 processor, is the lack of availability to source this part. When we did the second round of PCBs JLC did not have the F405 in stock and we had to move our microprocessors from the test board to the new ones. In the future we would recommend finding a supplier or processor with enough stock that the team can have processors on each board made without having to move them around. The time we put into transferring the processors to a new board could have been spent on other more important tasks, the risk of damaging the processors during transfer is also extremely high and can set the project back.

Power System – AP1509 + NCP1117

There are several unique parts about designing the power supply for this board. The first part was using the auxiliary battery from the vehicle meant that we had to step 12V to 5V and 3.3V. We first considered using a LM317 linear regulator but quickly decided against it due to the amount of heat that could possibly dissipated during operation and inefficient power conversion. Instead, we went with a switching buck converter to get 5V then a linear regulator for the 3.3V needed for the microprocessor. The AP1509 and NCP1117 were chosen as regulators due to its availability and simplicity. The AP1509 has a maximum output current of 2 amps which is much more than necessary for any of the board we made. We chose to use a 3.3V linear regulator to give a stable, ripple-free supply for the microprocessor and other 3.3V devices. We also added a fuse and ferrite bead in order to protect the microprocessor and other



devices against over current and high frequency noise.

CAN – SN65HVD230

The CAN transceiver for this board was chosen mainly because it was a JLC Basic part and could be easily procured elsewhere. This transceiver was easy to set up and allowed us to test the CAN functionality of the STM32 processor. The one thing that we had to be careful about when testing with more than one board is making sure there are only two termination resistors in the CAN chain. CAN communication is very specific in how the network must be setup, there can be many devices or transceivers on a single line but there can only be one 120Ω resistor at each end of the line to terminate.

IMU - MPU 6050

When initially designing the boards for testing we decided that it would be easy to add an inertial measurement unit or IMU allowing us to gather more data that could be used for autonomous driving later. With the MPU 6050 we chose being relatively inexpensive there could be an IMU per board on the vehicle giving lots of data that we can use to filter for more precise measurements. This IMU is no longer manufactured and therefore hard to find, in the continuation of this project we would recommend finding another, more modern IMU that has more stock available.

GPIO

For this board we wanted to use it to test a variety of things, but we were not sure what that may be so in order to overcome this unknown there are 8 general purpose I/O pins on the board. These pins can be used for anything, they just need to be initialized in CubeMX. One important thing to note is that both DAC outputs are available to use on the GPIO port for our board, this was very useful when prototyping and testing the throttle system as we were able to ensure that the DACs worked properly.

USB

All the boards we created have a micro-USB connector mainly for future use. They are wired on the processor so that they can be used in future years. The idea behind adding a USB port on the boards is to allow data to be directly streamed to a computer in the vehicle for testing and data gathering purposes. These USB connections can run at speeds up to 12Mb/s which could allow for some high-speed sensor interface.

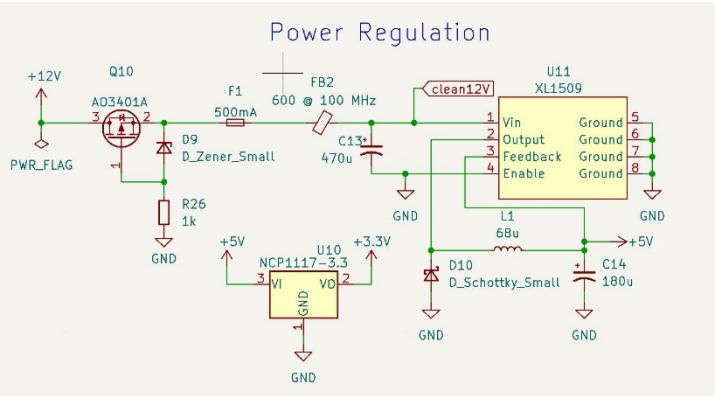
Status LEDs

On every board that we made there are 4 status LEDs, 3 of which are controlled by the microprocessor. The fourth LED is connected directly to the 3.3V rail and is used to determine whether the power supply is working. The other 3 LEDs can be utilized however the user wants, for example we had one blink when the processor was functional and another flash when it received a CAN message. This made debugging code much simpler as we could quickly determine if the hardware or software was to blame for issues.

Connectors

While designing the testing the board we had the goal of making the smallest footprint possible and one part that takes a lot of space is connectors. For the Test Boards we used connec

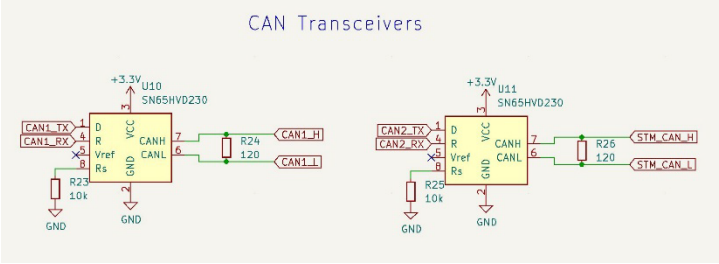
Final Boards



For the final round of design on the boards we learned from the mistakes in the test boards and made improvements in our final revisions. One small mistake that was found on the test board was that the buck converter needs to have its enable pin connected to ground in order to be enabled, this not a big deal as a floating pin also enabled the converter. We connected the enable to ground on the other boards and it worked fine. The biggest change that was on all the boards was the reverse polarity protection added to the power regulation of each board. This included a PMOS transistor, an 11V Zener diode, and a resistor. With this setup the transistor would protect against reverse voltage while the diode and resistor would protect the gate of the transistor. Another change that was made was purchasing a different fuse for the boards. When testing the initial batch of PCBs, we noticed that the fuses were failing at a very high rate but it took some time to determine why. After some testing we found that when

soldering the fuses by hand would irreversibly damage the fuses due to the fact they were made from glass. After ordering non-glass fuses, we had much less problems from soldering. This power design we consistent through all the second round PCBs we ordered.

Another addition to all the boards was a secondary CAN transceiver. This decision was made in order to add versatility to all the boards. The Dash Controller would need to



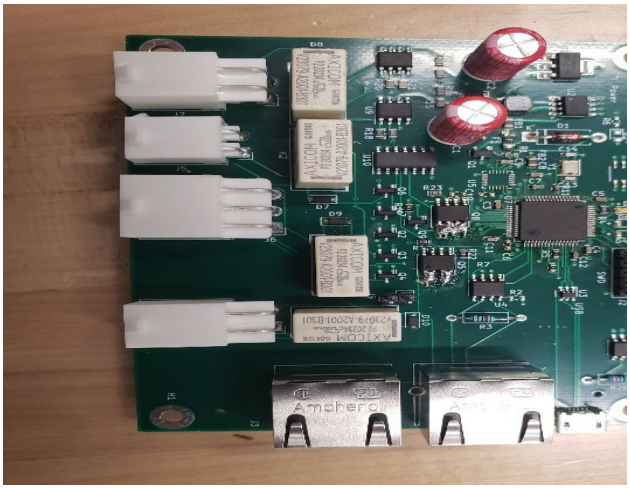
communicate with the EPS through CAN, the Dual CAN to the vehicle, and the Brake Controller would have an extra CAN connection. An important note about all of the CAN hardware on the boards is all of the termination resistors are through hole and not surface mount. By using through hole components for the termination resistor we could easily add or remove it as needed based on where the board is in the bus line.

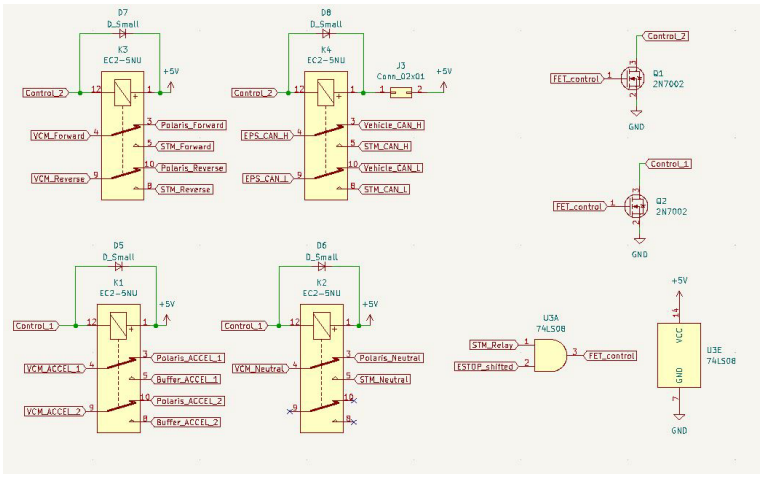
Finally, we fixed all of the connectors on the board to something easier to work with and more durable than the 1.25mm pitch connectors we were using on the test board. For the CAN and power daisy chain we decided to go with RJ45 connectors and CAT5 cable. The CAT5 cable would give us a twisted pair for CAN and the RJ45 connector is easy to plug and unplug but will not come out due to the jostling of the vehicle. For the connectors that would carry the signal wires to various spots in the car we decided to go with TE Connectivity Mate'n'lock connectors for their durability and ease of use. This line of connectors had a 4.14mm pitch between pins which made it very easy to solder to the boards. The connectors themselves were also meant for 18awg wire which is much easier to crimp and make the actual bundles of wire that would run around the vehicle.

