Cornell Autonomous Bicycle
Steer-by-Wire Sub-team

Olav Imsdahl, Max Kester, Conrad McCarthy, Dylan Meehan and David Miron

Fall 2017

Contents

1 Mission

The steer-by-wire (SBW) subteam is building a new self-balancing bicycle that aid users in riding a bike. We will implement a steer-by-wire method of steering, in which the user remotely, rather than directly, controls the turning of the bicycle's front wheel. The front wheel itself will turn according to the lean of the bicycle, and the steering input from the user. The bike alone is self- balancing but can also be steered by the user.

Last semester we were able to obtain parts for this bicycle, design and build the mechanical system, and perform preliminary testing of the front motor controller. We tested the gains of our proportional-derivative (PD) controller so that a rider could lean to successfully ride and steer the bike by the end of the semester. It had a stiff handlebar and the bike itself was self-balancing.

This semester we obtained a new motor and motor controller, designed a new power transmission system for the front wheel's rotation, mounted the handlebar, and implemented user input from the handlebar's rotation. We tested our PD controller gains and began using the handlebar's position and the bike's lean rate to control the front wheel. This allowed the user to ride a self-balancing bike and also be able to steer the bike by turning the handle bar

Future work of the subteam will include fine-tuning the gains of the PD controller, testing the gains of the balance controller, implementing a second motor to supply torsional resistance to the handlebar, and developing a method to test quantitatively whether a rider can stabilize better on our bike.

2 New Hardware

Conrad McCarthy: cdm223

We replaced our brushless motor and chain drive transmission to the front wheel with a brushed motor and a belt drive. Since we control the wheel between a small angle range and at a relatively low angular velocity, we noticed our previous motor produced a distinctive ticking effect that destabilized our wheel's motion at certain positions. The ticks occurred at angles where a gap existed between the motor's stator windings. At these positions, the motor's rotor experiences a lower magnetic force comparatively. Therefore, when we attempted to stabilize the front wheel at a position which corresponded to one of these ticks, the nature of the motor and PD controller produce an undesirable jitter. Our new brushed motor does not produce the same effect. Refer to our Appendix for information regarding our motor specifications (6.1) and our new motor controller (**??**).

We needed to adjust our gear ratio between the motor and the front wheel

because of new motor's gear box specifications (refer to Appendix 6.1). We decided to abandon our chain drive and use a belt drive instead because it is smoother and quieter.

We secured our handlebar to the bicycle by designing an assembly which connects to the motor support assembly (Figure 1). By doing so, we could ensure that our handlebar is collinear to the shaft of our front wheel. This allows us to insert a coupling to connect the handlebar and the front wheel so we can ride and test the bike in its normal operative state. We therefore have a reference to analyze how well our self-balancing and steer-by-wire systems work. As a temporary means of providing torsional resistance to the handlebar, we attached a linear spring to the handlebar's collar and to one of its support plates.

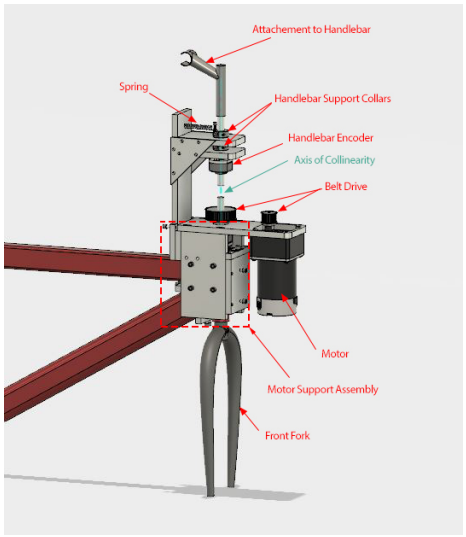

Figure 1: CAD model of Steer-by-Wire mechanical assembly[1]

To read our handlebar's position, we use another rotary encoder. For simplicity, we use the same encoder as the front wheel's encoder. Reference our Appendix (6.2) for a discussion on how an optical encoder works and how our bike implements two encoders simultaneously.

Finally, we added protective cases for our motor controller and inertial measuring unit (IMU). Our Appendix (6.3) details the designs of these cases.

[1]Email Conrad McCarthy (cdm223@cornell.edu) for access to the CAD file.

3 Code Structure

Dylan Meehan: dem292

The Steer-By-Wire bike has 3 modes.

