Pedals Subsystem:

The purpose of this subsystem is to translate the rotational movement of the bike's pedals to the DC motor resistance system. The main constraint is that no major modification can be made to the bike so that it may still be rideable on the road. The design outlook has changed since design phase 2 was written. It has evolved into an easier method, while still maintaining the specs and constraints.

To test this subsystem, a rider pedaled the bike and it was demonstrated that the bike tire roller was able to transfer the motion through a flexible shaft to the DC motor sitting on the floor. No modifications were made to the bike besides adding a couple of bolts to the rear axle so that it may sit in the bike trainer more securely.

Various riders have indicated that having the pedals attached to the DC motor does not hinder gameplay, but it does add a noticeable amount of resistance even when the motor is not being braked. Therefore, it can be concluded that the subsystem is successful and no further improvements are necessary for base functionality of the subsystem.

The following video demonstrates the experiment:

https://drive.google.com/file/d/1FQCbuubpCnskCKTeCOQMqnzUGAGgylcD/view?usp=sharing

Steering Subsystem:

The purpose of this subsystem is to transfer the rotational movement of the bike's handlebars to a potentiometer that will convert it into game data. The constraints are that it must be modular and removable when not in use. The design has changed slightly since phase 2, but the basic idea is the same. Two gears translate the motion to the potentiometer, and the whole assembly is removable via screws. Although, the gear attached to the bike doesn't have to be removed as it doesn't hinder normal operation of the handlebars.

To test this subsystem, a rider turned the handlebars 180 degrees at varying speeds and forces. The gears can turn further than 180, but no sane rider would need to do that in-game. It has been found that the steering control could be tuned to provide more steering per degree of rotation, but that will be fixed in software. As far as the physical subsystem goes, the design is successful and no changes need to be made for base functionality.

The following video shows a demonstration of the experiment:

https://drive.google.com/file/d/1FfzsU2vSYcDkufuFrZ7AIU6jiKNoKl8x/view?usp=sharing

Frame Subsystem:

The purpose of this subsystem is to suspend the bike and the rider off the ground. This is to ensure that the front and rear wheels can move freely to serve functionality for other subsystems. It also must hold the game's display for an arcade-style gaming experience. The constraints are that the frame must be height-adjustable, foldable and/or easily disassembled, and rigid/wide enough to support any rider and prevent tipping over.

The design of the frame was not changed from phase 2 except for the way the DC motor is mounted. It is no longer secured to the bike frame. It instead sits on the floor with rubber feet. The rest of the frame was built to spec. To test the strength of the frame, a rider stood on the bike, leaned both ways, and pedaled. The monitor mount was also nudged to test for wobble and stability. The results were good. The frame did not exhibit any significant wobble. The monitor can be wobbled by hand, but it does not move while the rider is pedaling. The bike is held off the ground securely. The only way that the constraints were broken is that the front wheel had to be removed. However, it is easily reattached with two bolts and it even makes steering easier and more fluid to have no front wheel.

The following video shows the experiment:

https://drive.google.com/file/d/1FirlIeNljJ6o62IekmWVKg16TLf-UYEZ/view?usp=sharing

DC Motor Resistance Subsystem:

The purpose of the DC motor is to provide dynamic resistance to the pedals of the bike and therefore the rider. The main constraint is that the system must be self-contained and unable to harm anyone using it. This DC motor outputs high voltage and current so this is imperative. Several precautions have been made to prevent shock.