When redesigning these boards we found that adding more labels on the silk screen for the debugging connection made that connector much easier to use. When these boards are updated we highly suggest adding more label to any other connectors on the board to prevent the need to go back to the schematic in order to find the pinout for a given connector.

Dash Controller

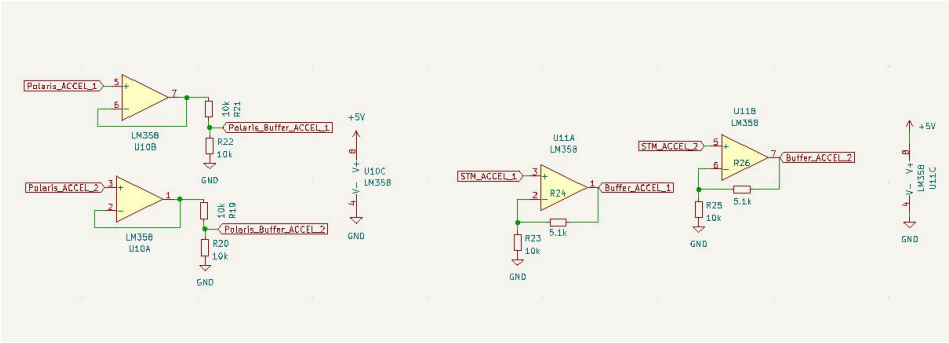
The Dash Controller was the only board to go through two iterations of design before we had a working product. After getting it in the first time and beginning hardware testing we realized their were several unworkable issues that forced us to order another round of this version of the board. The unique part of this board is the relay system and throttle control that was implemented. The relays were used to interrupt the vehicles signals and insert our own





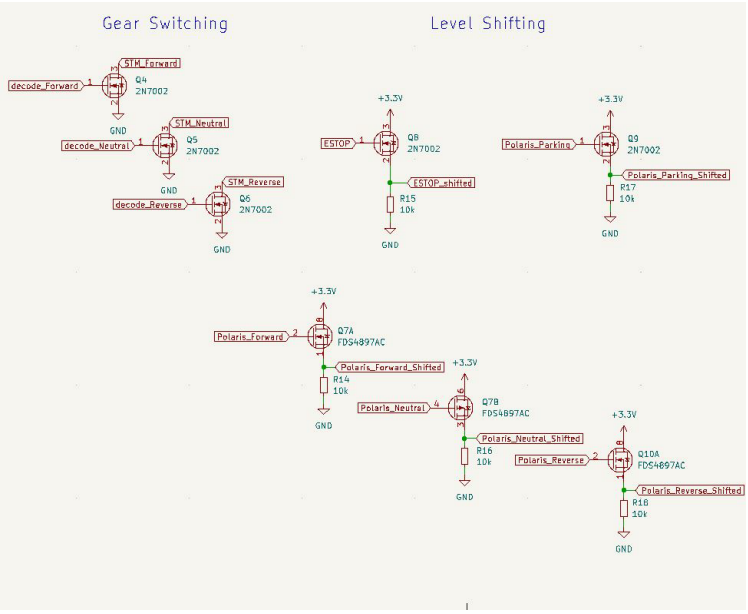
during autonomous operation. All of the relays have the vehicles signals connected to the normally closed side so that in the case of catastrophic power failure they would default to manual operation of the vehicle. We briefly considered using analog switches for this purpose as they are much smaller and would allow us to further shrink the size of the board but a key drawback would be lack of signal transmission during power failure which would not pass safety inspection.

The Dash Controller was also in charge of spoofing the throttle control on the vehicle and reading the current throttle level. The Polaris read a potentiometer that had a range from around 1V to 4V but this is outside the range that the microprocessors onboard DAC can provide. In order to do this we used several non-inverting op amps to boost the DAC output to the appropriate level. We also used a unity gain amplifier with a voltage divider to read the manual throttle of the at any given time. One mistake that was found during testing was that the op amps needed to be connected to the 12V rail on the board and not the regulated 5V rail. We forgot to account for the drop out voltage on the op amps and this meant that with a 5V rail the highest output we could achieve was about 3.6V which translated to a max speed of about 18mph. This is fine for IGVC



as they limit speed to 5mph but for other purposes it would be good to fix this problem and not have hardware limits on the system.

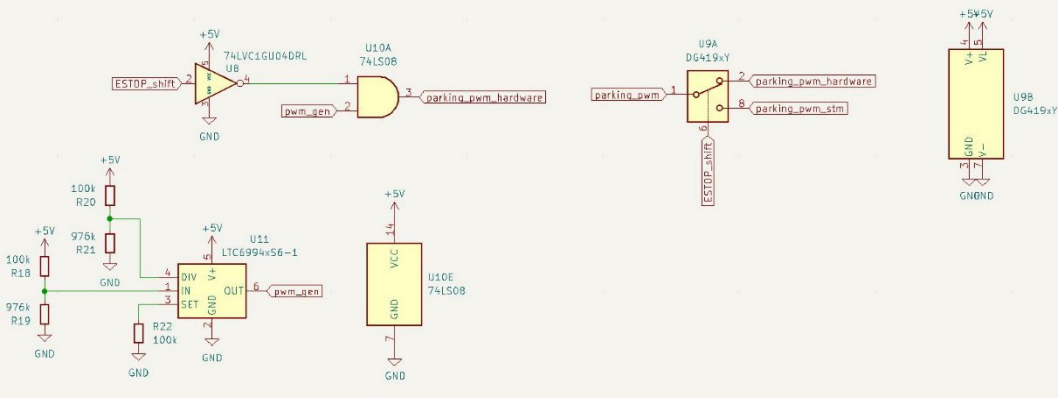
The final notable circuit on the Dash Controller is the level shifting on all of the cars inputs and gear shifting transistors. Most of the switches on the Polaris run at about 13V which would surely destroy the microprocessor if we tried to read this signal directly. Instead we used



NMOS transistors in a source follower setup to shift the signals from 13V to 3.3V that is safe to be fed to the microprocessor. This system worked good enough that we also used it to pull down the Estop line and read it on this board instead of a voltage divider. The gear shifting work in much the same manner, in order to shift into a gear we had to pull that line low. For the gears instead of feeding the input to the gate of the transistor it was now at the drain and the microprocessor controller the gate, this allowed for us to pull any of the gears low and put the Polaris in that gear. On the surface this system seemed to work great but during testing we found that when our board was left unpowered but connected to the vehicles signal line it would cause all of the inputs and outputs on the board to read 13V, this was very concerning at first, but we quickly realized as long as the board was power, we did not have this problem. We think it may have been the transistors causing this problem but without further testing we could not be sure, one solution would be to use optocouplers instead of transistors and isolate the inputs from the rest of the board.

Brake Controller

For the Brake Controller board, the biggest addition is the hardware PWM that is required by IGVC for safety. The entirety of the Estop system must be solely hardware, which



means that we cannot use the microprocessor to actuate any of the brakes to stop the vehicle and had to come up with another solution. We found a hardware PWM generator that we could tune the period and duty cycle externally, this was then piped through a logic gate and a switch before reaching the brake controller. WE had trouble getting the PWM generator to work properly and could only get a 2.4MHz PWM signal which was much too fast to be useful. The motor controller for the brakes also had issues and we had a simple NMOS pair that would drive the linear actuator in one direction. For the future of this project we would recommend using a 555 timer to generate a PWM signal at whatever period and duty cycle is needed. The control logic should work fine but it was not extensively tested, all it does is invert the Estop signal and then ANDs it with the PWM signal. The analog switch works the same way as the relays on the Dash Controller and when the Estop line goes to 0V it switches to engage the hardware brake. One nice feature of the linear actuators is the internal limit switches that prevent them from being over extended and allows for a simple design for the Estop.

This board also had a lot of external connections to run the various hardware on the vehicle. All the control signals for the linear actuators and valves were run to this board. For the main braking linear actuator, we also had a quadrature encoder that had to be powered and read. All the limit switches also had to be run back to the board so we could know if the linear actuators were fully extended or retracted, this part we necessary for the main brakes since the quadrature encoder is a relative encoder. One thing we would change with the connectors that run to other parts of the vehicle is making sure each set of limit switches had their own connector instead of making a single 6 pin connector for all four limit switches. This would make running wire much easier and debugging connection much more straight forward.

Dual CAN Controller

The Dual CAN Controller was created with the idea that we would need to read messages off the car's CAN bus and transcribe some of that information to our CAN bus. We tested the idea of extending the car's CAN bus and adding our own hardware on to it but after some testing we realized that we could not write our own messages to the car as it would recognize that the message was not from its own system and shut down the car. When fixing the problems with the

Dash Controller we decided to add a second CAN transceiver in order to read the car's messages and making the Dual CAN board obsolete. With all that the Dash Controller does it would be wise to keep this second board in order to spread the computational load around and maintain the flexibility of our leaflet design.