1. Normal Bicycle Mode: The handlebar is mechancially coupled to the front wheel, as in a normal bicycle.
2. Steer-By-Wire Mode: The front wheel of the bike mimics the handlebar one-to-one. There is no physical connection between the handlebar and front wheel. There is no balance controller.
3. Balance Controller Active Mode: The balance controller steers the front wheel in order to keep the bike balanced. The front handlebar inputs to the balance controller to indirectly control the direction of the bicycle.

Mode 1 can be enabled or disabled by inserting a mechanical coupling between the front wheel and the handlebar. Mode 2 is enabled by setting the boolean variable `IS_BALANCE_CONTROLLER_ON` to `false` in the file `ROS_ARDUINO_WRAPPER`. Setting `IS_BALANCE_CONTROLLER_ON` to `true` enables mode 3.

3.1 Steer-By-Wire Mode (Mode 2)

Figure 2 shows a flowchart of the software In Steer-By-Wire mode.

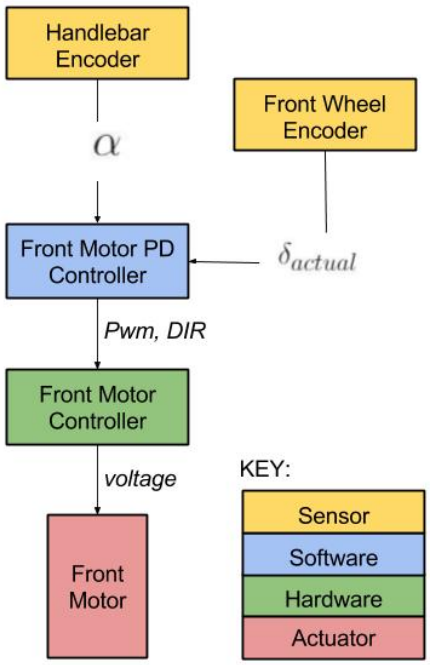

Figure 2: Flowchart of Steer-By-Wire Mode

symbol	description	Name in software
δ_{actual}	position of front wheel	`encoder_position_W`
α	position of handlebar	`encoder_position_H`
pwm	front wheel commanded speed	`PWM_front`
DIR	front wheel commanded direction	`DIR`
voltage	voltage to front DC motor	

The Front Motor PD controller attempts to minimize the error between δ and α.

3.2 Balance Controller Active (Mode 3)

Figure 3 shows a flowchart of the software with the balance controller running.