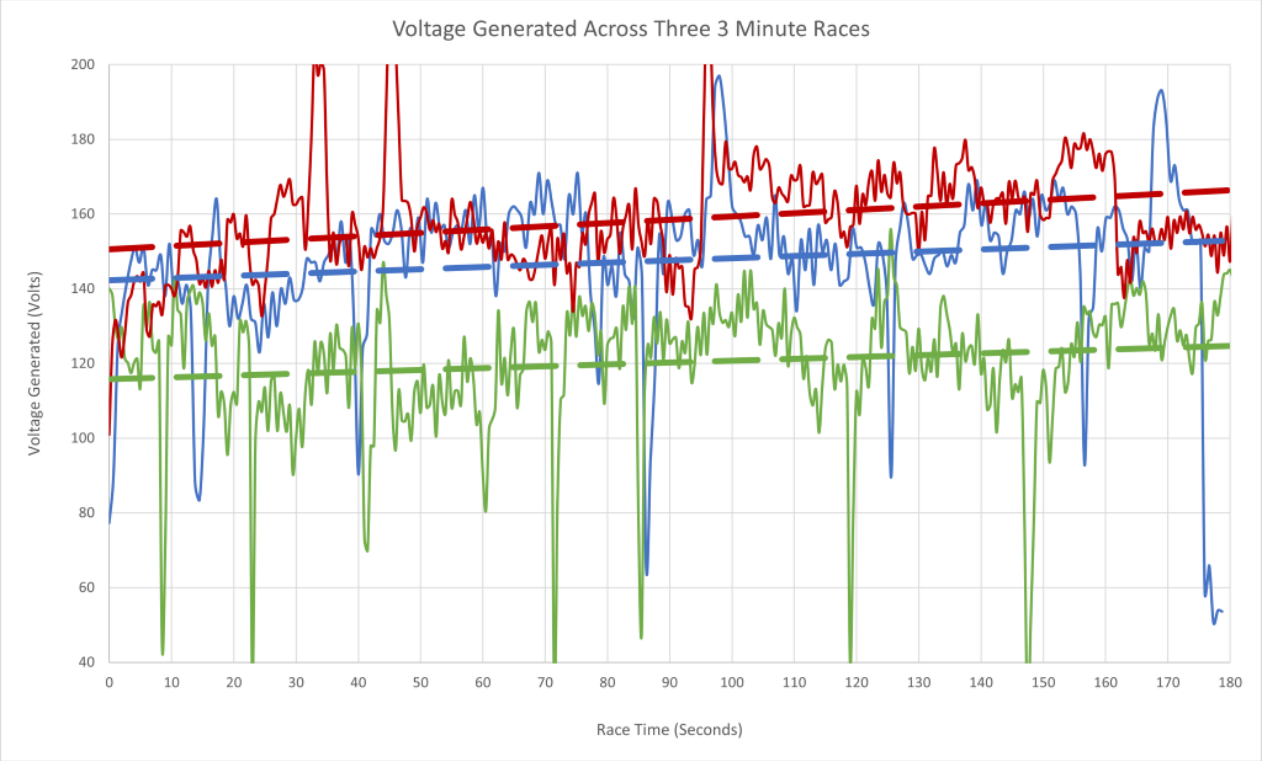
The purpose of the MOSFET is to provide a way to control the braking of the motor. However, through extensive testing, it has been found that the selected MOSFET will not work for two reasons. One is that the transistor is rated to dissipate 160W while switching. It has been found that the motor can generate up to 260V and 1.2A, which obviously leads to a power much higher than 160. The second reason is that this is a typical n-channel MOSFET, meaning its intended use is as an on/off switching device. By operating it in the linear region, we are