Vehicle Control

The control system for this vehicle will consist of three microcontrollers at the lowest level to run actuators, control algorithms for the actuators, and to flip relays between vehicle control and computer controller. The relays shown in Figure 1 below will be set up so that to they will always default to the vehicles original systems. This is for safety reasons; in the event of an

Figure 2:

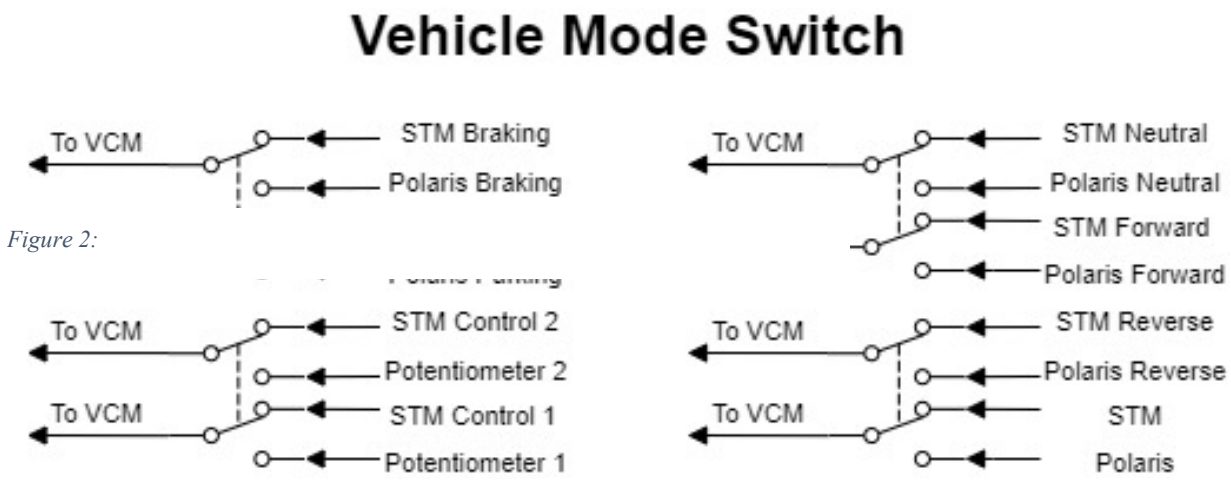
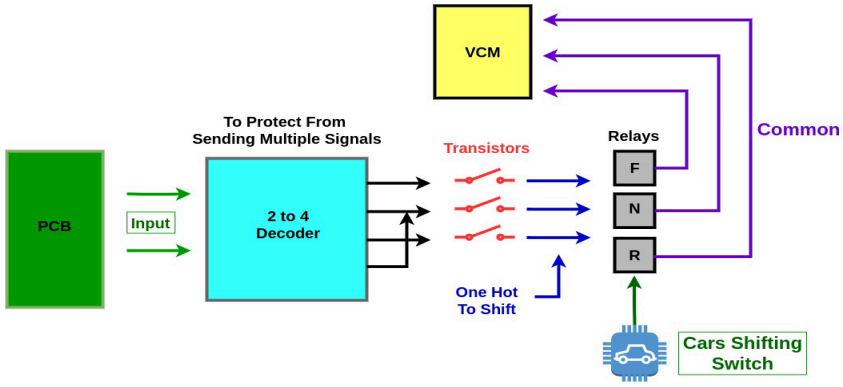


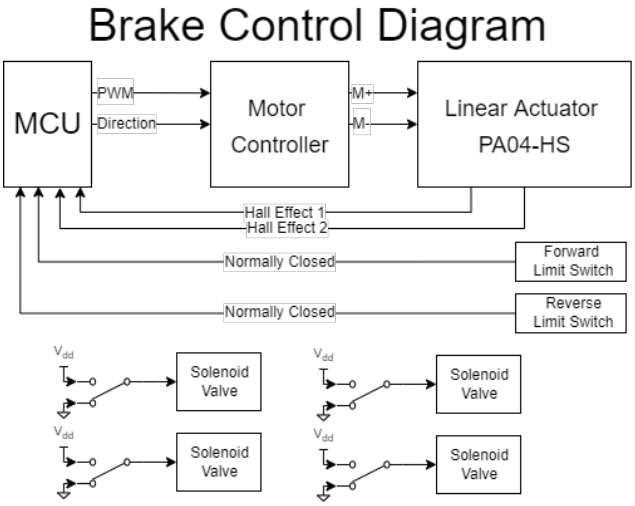
Figure 1

electrical failure the vehicle can still be driven by its operator.



The design for the gear switching is show in figure2 above. The diagrams shows a PCB with a decoder on it to control the shifting. This decoder then leads to a set of three transistors, one for each gear. The need for these transistors is that whatever gear gets the 15v's from the car is put into position. The decoder sends its single to the transistor to allow the 15v's to go to the next part of the system. After this is the relays, this allows the throttle to still be useable. The relays are set to normally closed, in case our

system fails it will revert back to the driver. When the relays are open our stm32 micro-controller are in charge of throttle.



Emergency Brake

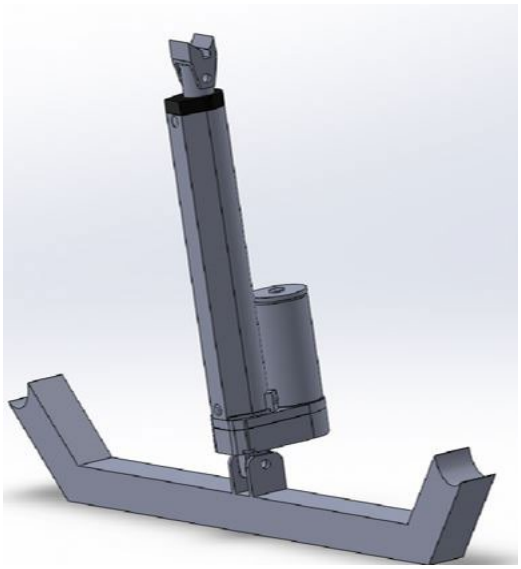
Design Considerations

Before we settled on our linear actuator design, we considered two cam-based designs, one with a motor driven cam and another with a linear actuator driven cam. For the motor driven cam, we would run the brake cable through a cam which would be turned via an electric motor in order to engage the breaks. We decided against this design as finding a small enough motor that would have enough torque would be an issue, alongside the fact that the direct contact from the cam could result in wear in the cable over time. Additionally, for our linear actuator driven cam, we found similar issues where we could not find enough space for the linear actuator alongside having potential wear issues in the cable, making it not a desirable design. This led us to manually engaging the emergency break with a linear actuator due to an ease in mounting alongside not causing any wear in the break wire.

Direct Contact Linear Actuator

We went with this design because it takes up the least amount of space, has the lowest potential for failure, requires the least complex fabrication, and is the most cost effective. With a required thrusting force of 43.5 lbf measured using a force gauge, a full extension time of two seconds, and a required stroke of four inches, we have selected the Progressive Automations PA-14 linear actuator with a max force of 50 lbf and a stroke of 4 inches. The design calls for the actuator to utilize a steel tab located on the existing parking brake mechanism to push and activate the

breaks. We ran a force analysis on the tab to ensure the tab would not deform under normal operation and we found that there will be no deformation of the tab or the attached mounting structure. As for our mounting structure, we have decided to use 1”x1” 16-gauge square tubing. This was chosen because it is inexpensive and easy to use while fabricating. Utilizing the deflection calculation from loading layout number 14 of Table A-9 in Shigley’s Mechanical Engineering Design (Eleventh Edition), $y_{max} = -\frac{Fl^3}{192EI}$, we found the deflection to be roughly 0.007”.



SolidWorks Model of PA-14 actuator mounted on brackets

Figure 3: SolidWorks model of our mounting bracket connected to the PA-14 linear actuator

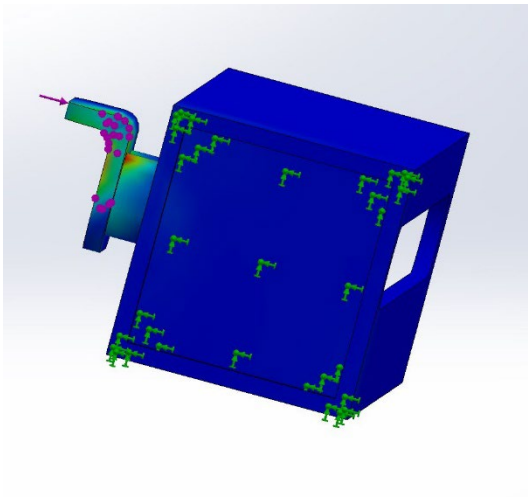


Figure 4: FE Stress analysis conducted in SolidWorks applying 50lbf

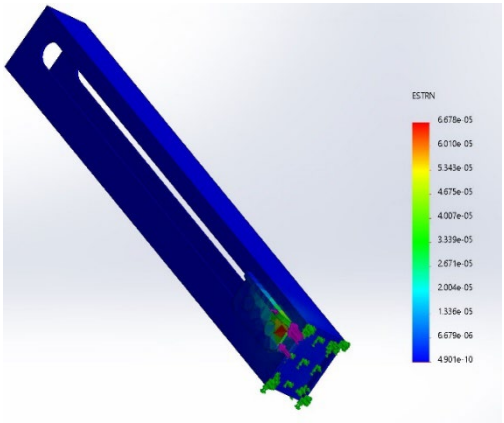


Figure 5: FE strain analysis conducted in SolidWorks applying 50lbf

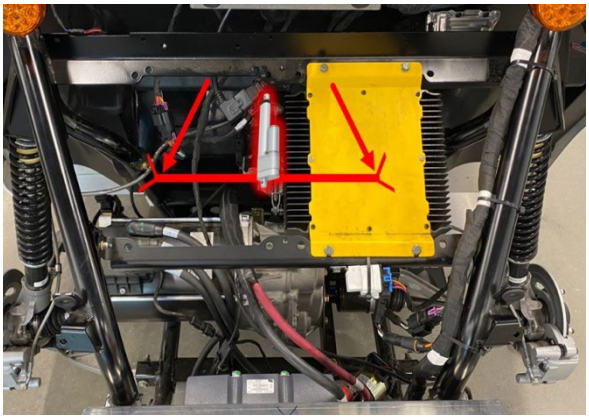


Figure 6: Layout and positioning of actuator mounting location relative to engine bay:

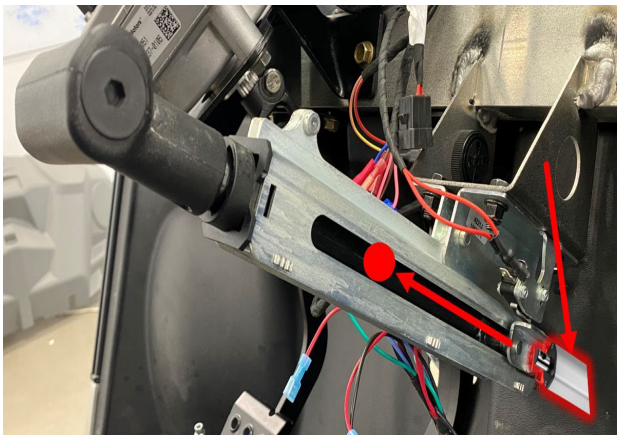


Figure 7: Visualization of actuators dynamic function from inside the vehicle

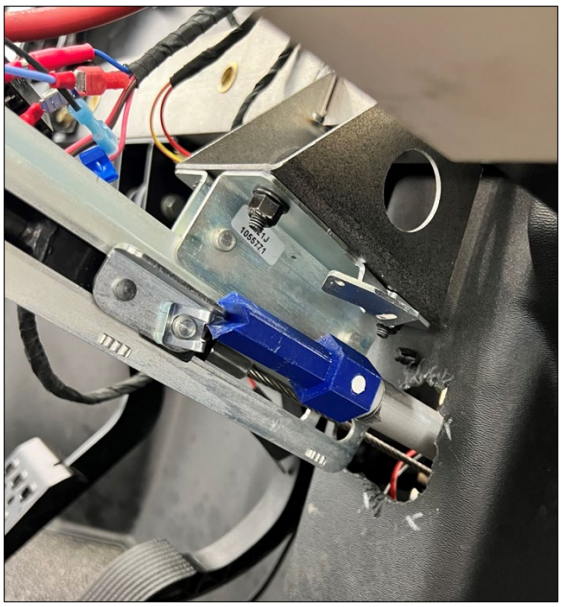


Figure 8: View of actuator attachment making contact with parking break mechanism

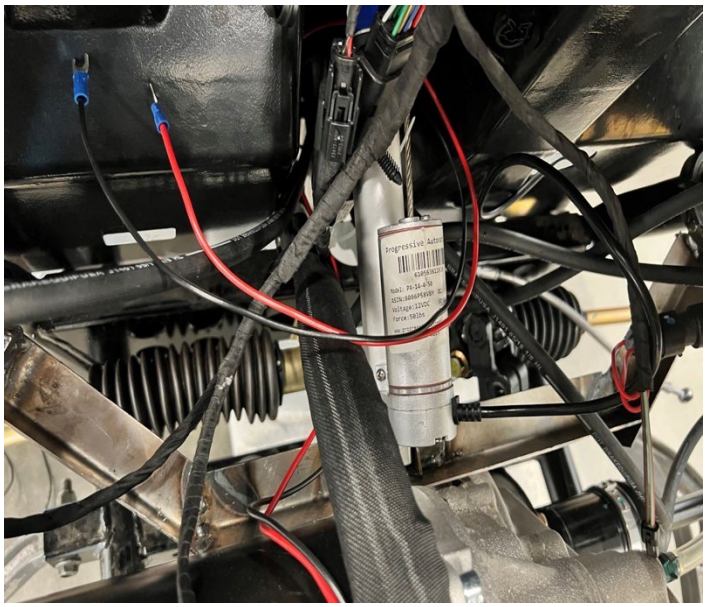


Figure 9: View of mount and actuator from inside the engine bay

The mounting bracket used to attach the actuator to the vehicles frame was fabricated at the DML. Team members first took detailed measurement of the inside of the engine bay and made a cardboard prototype to ensure fit. Angle iron of appropriate gauge and size was selected and then welded together to mimic the cardboard model. Additionally there were two steel shaft collars welded to each end of the mount to aid in attachment to the vehicles frame. A specially designed actuator attachment was designed to fit over the end of the actuator arm and grab the parking brake slide with a notched tip. This attachment was designed using SolidWorks and 3D printed.

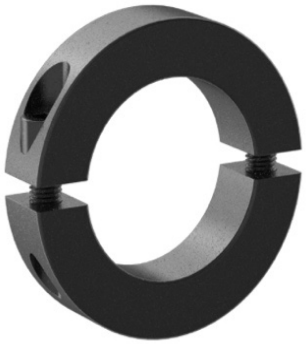


Figure 10: 2-piece shaft collar used to attach mount to vehicles frame

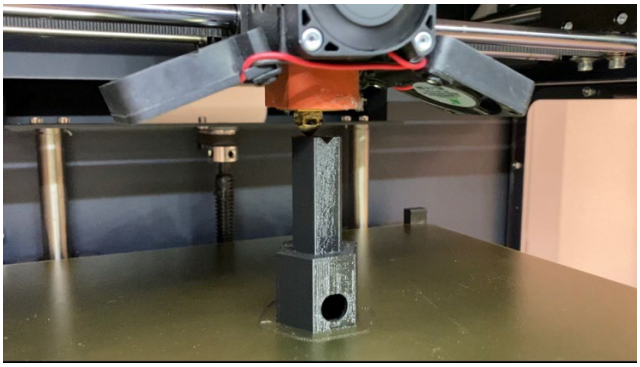


Figure 11: Actuator attachment being printed

Critiques and Recommendations

We are incredibly satisfied with the success of our emergency brake design. It works every time, required minimal permanent modification to the vehicle, and will be easy to modify for future design teams. We have no complaints about its performance and no recommended changes.

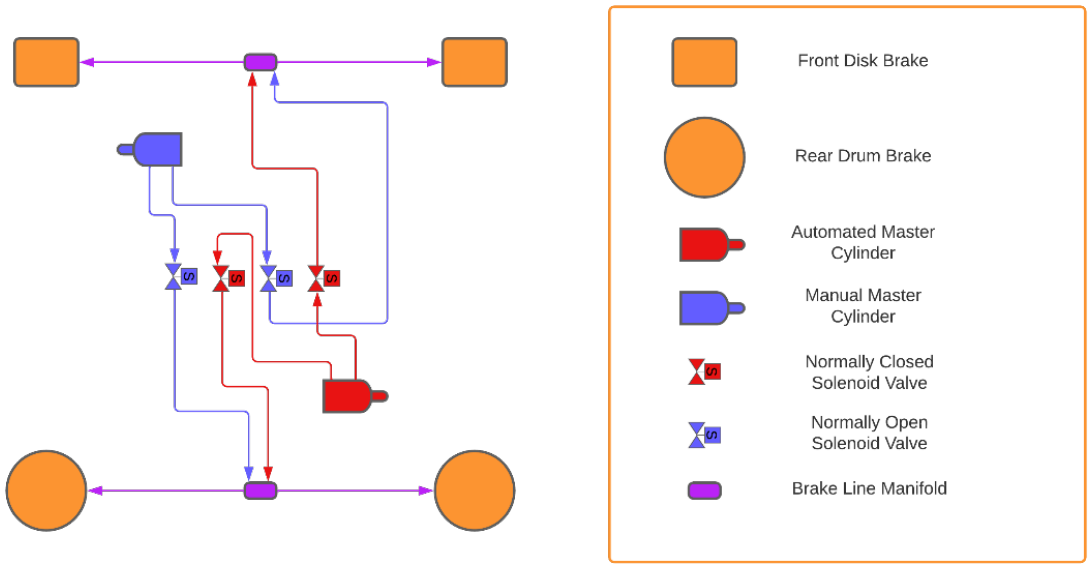
Automated Brakes

Design Considerations

During the preliminary design of the breaks, we considered three main options of running the automated braking system, which are the following: using a hydraulic pump, using a motor driven master cylinder, and using a linear actuator driven master cylinder. For our hydraulic pump design, we would mount a hydraulic pump in the back of the vehicle, which would run in order to engage the breaks. We were initially confident in this design because it is a design commonly used in trailer breaks, however, we decided against it due to the pump requiring more power than desired. Next, we considered mounting a second master cylinder that would be pressurized by an electric motor actuating a rack and pinion system. While this design would require less power than the hydraulic pump, this design would require a significant amount of

fabrication alongside being expensive due to needing a high torque, high precision motor to run the system.