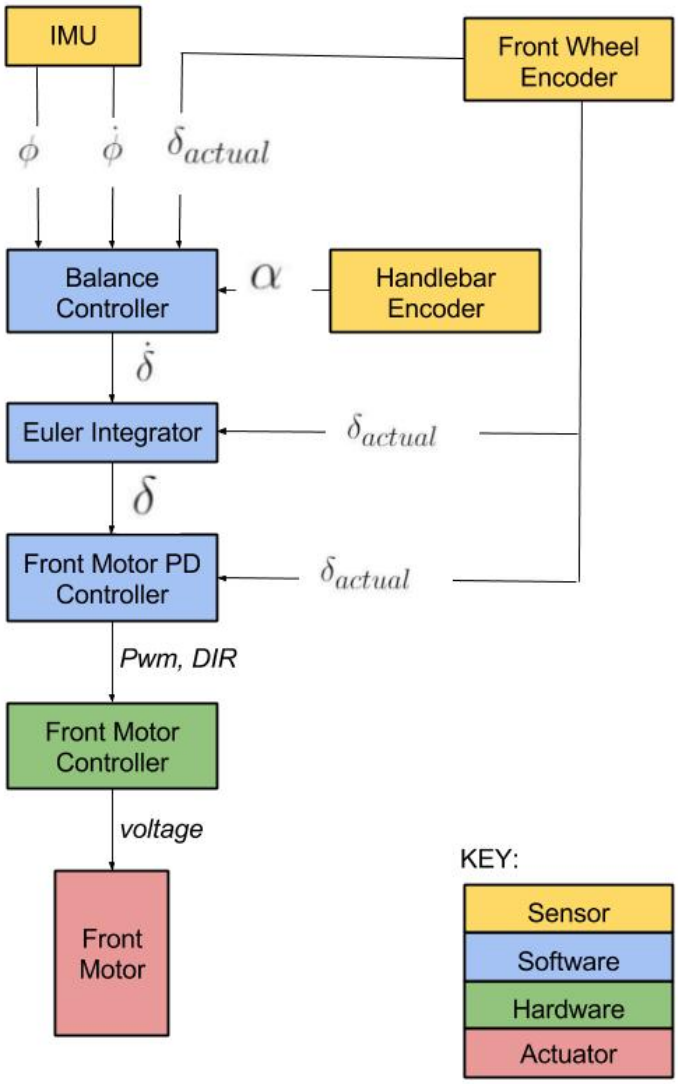

Figure 3: Controller Structure with Balance Controller Engaged

symbol	description	Name in software
δ_{actual}	position of front wheel	`encoder_position_W`
α	position of handlebar	`encoder_position_H`
ϕ	lean angle of bike frame	`imu_data.angle`
$\dot{\phi}$	lean angle rate	`imu_data.rate`
$\dot{\delta}$	output steer angle rate from balance controller	`desiredVelocity`
δ	commanded steer angle	`desired_pos_W`
pwm	front wheel commanded speed	`PWM_front`
DIR	front wheel commanded direction	`DIR`
voltage	voltage to front DC motor	

The balance controller outputs $\dot{\delta}$ such that the bike does not fall. The Euler Integrator multiplies $\dot{\delta}$ by the timestep to find δ, the position the wheel should be at at the following timestep. The PD controller attempts to minimize the error between δ and δ_{actual}. α is passed to the balance controller as the parameter δ_{ds} - see Section 4.

4 Balance Controller Development

Dylan Meehan: dem292

The Steer-By-Wire bicycle uses the same form of a balance controller as the prototype autonomous bicycle. The linear equation of the controller is below. See Autonomous Bicycle Team Fall 2016 report chapter 1 for a more detailed explanation of this controller.

$$\dot{\delta} = k_1 \cdot \phi + k_2 \cdot \dot{\phi} + k_3 \cdot (\delta_{actual} - \delta_{ds}) \tag{1}$$

The variables in the linear controller are defined as:

$\dot{\delta}$	steer angle rate of front wheel
ϕ	lean angle
δ_{actual}	steer angle of front wheel
$\dot{\phi}$	lean angle rate
δ_{ds}	Desired steer input to balance controller. $\delta_{ds} = 0$ when the bicycle is commanded to balance in a straight line.

The constants (gains) of the controller are k_1, k_2, and k_3. These are found via MATLAB optimization using the same script used for the prototype autonomous bicycle. See Fall 2017 autonomous bicycle report for more. The following parameters are specific to the Steer-By-Wire bike.

$p.l$	1.067	distance between front wheel and rear wheel contact points (m)
$p.b$	0.5	distance from ground contact point to center of gravity projected onto ground (m)
$p.h$	1.00	height of bicycle center of mass (m)
$p.c$	0.13	Trail- distance between front wheel contact point and front steering axis projected onto ground (m)

The balance controller optimizer tested 750000 gains. The bike's speed was set at $3m/s$. A 1° error in the lean angle was toggled on and off every second. Each set of gains produced a balance score and a path score. $\delta_{desired\,steer} = 0$. See Autonomous Bicycle Fall 2017 report for discussion of the scoring system.

The best gains judged by balance score are:
$k_1 = \quad 7$
$k_2 = \quad 77$
$k_3 = \quad -6$

The best gains judged by path score are:
$k_1 = \quad 48$
$k_2 = \quad 17$
$k_3 = \quad -47$

Moving forward, these gains need to be tested on the Steer-By-Wire bicycle[2].

5 Future Work

5.1 Prototype Balance Testing

Dylan Meehan: dem292

The balance controller needs to be tested on the physical prototype. Data from these tests need to be compared with the MATLAB simulation. A raspberry Pi records this data. The Arduino prints relevant information over a USB serial port to the raspberry Pi which saves the data to a .csv file.

Data should be compared between the 3 different bike modes: 1) Normal bicycle with mechanical coupling between front wheel and handlebar. 2) Steer-By-Wire bike with one-to-one control between the handlebar and front wheel. 3) Steer-By-Wire bicycle with balance controller active. A scoring system needs to be chosen to compare each mode. One scoring system is to sum the lean angle rates over time. The lowest score would then indicate the best balancing bicycle.

[2] An excel spreadsheet called `750000vals_3speed_test.xlsx` contains the full results of the simulation. The spreadsheet is in the Fall 2017 Steer-By-Wire google drive.