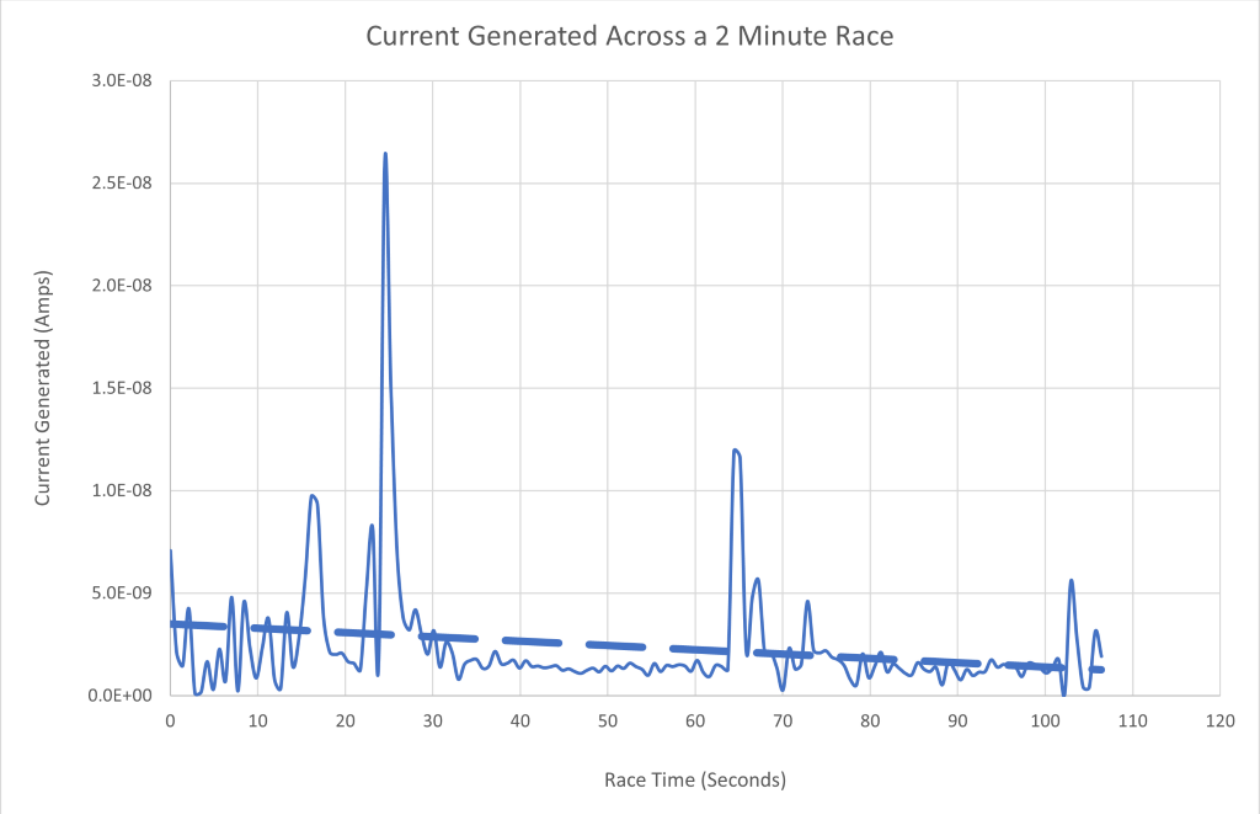
introducing a catastrophic amount of internal heat. This has led to two of the MOSFETs internally shorting and becoming unusable. Additionally, the chips meant to measure the current and voltage generated at all times are not functional. In phase 2, it was thought that the chips could be powered by the DC motor. However, that is not the case. The op amps and hall sensor need a dedicated power source to be functional. Therefore, the measurements conducted in the experiments below were done by a DMM.

The solution that has been implemented (for now) is to instead put a relay across the motor leads. This is not a permanent solution, but it is what we used for the experimentation that will be mentioned below. The relay serves the same purpose, which is that we can control the high power motor with a low power device (the Arduino). However, the resistance is not dynamic. Because it is a relay, it can ONLY be treated as a switching device. We have implemented a program that randomly turns the relay on for 30 ms, which creates a noticeable jump in resistance for a brief moment. These jumps in resistance happens randomly every 15-45 seconds. This adds an element of exercise on top of what is already present. Driving the motor in an open state already leads to a considerable amount of resistance, along with the random jolts.

Again, the above solution with the relay is only temporary. However, some experimentation has been done using the temporary system. To test the performance of the motor, we had a rider run 4 races. In 3 of them, we measured the voltage generated by the motor. In the fourth, we measured the current generated. The reason we only did one trial of current measurement is because the motor is open for the majority of the race due to the relay. That means that theoretically, zero current will be generated. We did one trial to prove that. For the voltage trials, it was found that an average of 140-160 Volts is steadily generated throughout the race. Further, because "zero" current is being generated, that means a very small amount of power was generated (microWatts). The results have been plotted below.