Linear Actuator Driven Master Cylinder



The automated braking system consists of a Hayes Sidewinder II master cylinder with a 5/8” bore diameter actuated by a Progressive Automations PA04-HS linear actuator, all of which are isolated when manually operated by a pair of normally closed Omega SV121 solenoid valves with PCTFE seals. We began component selection by determining the brake line pressure required to bring the vehicle to a stop without skidding. By calculating the energy dissipation required to bring the vehicle to a stop from a speed of 25 miles per hour, we were able to calculate the required torque between all the braking points. This was found to be 8564 lbf-in. We then utilized equations 16-1 through 16-10, 16-29 through 16-36, and 16-50 through16-52

Figure 8: Flow diagram for both the automated and manual braking systems

from Shigley’s Mechanical Engineering Design (Eleventh Edition) to develop the following master formula for the total torque output in relation to the brake line pressure:

$$T_{total} = \frac{P_{line} * A_{DiskPiston} * f * (r_o^2 - r_i^2)}{(r_o - r_i)} + 2 * \left(\frac{A_{DrumPiston} * f * c * r * (\cos\theta_1 - \cos\theta_2) * P_{line}}{\sin\theta_a} \right) * \left(\frac{1}{a * \int_{\theta_1}^{\theta_2} (\sin\theta)^2 \, d\theta + f * \int_{\theta_1}^{\theta_2} \sin\theta * (r - a * \cos\theta) \, d\theta} + \frac{1}{a * \int_{\theta_1}^{\theta_2} (\sin\theta)^2 \, d\theta - f * \int_{\theta_1}^{\theta_2} \sin\theta * (r - a * \cos\theta) \, d\theta} \right)$$

When all variables are replaced with their respective values taken from the geometry of the braking components, we were left with $T_{total} = 13.35 * P_{line}$. After substituting our required torque for T_{total} and solving for P_{line}, we found our required brake line pressure to be 641.5 PSI.

Since our team decided we would like to go with the OEM master cylinder for our design, we utilized the surface area of this piston to determine the force required to pressurize the

brake lines. We found this force to be 197 lbf. This was set as our minimum force when searching for our linear actuator. Due to the nature of our design, we also needed a linear actuator with built in position sensors, precision movements, a minimum stroke of ½", a 12V operating voltage, and an IP66 rating. This led us to select a PA-04-6-400 linear actuator with Hall effect sensors.

In order to isolate each of the braking system when not in use, we needed solenoid valves that could handle the high pressure as well as the corrosive nature of braking fluid. We initially wanted to use a 3 way/ 2 position universal solenoid valve that would allow us to select which master cylinder would act as the pressure source for the brakes, but these valves were either out of budget or had prohibitively long lead times. We then decided to use a bank of four valves, two normally open, two normally closed. After multiple changes due to availability, we selected the Omega SV121 for the normally closed valves. These valves have a max operating pressure of 1000 PSI and all wetted materials are chemical resistant to glycol-based compound, the corrosive agents in brake fluid. Due to the nature of normally open solenoid valves, these were more difficult to find. However, we were able to find solenoids commonly used in trailer brake systems in order to disable the brakes during backing maneuvers. These valves are rated to 800 PSI and are also designed to be resistant to braking fluid.

Once we received everything we needed, we began installation of the valves and brake lines manufactured by NAPA. During the installation we couldn't completely remove the original brake line. However, this did not interfere with the installation of the purchased brake line or any other functions on the vehicle. We also came across a connection issue on the brake line. We were able to purchase another connector to replace the damaged part of the brake line. To make sure that the line worked properly, we pumped fluid throughout the system by stepping on and off the brake pedal. This helped us check for air within the system. We did this in all four brakes. We were able to get the lines to flow properly as needed.

In order to mount the master cylinder and the linear actuator we need to design two mounts that would allow the master cylinder and linear actuator to align. Once we received the linear actuator, we realized the actuator was bigger than expected. We were able to design a mount for the master cylinder and linear actuator to mount on the back of the vehicle. We were able to create mounts using 3/16" steel sheet metal. Although we still needed to design a part the shaft of the linear actuator can be secured to for alignment and stability. Due to the size of the linear actuator and assuring we have adequate alignment with the master cylinder we had to set the actuator on a steel beam to get the necessary height for alignment. We secured the beam to the actuator with a U-bracket. We also had to create a part that will connect the master cylinder shaft with the shaft of the linear actuator. We had to make the part small enough to fit into the linear actuator and big enough to press the switch for the limit switches. Which were mounted on the front side of the master cylinder on a metal bracket. We made sure the switches would not interfere with the function of the master cylinder and linear actuator.

As a result of the brake controller hardware failure, to test and demonstrate the automated braking a manual switch was connected to a relay board that would flip the polarity of the motor allowing you do move it in and out, the limit switches are not connected to this systems so it is

possible to overextend the linear actuator. This is the current wiring configuration the brakes are in, with the solenoid valves with connectors that have to be connected and disconnected manually to power.

Critiques and Recommendations

Overall, we are pleased with the performance of our braking system but there were several design changes we would make if we were to do it again. The two primary changes we would make would be to design a one-piece mounting structure for the linear actuator and master cylinder and to have the brake system pressure bled by a professional.

As for the mounting structure, our current design gets the job done sufficiently well but due to some of the looser tolerances we held during fabrication, it is not the most consistent. To address this, we would build a more substantial mounting system that would allow for more precise brake control.

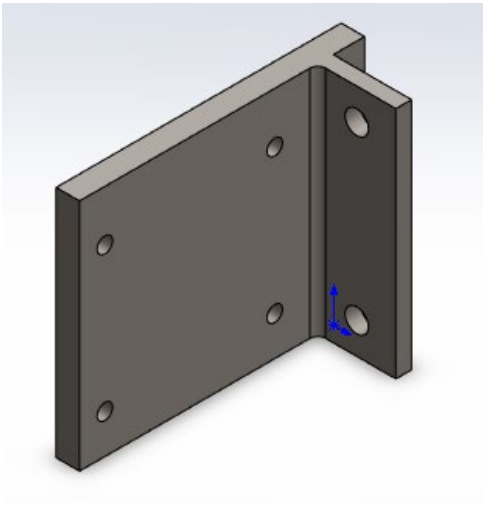
As for bleeding the brake lines, due to the complex nature of our brake plumbing, our lines are difficult to bleed. We found that one particular set of lines was prone to trapping air pockets that, regardless of our efforts to bleed the brakes, would not come out. To address this, we would need to take the vehicle to a professional with the equipment required to pressure bleed the brake system. This would remove any trapped air pockets and substantially stiffen the brakes.

Mounting

E-Stop

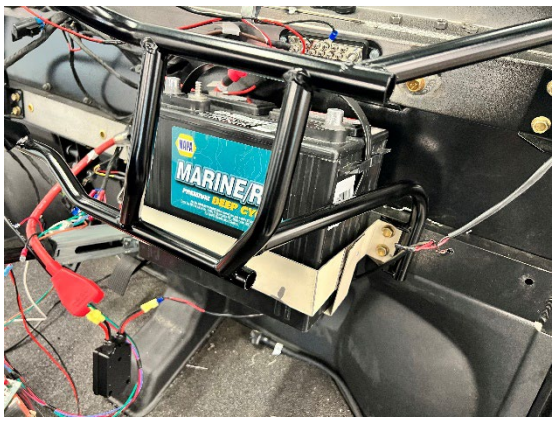
To follow IGVC rules, we were required to install two emergency E-Stop buttons on both sides of the vehicle. Initially, we planned to mount both buttons directly on the side of the vehicle with an industrial adhesive. We found this solution to be desirable as it required no fabrication alongside us not having to worry about the mount having any wear or fatigue form continual use of the E-Stop. While this solution worked initially, we found that in mounting the buttons directly on the side of the vehicle, we had made the vehicle wider than IGVC regulations by an inch.

To remain withing IGVC rules, we then decided to create mounts for the E-Stop Buttons as shown below and attach the buttons to the outer frame of the car. We decided to do this as it required no modification as we could run a bolt and nut through a channel in the frame that we found. Additionally, this would allow us to use this channel to discreetly hide the wires for the button. The mounts were secured to the vehicle by running a nut and bolt through a channel in the car's frame. Overall, we were content with our button mounts as it allowed us to cleanly mount the buttons while also remaining in IGVC rules, however, in the future, we would ideally want to fabricate these mounts out of sheet metal in order address any concerns about wear or fatigue in the mounts



Battery Mount

In order to accommodate for the increase in power consumption, our group upgraded the 12AH battery included in the vehicle to a 72AH battery. Because the 72AH battery was significantly larger than the 12AH battery, we needed to update the mounting bracket to properly contain the larger size. Initially, we borrowed heavily from the design of the original battery bracket, where we primarily upsized the bracket while also adding an additional strap below to account for the increase in weight. We then fabricated the mount using basic sheet metal as shown below and installed the bracket using the holes from the initial battery bracket.



After remounting the battery, we realized that we needed to shift the battery upwards in order to reinstall the dashboard back onto the vehicle. As a solution, we mounted straps on the bar above the battery which were then used to secure it in place. In the future, we would ideally modify our preexisting bracket to where it can be mounted on this mounted bar to give us enough clearance to put the dashboard back on the car.