5.2 Torsional Resistance to Handlebar

Conrad McCarthy: cdm223

Bike riders rely on torque feedback from the handlebar to balance when riding. In future, we will need an active method of controlling the resistance supplied to the handlebar so we can more accurately replicate the normal feeling of riding a bike in our steer-by-wire system. The simplest way to implement this resistance would be to mimic the power transmission system (motor and belt drive) of our front wheel for our handlebar. Figure 4 demonstrates one way of achieving this mechanically while still using our current support system for the front wheel's motor.

Figure 4: CAD of bike's assembly with motor and power transmission to handlebar

We can measure the amount of torque which our front wheel experiences through the implementation of a current sensor or strain gauges. The current sensor would measure the amount of current which our front motor draws. Alternatively, we would place strain gauges on the handlebar's shaft to measure the angle of twist and calculate torque. We could also use a torque sensor but they are expensive and not likely to be within our budget range.

6 Appendix

6.1 Design Decisions for Front Motor

David Miron: dm585

6.1.1 Brushed DC Motors

The motor we are using is a 24V Brushed DC Motor from Anaheim Automation. Model number is BDSG-83-125. There is a 7.5:1 gearbox on the motor. This gearbox gears the motor down. The motor is geared down again by a 2:1 belt drive. See HERE for specification sheet

This motor is a low cost Brushed DC motor that is marketed for high volume applications and has a high torque. The rated torque is 250 oz-in. The mini-bike has a rated torque of 240 on-in after taking the gear box into account. With our belt drive taken to account, our motor has a torque that is over double the mini-bike's motor. We felt that this torque was high enough to accommodate the larger wheel and increased weight.

6.1.2 Motor Controller

We used two different motor controllers this semester. Initially we used an Anaheim Automation motor controller, but after breaking it deiced to use a Pololu motor controller. (this is the same as the controller on the mini-bike and there are many around the lab)

The model number of the motor controller from Anaheim Automation is MBDC050-050101. It is made for use with brushed motors, and was purchased because it works with our motor and is not very expensive. We ultimately broke this controller by changing the output we sent to the direction pin on the controller without changing the output sent to the run/stop pin on the controller. In general motor controllers take a number of inputs to decide what output is appropriate to drive the motor. The run/stop pins tell the controller whether we want the motor to be moving. By the design of the controller it is necessary to stop the motor, change the direction input, and then run the motor. We did not stop our motor when we changed direction, and this caused too much current to flow in the board, and broke some of the integrated circuits on the

motor controllers circuit board.

We fixed this broken controller but have decided to continue using the Pololu controller. The break was a blown MOSFET, which was visibly damaged.

See HERE for specification sheets.

The Pololu Motor Controller is able to drive the steering motor which uses 24V power supply. The main reason that we choose the Pololu is that there are plenty of them available in the lab, and since they work well with the mini-bike's brushed motor, we don't have to purchase new models.

The Pololu motor controller is powered by a 5V output from the Arduino. It gets voltage to control the motor from our 24V battery. There are two control signal inputs (PWMH and DIR). PWMH controls the speed of the motor and takes in a PWM. We use an output directly from the Arduino for this. The DIR pin controls whether the motor spins clockwise or counter-clockwise. We control this input with a PWM directly from the Arduino that is either set to 0 or 255. (255 is clockwise and 0 is counterclockwise) The motor controllers outputs are OUTA and OUTB, which are control signals transmitted to the motor.

6.2 Arduino Registers: Reading Values from Two Encoders

Conrad McCarthy: cdm223

6.2.1 Hardware Overview

We implemented two optical encoders simultaneously to read the position of the front wheel and the handlebar. Our encoders are quadature encoders which are characterized by three channels, A, B and Z. Each channel outputs a positive and negative signal, dubbed +A/-A, +B/-B, and +Z/-Z, for the purpose of filtering noisy data during signal processing.

An optical encoder uses an internal light and an opaque disc with markings that block and unblock the light as the inner shaft rotates. Photodiodes convert the shutter pattern of light to electrical signals. The photodiodes of the A and B channels read sets markings spaced around the circumfrance of the inner disc and respectively offset from one another. As the shaft rotates, the A and B channels output two square waves shifted by 90 degrees. The phase shift of the two waves indicates the direction of the rotation during signal processing and the period of the wave can be used to caluclate angular velocity. The Z channel reads only one marking on the circumfrance and subsequently outputs one pulse per revolution. This signal, called the index of rotation, serves to define the relative position of the inner shaft.