Experimental Analysis of the Various Subsystems





The dashed lines represent the average across the data spread. The voltage graph has 3 trials on one plot, each taking about 3 minutes. The rider pedaled at various speeds in each trial, which is evident by the slight difference in the average voltage generated when comparing the three. The large dips in the lines were caused by the random jolts from the relay being turned on. The reason the voltage dips so much is because when the jolt happens, the RPM of the motor drops significantly, leading to less voltage generated.

I would consider the results of the experiment successful. The relay worked as intended and added a considerable amount of exercise to the riding experience. It has also been shown that a considerable amount of voltage can be extracted from the system.

In the future, the relay will be replaced with a linear mode MOSFET with high power dissipation. A linear mode transistor is meant to operate in the linear region, which means the high internal heat will be reduced. Along with higher power dissipation, this should lead to a system that functions as originally intended. Additionally, the op amps and hall sensor will be given a dedicated power source and their functionality will be tested later. Once those improvements have been made, we will run the same trials again and compare the results.

A video was not recorded for this experiment because the operation of the resistance system has been demonstrated to the instructor on multiple occasions. It is also showcased in the video for the Pedals Subsystem.

Controls Subsystems:

Steering

As a reminder, the steering subsystem consists of a 10k Ohm potentiometer and a 12-bit ADC chip with an onboard gain amplifier. The potentiometer is connected to the 5V power and ground of the RPI, and its output is fed into the first analog channel of the ADC. The ADC is also powered by the RPI, and its digital output is returned via I2C. Steering is mechanically translated from the user turning the handlebars by a 1 to 1 gear ratio with one piece around the turn shaft of the bike and the other around the knob of the potentiometer. The potentiometer is mounted such that the middle position of the range is facing directly forwards away from the user of the bike. It has 135 degrees of physical rotation available in each direction.

Design Specifications:

We originally expected the user to only practically operate the handlebars in a range of 90 degrees left or right of the center point. This limited our range of the potentiometer to only 180 degrees out of 270 total degrees available or 2/3 of the base range. The ADC chip measuring 12

bits gives 4096 partitions of measurement possible. Over a range of 5 volts, this works out to be 0.0012207 Volts per partition. Because we are only using 2/3 of the voltage range for controls, we expected about 2730 partitions or 3.333 Volts of practical range. The Nintendo Switch controller analog signal for horizontal movement only scales up to 4096 partitions as well, meaning we still planned to have 2/3 of the range of standard controller steering. Given the already high precision of the controller, we had concluded that the output of the ADC should be directly mapped to steering data from the switch, and that the difference in range would be insignificant for standard play. The physical design of the gear system should ideally result in the same turning distance and speed as turning the potentiometer by hand via the knob.

Experimentation:

The steering implementation ended up being relatively close to the specifications. It has the highest priority of all control signals because of its necessary precision and responsiveness. In terms of the range of fidelity the user has when playing the game, the 180 degree range allows access to the 500^{th} to the 3500^{th} partition in the range. However, the 12-bit ADC chip has peculiar gain options that prevent exactly 5 volts from being utilized as the range limiter. The closest choices were either 4.096 V at a gain of 1 or 6.144 V at a gain of 2/3. Selecting the 2/3 gain option allowed us to use the full range, but the value of the voltage affected by each partition changed, so the voltage range the user has access to practically in the 180 degree range is 0.75 V to 5.25 V. Fortunately, the potentiometer and the RPi have no issue with this difference in range and function just as well as predicted. The steering is mapped one-to-one from the returned digital bit value to the horizontal analog left stick input (steering stick in MarioKart). This was possible because, coincidentally, the analog stick in a Nintendo Switch controller also has 12 bits of fidelity for the horizontal position. The slight loss in range on either end of the steering has not proven to be significant to performance due to the gradual steering process in-game. Functionally, this subsystem works exactly as planned. Subjectively, a different approach to the mapping might be beneficial in the future. While pushing an analog stick all the way to one side is relatively easy and makes sense for a hard turn, turning bike handlebars all the way to one side while pedaling has proven to be awkward and difficult to adjust to while playing. An exponential form of mapping, where the partition value being mapped is increased faster as the user turns, should solve this problem. One other issue has appeared recently with the connection between the ADC and the RPi where the I2C occasionally loses connection, but this is most likely due to the connection being between jumper wire headers and not permanently soldered in place.