SOFTWARE

All software on the custom PCBs is housed on the dbw_core GitHub repository. The software team made heavy use of branches; code is organized into the main branch, development branch, and numerous feature branches. The main branch should compile and run at all times. The development branch should compile, but likely has runtime issues. Bleeding edge code is stored in the feature branches; successful compilation is not guaranteed.

Much of the code is modeled after Mbed OS, which provides an Arduino-like ecosystem on many powerful processors. This said, porting Mbed to custom boards is arguably more labor intensive than simply coding a simple replica. Our version supports abstracts multithreaded functionality and GPIO access, including libraries for digital input/output, analog input/output, and PWM. While much of the peripheral initialization is handled in object construction, we still require the use of STM32CubeMX. This is a graphical interface included with STM32CubeIDE which generates a substantial amount of low-level code handling external interrupts and the Hardware Abstraction Layer (HAL). All GPIO classes are thread safe.

We implement a unified code base for all custom boards. This is designed to support EEPROM parameter servers in the future, where the board type is saved as a nonvolatile variable. The following section outlines the functionality included by our API.

Helpful Libraries

AnalogOut.cpp/h

AnalogOut handles all the analog writing, which also includes DAC.

CAN.cpp/h

These files include the CAN library, which provides a high-level abstraction of the CAN bus. Users need only specify which CAN bus to use in the constructor and update the CubeMX UI. The CAN class support a subscription interface similar to ROS, allowing developers to register callbacks to a specific CAN ID. The low level callback functions map CAN IDs to callback handles, efficiently bridging between the HAL API and a succinct C++ interface.

ParamServer.cpp/h

This contains the code for using the eeprom for saving and writing parameters that you might need to save to a board.

eeprom.cpp/h

This is for the eeprom of a given chip. Every chip has its own method for eeprom. Some chips vary little in code, but a couple numbers off and it will not work.

Thread.cpp/h

These files contain an abstract class to simplify thread creation. Simple threads may be passed as a void function of the type osThreadFunc_t. More complex threads can be encapsulated in a class which inherits the Thread class, requiring that the child class implements a void run(void) function. This is similar to the Python threading API.

DigitalIn.cpp/h

The DigitalIn class supports both polling and interrupt based execution. Interrupt callback functions must be of the type digitalIntIrqCb, which is of the form void(uint8_t pinState). Comparison operators are overridden to simplify operations which require the pin value. All external interrupts must be initialized in STM32CubeMx as much of the functionality is baked into the HAL API.

DigitalOut.cpp/h

The DigitalOut class automatically configures GPIO pins to be in output mode. No further action is required in the STM32CubeMX UI. It also overrides multiple setting and comparison operators to simplify interfacing with other classes.

PWM.cpp/h

The PWM class supports PWM output. The pin must be previously initialized in STM32CubeMX. The class constructor requires the HAL Timer instance and Timer channel as input. These values can be found in the STM32CubeMX UI.

InitDevice.c

The STM32CubeMX UI generates code in main.c. We use C++ to simplify code development, which means that the initialization code must be moved to another file. initDevice.c is where all auto-generated code is moved to ensure proper initialization of low-level structure in the HAL API.

Main.cpp

Since the same project is used for all the boards, to be able to upload separate code to each board main.cpp contains the way to do this. When you upload a code to the board you can change the BoardType to whatever board you are uploading code too. This way when you run the code it will check to see which board you are using and the branch off to that part of the code.

Controlling the Car

Brake Control

The brakes are controlled by the Brake Controller board that is to be mounted underneath the Polaris. Both of the linear actuators are run by a motor controller that in turn is controlled by PWM from the Brake Controller. Due to time and product availability we were only able to get a simple NMOS transistor for running the brakes, this worked as a proof of concept but is nowhere near what is going to be needed to safety control the Polaris. For the next team a top priority should be ordering two motor controllers with the proper power rating and safety features, like overcurrent protection, to run the braking linear actuators. For the solenoid control all we had to do was implement relays that could flip on and off to switch the hydraulics between the manual and automated systems. This was implemented on its own daughter board as two of the solenoids ran on 24V and we wanted some isolation between our electronics and the 24V of the valves.

Gear Control

The gears are controlled by grounding the position of the gear pin that you want to go into. The layout of the gear switch has six pins. Three of the pins control which gear you are in. The three pins have wires coming from them to our board. These three wires carry 12 volts. To switch between the gears, you send out DAC command from the GPIO pins from the board to the transistor on the board that is connected to the gear you want to change. The transistor uses pull-down resistors to ground that pin and put the car in to position. There are relays on the board that allow to switch between the car control and the board control.

Throttle Control

The throttle is controlled from the DAC output of the stm32 board from the GPIO pins. Starting from the pedal of the car there are two wires that do the main control of the throttle. Both wires are blue striped coloring. Each of the wires has a voltage from a potometer inside the pedal. The wires carry different values though, one wire is basically double the value of the other wire. The smaller voltage wire carries a voltage ranging from 0.538 to 2/089. The bigger voltage wire carries a voltage ranging from 1.082 to 4.14. These two wires go to the stm32 boards to a 4-pin connector. This allows us to control the throttle of the car or use the relay to allow the pedal to still work. The other two wires from the board carry the signal back to the car.

Steering Control

The steering on the car **now** (at the time of writing this) is controlled using a PWM signal to one of the pins. The steering that will be put in next year uses CAN to control to the steering unit, the GEM e2 EPS. Here is a helpful link to one of the teams that uses the steering http://www.igvc.org/design/2021/10.pdf. On page four of their report, they talk about PID tuning for the steering. On page 11 they talk about how to communicate with the steering unit. Once you get the steering unit installed you should be able to send it commands. Our plan was to use mode 5 like they did. If you want to be able to control this and know what mode 5 is, you will need to refer to the manual. The steering unit is limited to 2.5 revolutions left and right, therefore 5 in total once you put it into autonomous mode.

Steering hardware Details

This design has been proven to work. The EPS unit is built around an Allied Motion POW-R STEER EPAS Actuator, with a torque output of 0.65 ft-lb/A and a max current of 50 amps. Since, in the worst loading condition, steering requires 5ft-lbs of torque, this unit would only require 7.69 A to provide adequate steering.

CAN Bus

To access the CAN bus of the car there is a diagnostic port under the hood to the left of where the charger plugs in if you are looking at the car. The car's CAN system operates at 250 KHz. To read from the car's CAN system you need a processor which supports CAN 2.0B. Additionally, a CAN transceiver is required to implement the physical layer of the CAN stack. We use the Waveshare SN65HVD230. These chips need 3.3 volts, GND, RX and TX. For a quick hookup to the CAN system, you can use MBED to code the F7463G. On connector 9, pin PD_0 and PD_1 are RD and TD which are RX and TX pins of the board. Wire these up to their respective pins on the transceiver. On the diagnostic port of the car there are 4 wires that are CAN. Only two of these wires will work, the two wires that work are across from a brown ground wire. Yellow is CAN high, and green is CAN low. From there good MBED OS 6 CAN bus for the code.

Integrated Development Environments

MBED

MBED is an easy platform for start off with for uploading code to stm32 boards. They provide a lot of libraries to make it easier to program and upload code. Which can be extremely useful for testing. With MBED you can google the board you are using and see all the pinouts.

STM32CubeIDE

This IDE allows you to configure the outputs of custom chips through STM32CubeMX. They do not have libraries for everything else like MBED does though.

STM32CubeIDE DEBUGGING

Stm32's IDE is especially useful for debugging, they have stack trace backs, last called line of code, register's values, and a lot more.

STM32CubeIDE troubleshooting

One of the easiest ways to fix a compiling issue is to, step1 open the .ioc file for that project. Step2 select a random pin that is not assigned to anything. Step3 change it something random, it does not matter what, Step4 change it back and click in the white area of the back and it will ask you if you want to save and compile code. Another way to fix any problem that should pretty much always work is if it's not a coding problem. Step1 backup any code you have that you want to save. Step2 delete all the projects in your stm32 IDE (this is why we backed up). Go to file and click start new project from .ioc. Step3 fine the .ioc you want to use and go through the setup. The only thing I remember having to change in the setup is changing the project from a c project to a c++ project. Step4 if it didn't compile and generate new code already, then build the project. Step5 move all your source code and include files back over and build again. Since this makes a new project from scratch, and you are still having compiling problems then you can rule out the IDE. Although it could still be a wrong setting the .ioc, or your code.

Power Analysis

The existing electrical system is supplied by a 300W 48VDC-12VDC converter supplying ideally 25A at 12V, with an auxiliary 12V battery that can supply additional current. Adding in the EPS and linear actuators and various other devices will increase the current draw on the current system. The sum of the peak currents, according to each device's datasheet, is 45A, with only 25 being supplied from the DC-DC converter the potential for 20A being drawn from the battery is possible. The current battery is 12AH, at max draw the battery could be drained in 36 minutes. The best solution to this problem was to add a new battery, at 109 AH the new battery will ideally be able to output 20A for 5.45 hours. Batteries are not ideal but increasing the battery size will still give us plenty of auxiliary power. It is also recommended in the Allied Motion EPAS manual the power steering unit not rely solely on the DC-DC converter as a main power source due to current sourcing issues.