The signals from our optical encoders are processed by line recievers on the PCB and quadature decoders native to the Arduino Due. The line recievers filter (separately) the +A/-A, +B/-B, and +Z/-Z signals to single A, B, and Z signals. The Arduino's decoders in turn read these three signals to process position or velocity data according to their software-enabled registers.

See HERE for the Quadatrue Decoder specification sheet.

6.2.2 Software Overview

The Arduino IDE is a tool which simplifies the coding structure of the AVR microcontroller. The inherited Arduino library includes a vast array of packages and functions which makes it easy to define pin functionality at a high level. To access and control certain intricacies of the microcontroller, such as its peripheral devices, a deeper knowledge of the AVR architecture is required.

I recommend the following tutorial to learn the underpinnings of coding in Arduino IDE: A Tour of the Arduino Internals: How Does Hello World Actually Work?

In the setup of our optical encoders, we differentiate between variables which read signals from the front wheel's encoder and the handlebar's encoder by the annexation of '_W' and '_H', respectively. We first define the physical Arduino pins which read the A, B, and Z values from the two line receivers. We call the function "digitalPintoBitMask()" as necessary to convert the physical pin number to an 8-bit byte.

The quadrature decoders (QDEC) of the Arduino are peripheral devices. In order to use a perpheral device, one must first disable the normal input and output functionallity of the pins that the device is connected to. One achieves this by manipulating Parallel Input/Output (PIO) Controllers of the AVR.

See HERE for specification sheet for Using a 32-bit AVR PIO Controller
The Arduino has multiple PIO controllers. For the pins connected to our front wheel encoder we manipulate Controller B and for the pins connected to our handlebar encoder we manipulate Controller C. We set the PIO Disable Register (PDR) and the AB Select Register (ABSR) of these controllers to the bit-masked pins of our encoder signals (A, B, and Z) such that the peripheral functionality of the pins are activated and the correct device (QDEC) is chosen. These specific registers are included in the "instance" header files of the Atmel SAM3U package of the Arduino Due's firmware.

The QDECs are present in the Arduino's Timer Counter (TC) which requires the configuration of several clock peripherals to operate. We activate clock peripherals TC0, TC1, and TC2 by setting Peripheral Clock Enable Registers 0

and 1 (PCER0/PCER1) to the correct pin IDs (PID 27-35). These registers are found in the Power Management Controller (PMC) instance of the Atmel SAM3u package of the Due's firmware.

Finally, we configure TC0 and TC2 such that they store Counter Values (CV) corresponding to the position and index of our front wheel encoder and handlebar encoder, respectively. As stated in the QDEC datasheet, we set Channel Mode Register 0 (CMR0) of TC0 and TC2 to hexadecimal 0x101 (or equivalently, number 5). We then configure the Block Mode Registers (BMR) of both TC0 and TC2 to activate the QDEC in its position measure mode (opposed to velocity measure mode) with no preset filters. To capture the index value, we set the QDEC Interrupt Enable Register (QIER) to HIGH (1). The index for the wheel and handlebar are stored in variables REG_TC0_CV1 and REG_TC2_CV1, respectively. Angular position relative to the index for the wheel and handlebar are stored in REG_TC0_CV0 and REG_TC2_CV0, respectively.

6.3 Protective Cases for Hardware

Max Kester : mk2368

Considerations such as wire length, spacial constraints, and function led to the placement of circuit boards at various locations on the bike. This meant that electronics not housed in the rear compartment would be exposed to the environment. Aiming to mitigate any possible damage and prevent any foreign objects from interacting with these components, we decided to design and build cases which would cover the electronics.

Once designed, the final part would be 3-D printed using Acrylonitrile Butadiene Styrene (ABS) or Polylactic Acid (PLA), the two materials available in the printing lab. This is the most efficient method because rapid-prototyping allows us to quickly make new models after changes have been made. For the case, either material would suffice and thus no material considerations and analysis was made.

6.3.1 Motor Controller

Since we were unsure of where we wanted to mount the MC itself, the case could only cover and be screwed from the top of the board, to allow quick and easy mounting on the bottom to the bike frame. Another important design feature was that the two LEDs, current limiter, and outgoing wire ports had to be accessible. Following the geometry and dimensions of the circuit board from an online schematic, the remaining design considerations were flexible.