Video Testing:

https://drive.google.com/file/d/1hmFfyGo91mm45ISRFt6g9mDjMVWMiVlL/view?usp=sharing

Future Experiments:

Once a better mapping for the steering is developed, I would like to test it and compare to the current mapping to see if the quality can be improved subjectively. If I can get a basic exponential function set up to correct increase the rate of steering as the user turns further to each direction, we can tune it after repeated play-testing. Additionally, I think another test with different ADC gain settings selected might be beneficial depending on the result of the new mapping design. It could help add just a bit more fidelity to the turning itself.

Acceleration

The acceleration subsystem is the method to convert the pedaling motion from the user into in-game character driving. It consists of the IR object sensor connected to one of the Arduino Nanos with code running that constantly calculates the RPM based on an equation involving a count of the number of things detected, the times per minute a sample is taken, and the number of things that are sensed on the rotating object. It uses this value to determine the acceleration of the character in-game.

Design Specifications:

The acceleration specification calls for a way to pick-up the pedaling, or more generally the attempted forward motion from the bike user. This data must be communicated over serial data from the Arduino Nano to the RPi and ultimately used to determine the acceleration of the in-game character. Ideally, it would correspond in speed such that the speed of the user equals the speed of the driving "kart". The component that does this must be fast enough to keep up with the approximately 90 – 120 RPM that could be seen at max from the user while pedaling.

Experimentation:

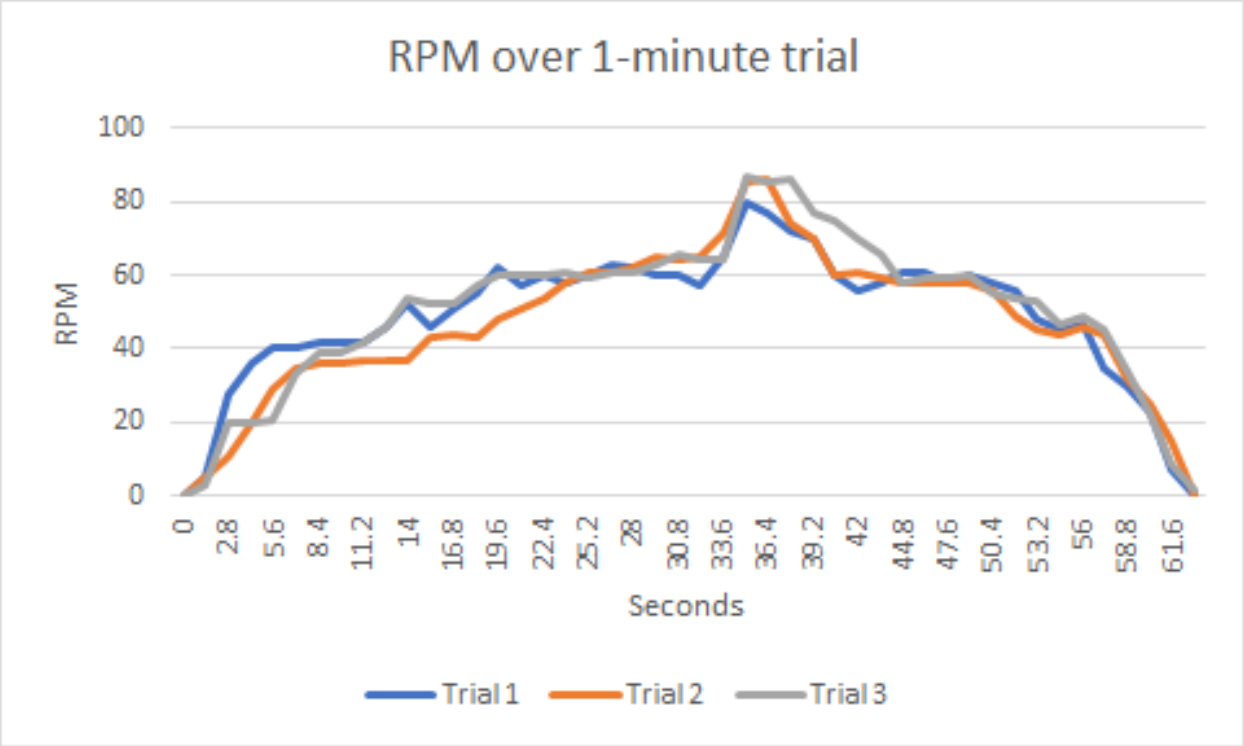
The IR sensor component selected for this subsystem does not inherently measure RPM, however connecting to the Nano and running the number of object detections by the component through an equation can result in a relatively accurate RPM number. This data has been successfully acquired with this method and can be consistently obtained from the user. The RPM data has also been successfully sent via serial data to the RPi and used to determine when the user is accelerating or not. Functionally, this system works to act as the "A" button (accelerate in MarioKart) and the game is fully playable with it as such. However, the original specification of mapping the RPM numerically to the desired speed of the user is not achievable in the way intended in the initial design. The acceleration in MarioKart is essentially either stop or go, determined by holding the "A" button. Although the user can tap "A" repeatedly to simulate a slower level of acceleration, it is not possible to do this fast enough with the control system now