Current Draw	
Linear Actuator 1	12A
Linear Actuator 2	5A

PCB/Misc Electronics	2.4A
Relays	600mA
EPS(16.25lb-ft)	25A
Total:	45A
Current Supplied	
300W DC/DC converter	25A
Aux Battery Draw	20A

Testing and Quality Plan

Component Testing

For the component testing part of this project, we went through a process of testing each of the components that we receive and ensured they were in working order before placing them on the Polaris. All the mechanical systems and actuators recieved a few tests specific to that part in order to confirm that they were ready to be used and would not fail prematurely. For the electrical side we confirmed that all the PCBs that we had manufactured were in working order and could be used on the Polaris.

System Testing

We performed system testing in our allocated space in Endeavor while the vehicle was on blocks. We evaluated each subsystem individually and confirmed that they work properly on their own.

For the throttle control we have already begun this process, as a proof of concept for the microcontrollers that we developed we were able to successfully use an Arduino along with a few discrete ICs to control the acceleration of the Polaris from a standstill to full speed. With our microcontrollers we will be able to utilize their onboard DAC and an op amp to control the throttle. The rest of the testing for this system will mostly be with controls and tuning our acceleration curve on different surfaces so it will take place in the Vehicle Testing.

For the braking subsystem testing we verified linear actuator that controls the master cylinder was functioning properly and that when actuated, we had fluid moving to each of the braking points. We also verified that the position sensors and limit switches were operating correctly. We also tested the solenoid valves and relays that control them in order to ensure a smooth transition between the automated brakes and human controlled brakes. During these tests we utilized the pressure sensor on the manual master cylinder and a set of rules on the

microcontrollers to make sure the system went through the correct steps in order when switching between the two.

The emergency brake systems were tested in a similar manner to the regular brakes. The linear actuator was powered until it reached the maximum limit switch and then retracted until it reached the minimum limit switch. With the emergency brakes now activated, we ensured that the rear drum brakes were locked up. We then ensured that the brake could be disengaged manually by the driver.

Vehicle Testing

For the Vehicle Testing we plan to take the Polaris out to a parking lot and test all the subsystems together, first following the steps in our SOP to get the permissions we need then taking the Polaris out to our designated testing area. The first tests run will the acceleration and braking in conjunction in order to confirm that we can safely start and stop the Polaris. This test will consist of getting the Polaris up to 5mph then issuing a brake command to bring it to a complete stop. The next tests will be with the steering and checking the EPAS unit functions properly. First it will be tested at a standstill and checked for a complete range of motion, then at various speeds. Once the complete drive-by-wire kit is confirmed to be in working order we can move on to testing the transition between the automated system and human controlled. These tests will be very important to ensure a safe transition without bringing the vehicle to a complete stop. All of the relays to switch the systems are connected together so testing this system will be as simple issuing the command and checking that all systems are read the cars own sensors and not our microcontrollers.

IGVC Unit Tests

The IGVC unit tests are the last tests to be run mainly because not all of them can be run with a strictly drive by wire system. From the 2022 Rule Book the units tests are as follows:

Unit test 1: Emergency stop Provide plot of speed vs time

Unit test 2: Emergency stop remote Provide plot of speed vs time

Unit test 3: Speed limit test Provide plot of speed vs time. Accelerate until speed limit of 5 mph is reached. Plot should continue at least for 20 seconds after the speed limit had been reached. Test your system with the following speeds:

1) Speed is 3 mph (1.34 mps)

2) Speed is 5.2 mph (2.28 mps)

3) Speed is 6.0 mph (2.68 mps)

Unit test 7: Backing up operation. Speed in reverse should never exceed 2 mph (0.89 mps). 26 Please test the system with the following speeds:

1) Speed limit is 1 mph

2) Speed limit is 3 mph

Provide the following plot: 1) Plot speed in reverse vs time

The rest of the unit tests provided by IGVC are related to autonomous control of the Polaris and is outside the scope of this project. There are also several "Qualification Tests" that are put out by IGVC including tests of the Emergency Stop, these tests are inconsistently worded and are not up to our standards, as such we plan on replacing them with tests of our own while keeping the basic requirements of the IGVC tests in mind. We will run the E-Stop system through several tests to show that it will work in all operating conditions of the car and properly bring the Polaris to a complete stop. The tests will include but not be limited to, when the vehicle is at a standstill, during operation of human control, and during computer control operations. These tests will also be run with the wireless emergency shut-off to make sure that if the driver is unable to press any of the shutoffs on the vehicle.

Preliminary Hazard Analysis

Tipping

Given the calculations shown below and the center of mass in the appendix, tipping was found to be a manageable hazard given that the minimum turning radius was found to be 13.67 ft given the vehicle's max speed of 25 mph. This is an acceptable turning radius as the car will be running at a lower maximum speed of 5 mph to be in accordance with the IGVC rules, so the turning radius will decrease drastically to 0.55 ft. Additionally, it is important to note that the center of gravity for the car was estimated by splitting the car into three sections and distributing the weight into each section.

$$\sum M = 0 = M_F - M_W$$

$$M_F = M_W$$

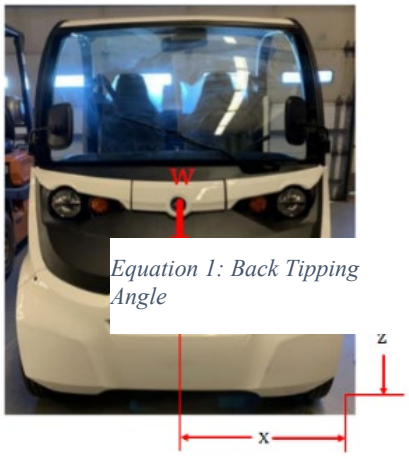
$$Fz = Wx$$

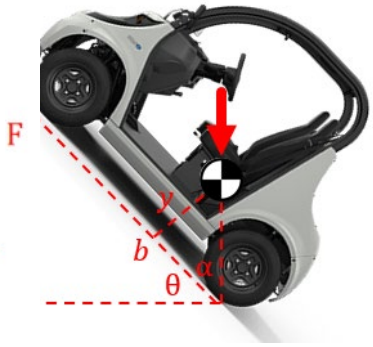
$$Fz = mgx$$

$$m\frac{v^2 z}{r} = mgx$$

$$r = \frac{v^2 z}{gx}$$

Equation 1: Back Tipping Angle

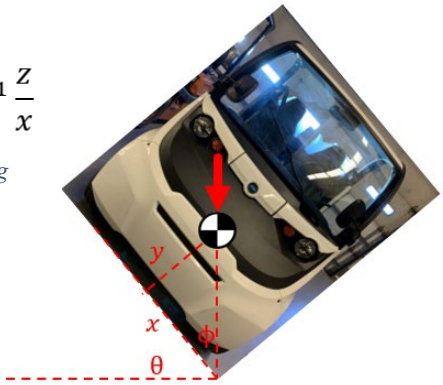




Finally, given the calculations below, our tipping angles were found to be acceptable as well given that the maximum angles for it tipping on its side was 65.69° and for tipping on its back, the maximum angle was found to be 56.79°

$$90° = \theta - \phi$$
$$\theta = 90° - \phi$$
$$\theta = 90° - \tan^{-1}\frac{z}{x}$$

Equation 2: Side Tipping Angle



$$90° = \theta - \alpha$$
$$\theta = 90° - \alpha$$
$$\theta = 90° - \tan^{-1}\frac{z}{y}$$

Costs

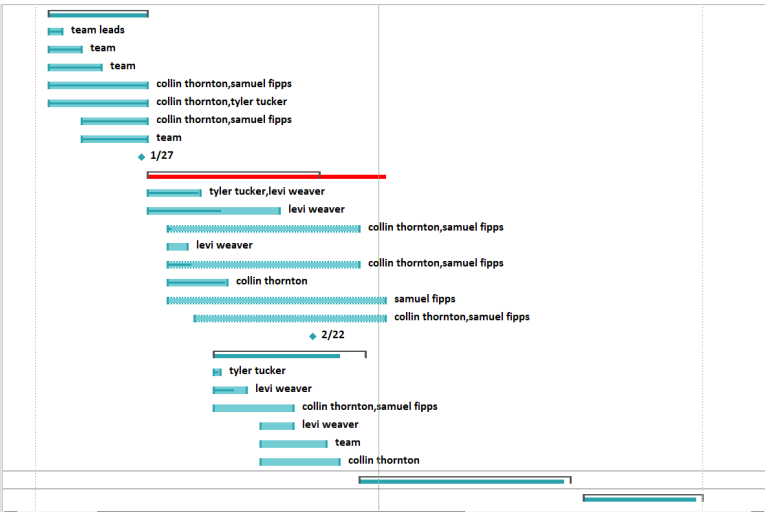
Currently, we expect to stay within our $10,000. The mechanical team has currently purchased most of the required parts, while the electrical team has purchased most of their required materials and devices. Below is a tabling showing the amount spent for different subsystems. For more information regarding our team's spending, refer to the purchasing sheet for a more complete breakdown.