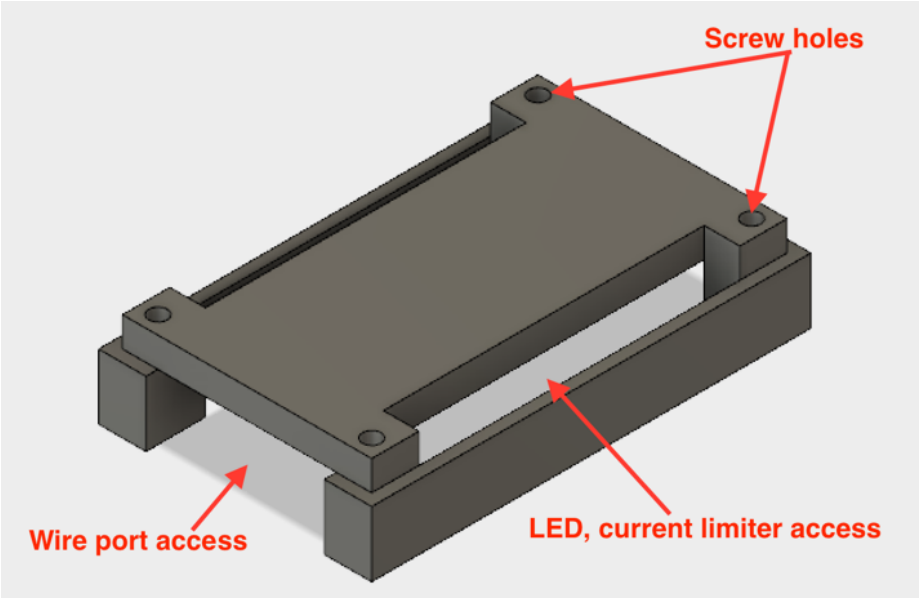

Figure 5: Design 1, Fusion 360

The result is a simple, block-based design. Two walls a tenth of an inch thick (2.54mm) are joined by a canopy to cover the top. Olav recommended a 2-3 millimeter minimum material thickness from prior experience to prevent any breaking in the material. Per the online schematic, the screw holes are 0.156 inches in diameter, and the cover encompasses the 2 x 3.5 inch board. The shape of the cover allows for access from above to the LEDs and current limiter, and horizontal access to the wire ports, which are all conveniently located on the edges of the circuit board. The advantage of this design is that it is simple and effective. A disadvantage of this design is that the access space for the LEDs/limiter is wide and exposes unnecessary parts of the board to the environment.

The first design functioned just as intended, but a second design would be necessary for two reasons. Firstly, we wanted to add engraved labels on the top surface of the case for each wire port for clarity. Secondly, I wanted to maximize protection by minimizing open space and having holes only where needed. Lastly, I wanted to utilize different Fusion 360 design techniques to further familiarize myself with the program.