without the character in-game stuttering. In other words, this system is not quite sensitive enough to smoothly vary the speed of the character in a way that looks and feels like the normal operation. Subjectively, this does not affect playability much, and holding a constant acceleration along with quickly slowing down when necessary is still possible, and racing competitively this way is still an option.

Video Testing:

https://drive.google.com/file/d/1hhtGuFOJsfW5j8NDNWvJnVqN-W55GKUF/view?usp=sharing

Below is some RPM data from 1 minute of pedaling. The test begins from a stop, starting at a normal pedaling cadence and then gradually increasing to monitor the sensitivity and consistency of RPM. The middle spike is from a sudden jump in RPM caused by a quick increase in pedaling speed to test the responsiveness of the code. This test was repeated 3 times with the same pattern of speed change each time. From this data and context, the system does a good job of capturing consistent and usable RPM data.



Future Experiments:

A follow-up experiment that may be worth doing is to compare the RPM measured while playing the game to that of a normal bike pedaling on the road. Any difference found could help in understanding the level of exercise the user is getting during the game compared to just riding on a track. Also, I would like to try another experiment involving the PWM of the "A" button, this time using the built-in "tap" function found in the "Joycontrol" software. After looking at the code behind it more, it seems like it will cause less stuttering than my original approach.

Buttons

The buttons subsystem is simply a method of allowing the user to use items and drift in-game using easily accessible brake-style buttons.

Design Specifications:

The original design for the external control buttons was to have standard normally open switches attached to the handle bars to serve as the item-throwing and drifting controls in-game. The buttons would be connected directly to the RPI via GPIO and, when pressed, would return the corresponding button press as the controller button input that it was meant to simulate. It is meant to be quick and responsive enough to feel like a regular switch controller while being conveniently located on the bike handlebars so that is could be pressed during focused playing. The buttons should be able to input separately but also at the same time. Lastly, you should be able to do both the press and the hold functions of a button like a controller.

Experimentation:

The buttons currently implemented in our design are essentially bike brake levers with an embedded normally-closed switch that is opened (signaled) when the lever is pressed down. They work as expected, and are assigned to the ZL and ZR (throw item and drift in-game) buttons on a standard Nintendo Switch controller. They can achieve both press, press & hold, press together, and press & hold together operations. They have the lowest priority in the control reading because they do not need as quick of a response as steering or acceleration. The video in this section shows their operation in the button test menu. Functionally, this subsystem is also a success as the buttons serve their intended purpose well. Subjectively, they are well positioned on the bike, easy to press during gameplay, and quick enough in responsiveness not to notice any delay. However, these particular brake handles tend not to settle back to the closed position all the way when released, which sometimes results in the button being held open for longer than intended. Stronger return springs may solve this problem.