Subsystem	**Amount**
Steering	$4970.76
Brakes	$2369.10
Emergency Brakes	$297.30
Electrical	$775.37
Miscellaneous	$600.84

Project Plan

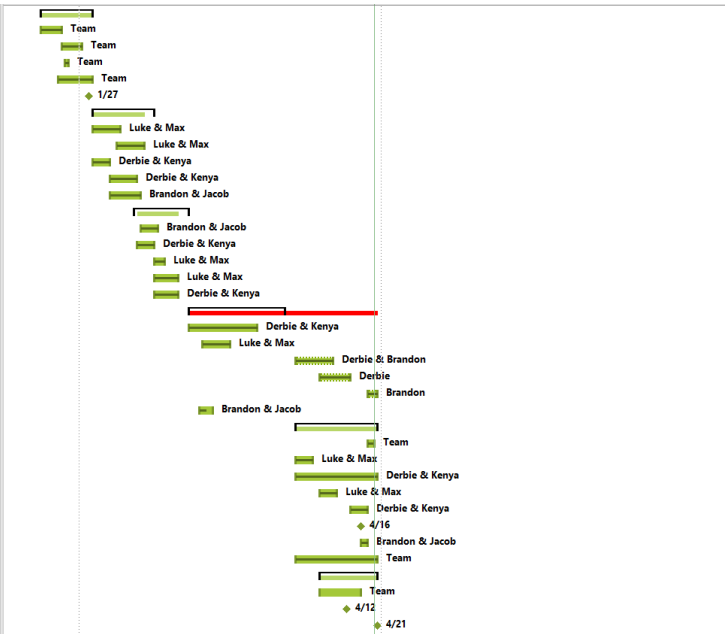
Electrical Plan

	Initial Design	11 days	Thu 1/13/22	Thu 1/27/22
	Dividing tasks	2 days	Thu 1/13/22	Fri 1/14/22
	Subsystem Breakdown	3 days	Thu 1/13/22	Mon 1/17/22
	Pick Design Route	6 days	Thu 1/13/22	Thu 1/20/22
	Threading & I/O Code	11 days	Thu 1/13/22	Thu 1/27/22
	Initial PCB Design	11 days	Thu 1/13/22	Thu 1/27/22
	API Layout	8 days	Tue 1/18/22	Thu 1/27/22
	Finalize PDR	8 days	Tue 1/18/22	Thu 1/27/22
	PDR	0 days	Thu 1/27/22	Thu 1/27/22
	Detailed Design	18 days	Fri 1/28/22	Tue 2/22/22
	Final PCB Design	6 days	Fri 1/28/22	Fri 2/4/22
	Wire Routing	14 days	Fri 1/28/22	Wed 2/16/22
	Finalize API	21 days	Mon 1/31/22	Mon 2/28/22
	PCB Case Prototype	3 days	Mon 1/31/22	Wed 2/2/22
	Code low level classes	21 days	Mon 1/31/22	Mon 2/28/22
	Write CAN comms	7 days	Mon 1/31/22	Tue 2/8/22
	Write parameter server	25 days	Mon 1/31/22	Fri 3/4/22
	Unit test CAN & parameter	21 days	Fri 2/4/22	Fri 3/4/22
	CDR	0 days	Tue 2/22/22	Tue 2/22/22
	Fabrication	17 days	Mon 2/7/22	Tue 3/1/22
	Order PCB	1 day	Mon 2/7/22	Mon 2/7/22
	Power system analysis	5 days	Mon 2/7/22	Fri 2/11/22
	Write application code	10 days	Mon 2/7/22	Fri 2/18/22
	Waterproofing	5 days	Mon 2/14/22	Fri 2/18/22
	Unit Testing	8 days	Mon 2/14/22	Wed 2/23/22
	Write OBC driver	10 days	Mon 2/14/22	Fri 2/25/22
	Testing and Validation	24 days	Tue 3/1/22	Fri 4/1/22
	Final Documentation	14 days	Mon 4/4/22	Thu 4/21/22



Mechanical Plan

	Initial Design	11 days?	Thu 1/13/22	Thu 1/27/22	
	Pick Design Route	4 days	Thu 1/13/22	Tue 1/18/22	Team
	Select Parts	4 days	Wed 1/19/22	Mon 1/24/22	Team
	Divide into Subteams	1 day	Thu 1/20/22	Thu 1/20/22	Team
	Finalize PDR	8 days	Tue 1/18/22	Thu 1/27/22	Team
	PDR	0 days	Thu 1/27/22	Thu 1/27/22	
	Detailed Design	12 days?	Fri 1/28/22	Mon 2/14/22	
	Final E-Brake Design	6 days	Fri 1/28/22	Fri 2/4/22	Luke & Max
	Plan Accenuator Instalation	6 days	Fri 2/4/22	Fri 2/11/22	Luke & Max
	Finalize Master Cylinder	3 days	Fri 1/28/22	Tue 2/1/22	Derbie & Kenya
	Plan Master Cylinder Instalization	6 days	Wed 2/2/22	Wed 2/9/22	Derbie & Kenya
	Plan EPS System Instalation	7 days	Wed 2/2/22	Thu 2/10/22	Brandon & Jacob
	Fabrication	12 days	Wed 2/9/22	Thu 2/24/22	
	Order EPS System	3 days	Fri 2/11/22	Tue 2/15/22	Brandon & Jacob
	Order Master Cylinder	3 days	Thu 2/10/22	Mon 2/14/22	Derbie & Kenya
	Order Linear Acuators	3 days	Tue 2/15/22	Thu 2/17/22	Luke & Max
	Fabricate E-Brake Mount	5 days	Tue 2/15/22	Mon 2/21/22	Luke & Max
	Fabricate Master Cylinder Mount	5 days	Tue 2/15/22	Mon 2/21/22	Derbie & Kenya
	Instalation	20 days?	Fri 2/25/22	Thu 3/24/22	
	Install Master Cylinder	14 days	Fri 2/25/22	Wed 3/16/22	Derbie & Kenya
	Install Emergency Break System	6 days	Tue 3/1/22	Tue 3/8/22	Luke & Max
	Install New Battery Mount	9 days	Mon 3/28/22	Thu 4/7/22	Derbie & Brando
	Install Break Lines	7 days	Mon 4/4/22	Tue 4/12/22	Derbie
	Install E-Stop	3 days	Mon 4/18/22	Wed 4/20/22	Brandon
	Install EPS System	4 days	Mon 2/28/22	Thu 3/3/22	Brandon & Jacob
	Testing & Validation	18 days?	Mon 3/28/22	Wed 4/20/22	
	Competition Rules Validation	2 days	Mon 4/18/22	Tue 4/19/22	Team
	Mount Stress Analysis	5 days	Mon 3/28/22	Fri 4/1/22	Luke & Max
	Master Cylinder Stress Analysis	18 days	Mon 3/28/22	Wed 4/20/22	Derbie & Kenya
	Emergency Break Testing	5 days	Mon 4/4/22	Fri 4/8/22	Luke & Max
	Master Cylinder Hydraulic Testing	4 days	Wed 4/13/22	Sun 4/17/22	Derbie & Kenya
	Driving Test	0 days	Sat 4/16/22	Sat 4/16/22	Team
	Power Steering Testing	2 days	Sat 4/16/22	Sun 4/17/22	Brandon & Jacob
	Misc. Debug Time	18 days	Mon 3/28/22	Wed 4/20/22	Team
	Final Documentation	13 days	Mon 4/4/22	Thu 4/21/22	
	Write Report	10 days	Mon 4/4/22	Fri 4/15/22	Team
	Final Video	0 days	Tue 4/12/22	Tue 4/12/22	Team
	Transport to EXPO	0 days	Thu 4/21/22	Thu 4/21/22	Team



Risk Management

Semiconductor shortage

Any part that has semiconductors in is in short supply because of the supply chain issues so identifying and ordering those parts early is very important.

Fabrication Delays

Along with supply chain issues there are labor shortages, and many places are having delays in fabrication, doing most of our own fabrication will prevent this.

Scope Creep

Keeping the design within our specified objectives will keep the project from not being completed on schedule.

Fall over from the autonomous system to driver control is an important part that we will be taking great care to ensure that this system is built properly and still usable when the vehicle is used as purely human controlled. We will also be looking at what to do in the case of an accident with the Polaris

Work Breakdown

Collin Thornton – Electrical Team Leader

Name	Work
Derby Whitefield	Mechanical Team Lead
Max Minnick	Emergency Braking System
Luke Johnson	Emergency Braking System
Brandon Dang	Mechanical Gantt Chart & Steering
Levi Weaver	Electrical Design Concepts, Power Analysis
Tyler Tucker	Team Lead, Electrical Design Concepts, PCB Design, Administration
Collin Thornton	Electrical Design Concepts

Samuel Fipps	Framework Development,
Kenya Williams	Design Matrix
Jacob Schoeling	Automated Steering

Appendixes

Center of Mass Calculations

$$z = \frac{z_1 m_1 + z_2 m_2 + z_3 m_3}{m_1 + m_2 + m_3}$$

	Distance from Center	Weight
High	17 in	160 lbs
Mid	15.75 in	678 lbs
Low	30.5 in	361 lbs

$$y = \frac{y_1 m_1 + y_2 m_2 + y_3 m_3}{m_1 + m_2 + m_3}$$

	Center of Mass	Weight
High	30.5 in	160 lbs.
Mid	15.75 in	678 lbs.
Low	17 in	361 lbs.