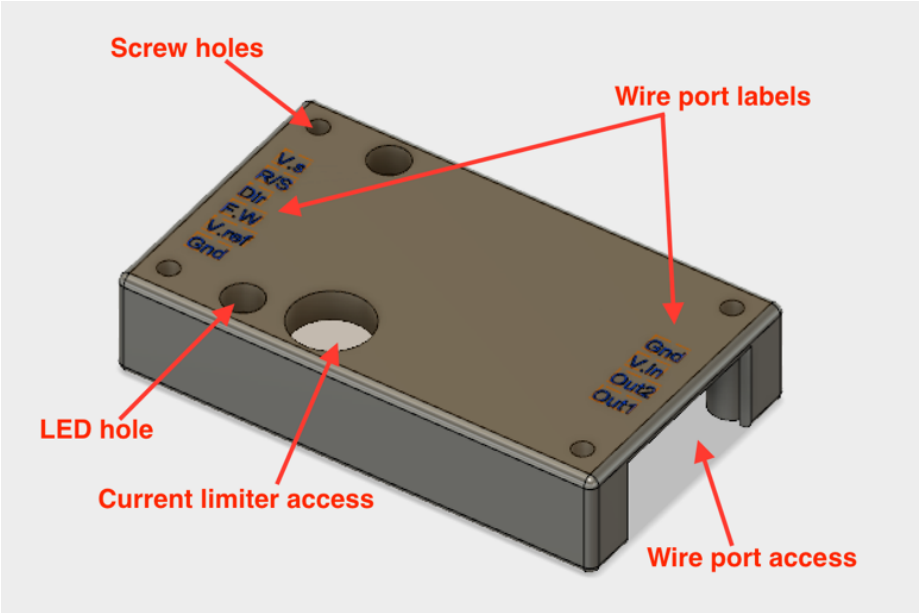

Figure 6: Design 2, Fusion 360

Not only was the design process different but it was easier; I used the "shell" feature to remove space and create the case cavity. This meant rather than having to add and connect blocks into a desired shape, I was able to remove negative space with one command. I rounded the corners just to make handling the object smoother. The LED and current limiter hole diameters are 0.1 inches greater than the measured diameter of related component. Compared to Design 1, this model appears more elegant, probably due to its more efficient design process. Additionally, this model covers all the non-essential space on the board and provides users with information on its contents.

However, a manufacturing limitation created two errors in the final product. The 3D printer is not extremely accurate on this small scale, and tends to add material on the inside of hole features. Thus, the screw holes were misaligned and the engraved labels were illegible and the case was essentially useless. The necessary edits – enlarging the screw holes' diameters and the size of the letters – have been made and the new version was successfully applied.

Figure 7: Applied Case

6.3.2 Inertial Measurement Unit

Continuing with the task of semi-waterproofing and protecting our important and exposed circuit boards, the Inertial Measurement Unit (IMU) was next. The specifications would remain the same – it needed to cover as much electronics as possible while accommodating its geometry and operation. This proved to be more complicated than the previous case for two reasons: various components broke the boundary of the board, and the board is screw mounted from the bottom. This meant that the case could not be screw mounted (like the MC case) to restrict z-axis motion.

The mounting technique we explored was a clamp system that would hold with compressive forces created by the case geometry. Getting the basic shape was easy based on the IMU dimensions, but optimizing certain dimensions was necessary to ensure easy and unobstructed clamping. This proved to be quite an iterative process, because multiple variables – clamp length, distance apart, and stiffness – all contributed to the case's performance.

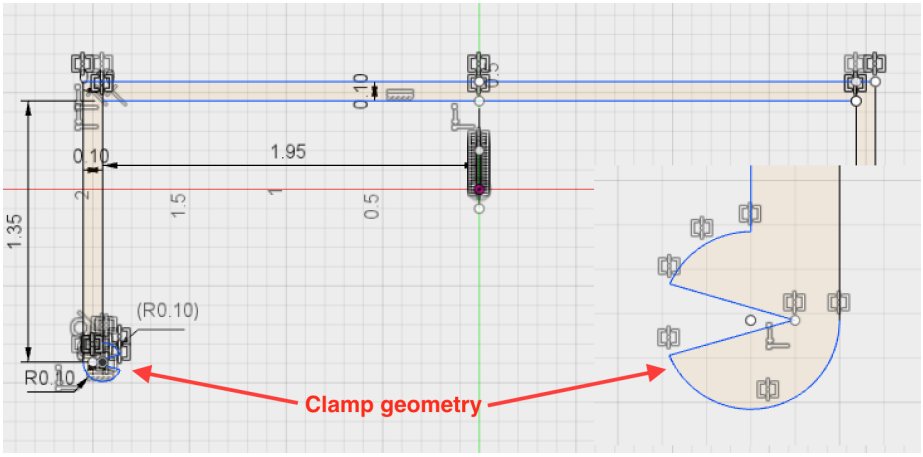

Figure 8: Case Cross Section, Fusion 360

Printing multiple half-inch thick sections of the space with ranging dimensions allowed us to narrow down the desired magnitude of stiffness for the clamps. To allow for accurate transmission of these tested forces/interactions with the board, the rest of the case was completed by leaving slits for the clamp arms, which leaves them unconstrained by the walls and free to strain as necessary.