Video Testing:

https://drive.google.com/file/d/1hl8dL9uW5UdxS6Jiiq3ZJV202Tk_pT7L/view?usp=sharing

Future Experiments:

The buttons do not need much more experimentation, but working to fix the springs sticking occasionally will require testing once it is addressed.

Translation Program and Controller Connection

This subsystem consists of the method of translating, restructuring, and packaging the control data from the other subsystems to work on the Nintendo Switch console as a simulated controller. It is based on the open-source "Joycontrol" Github package used for simulation Nintendo JoyCons and Pro Controllers

Design Specifications:

This subsystem is arguably the most important. It is intended to serve as the communication between all of the bike control options and the Nintendo Switch/Mario Kart. Basically, the specification here is that the controls need to be quickly and accurately forwarded by the RPi from the various systems to the Switch and consequently the game. Some less vital specifications also called for the return of vibration data from the Switch and for a typical controller to also be connected so that the user can easily navigate the menus of the game.

Experimentation:

The implementation of this system required the most detailed work out of all the others and, consequently, also has the most issues. The method of translation here is done through a modified version of the "Joycontrol" program. Originally, it was meant to connect to the Switch via Bluetooth, mimic the ID of a Switch controller, and then receive controller "input" from a terminal operated on the RPi. It has now been changed to work with both GPIO signals, I2C connection, and serial data lines for the controller input. The program packages the control data the same way, but now the bike controls can be used for input. In terms of basic functionality, that part of the system is successful. The game can be accurately played through at least an entire race with full control. The return of vibration data is very basic in the default program. There only exists a return signal to tell the controller when vibration is enabled, so the intention to use it as a motor control is not currently functional. Also, because the connections are now done over Bluetooth, a second controller cannot be used for the same user, meaning those portions of the subsystem are not operational at this point. Additionally, the program has proven to be quite

prone to different issues that result in disconnection from the Switch. Loss of connections with the I2C, random Bluetooth drops, terminal freezes due to memory leaks or CPU throttling, and issues with the program running too slowly have all occurred during the testing of this system. Functionally, it works when it needs to, but there is a lot of room for improvements in both the code and the hardware involved in the translation and transmission of data. Subjectively, it works well enough to play 1 to 5 races on average without issues. The occasional error mid-race is annoying, but the novelty of playing the game on a bike helps to make up for it.

Video Test:

https://drive.google.com/file/d/1iGuEhgO6bs-T5icZ7CG_KcLFKqcZzPuW/view?usp=sharing

Future Experiments:

There is plenty of potential for more experiments on this subsystem. Testing the real-time delay of the controls empirically, testing the control load limits to see how much data can be sent at one time before slowdown occurs, and gathering more subjective input from a wider user-base would all be beneficial experiments to consider in the future. I would like to focus first on finding out the reasons behind the occasional connection issues experienced by the system.

Sound and Display

This subsystem represents the method of displaying the video and audio portions of the game conveniently for the user. The Switch dock provides the HDMI output needed for these signals in the system.

Design Specifications:

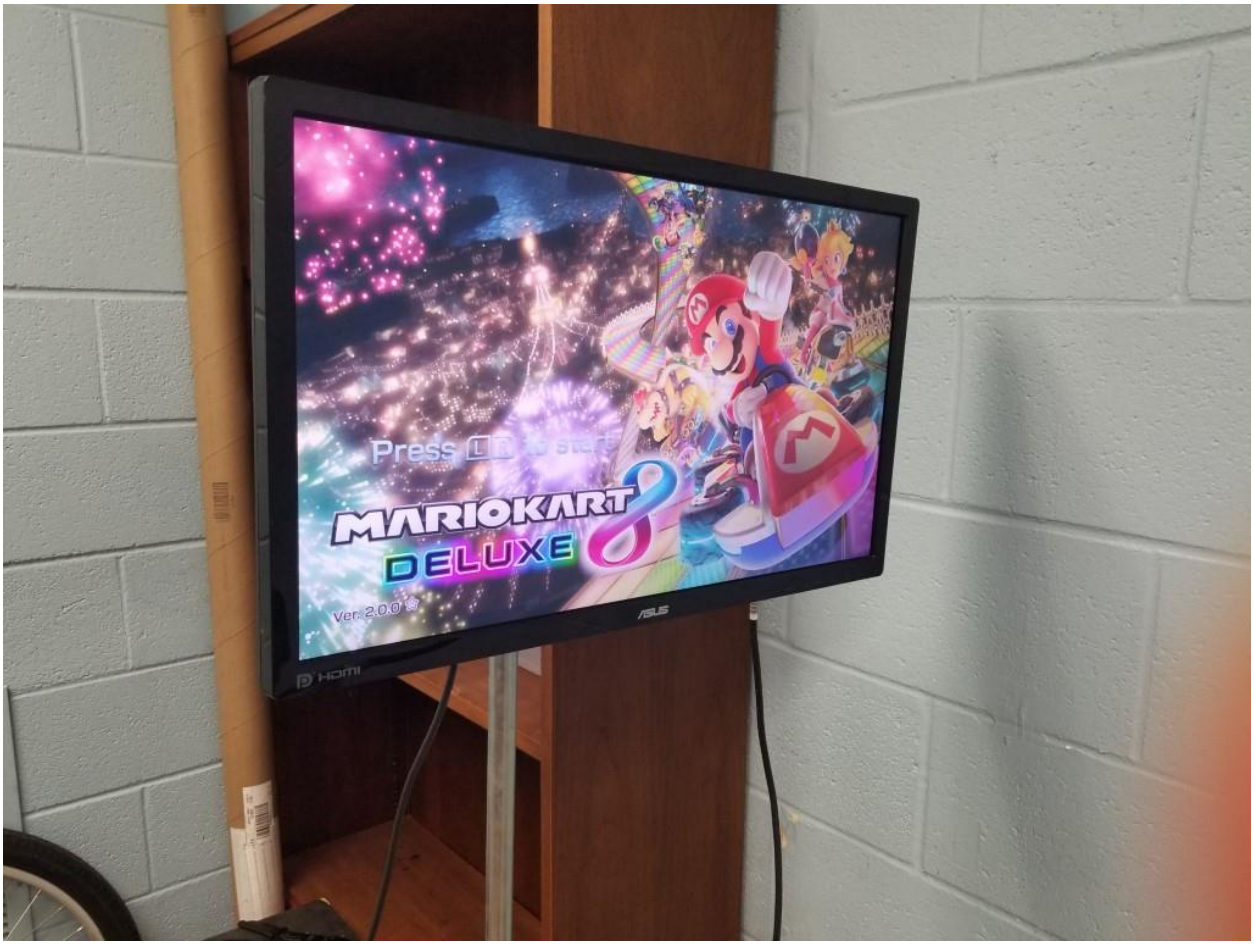
This subsystem simply requires a monitor to display the game in a position that the user can easily see and hear from. It originally called for two monitors and an HDMI splitter, but the second bike design was scrapped for now, so only one display system is needed.

Experimentation:

The implementation of this subsystem was straight-forward. An HDMI-capable monitor with speakers was obtained, connected to the Switch, and mounted to the frame directly in front of the bike. It is sturdy, well-placed, and close enough to comfortably see and hear the activities of the game. Functionally and subjectively, it works perfectly for what it is needed for.

Experimental Analysis of the Various Subsystems

Display Test:



Experimental Analysis of the Various Subsystems

Experimental Analysis of the Various Subsystems

Future Experiments:

We have concluded that this system does not need any more experimentation.