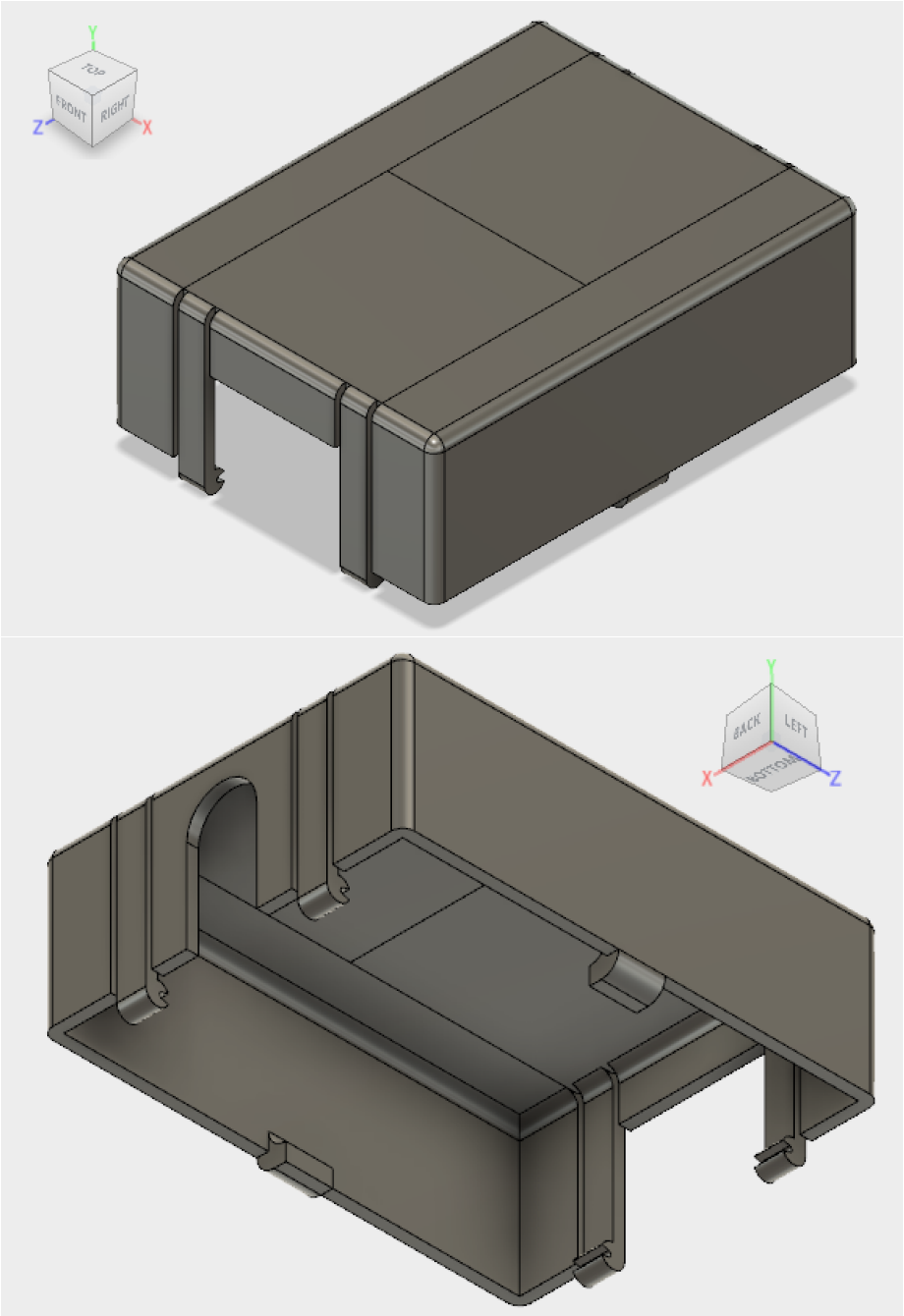

Figure 9: Design 1, Fusion 360

Applying a full case was more challenging than the testing sections; four clamps needed to be in position versus two, and the walls prevented that from being easy. Additionally, increased weight from the added material made the case heavier and slightly more unstable. Thus, further minor revisions were needed.

6.4 Conclusion and Future Applications

Conclusion:
The goal of this past year was to design, develop and build a working steer-by-wire prototype to aid in further testing of the interaction between a bike and it's rider. Throughout the project we ran into some issues with the front motor and controller as we found it difficult to control and later broke some of the new components. The bike is currently ride-able, self-stabilizes and can be controlled by user input through leaning and steering.

Future applications:
Going forward there a many tests that can be conducted. Without the handlebar torque feedback for the user it will be interesting to see how easy the bike is to ride. With some tests and data-logging we can also see if our balance controller can balance the bike better than a human and also how they work together. This bike can be tested with people who have never learned how to ride a bike to see if the bike really does self-balance with a rider.