[Virtual Cycling Reality]

By

Bryant Johnson

Chongxin Luo

Gregory Knox

Final Report for ECE 445, Senior Design, Spring 2016

TA: Luke Wendt

1 May 2016

Project No. 47

Abstract

The Virtual Cycling Reality Project is an attempt to bring the modern technology of virtual reality into the home of the average user. The virtual reality system is realized through a bike containing IR sensors to record wheel position and a potentiometer to measure handlebar position. The virtual world was created using the Unity 3D engine displayed using Google Cardboard while fan and resistance systems provide environmental feedback. This paper covers the design of the above components as well as the steps to verify and integrate them. This paper also covers some of the limitations of the current design and future possibilities to improve upon the already immerse bike riding experience.

Contents

1 Introduction

The age of Virtual Reality (VR) is fast approaching, but VR experiences are currently beyond the reach of the average consumer. Dedicated goggle systems and hardware designed to interact with them are prohibitively expensive for the average consumer at $800 for an HTC Vive system. The Virtual Cycling Reality (VCR) Project seeks to bring the power of VR into the homes of the average consumer. This will be accomplished by creating a 3D simulation world in which the user can bike. This world will come to life through a system of sensors, a fan based environmental feedback system, and by harnessing the power of the user's smart phone, coupled with a head mounted display. While other existing virtual biking experiences, such as CycleOps and The Virtual Bike, also attempt to bring a cycling experience into the home, they rely on 2D display measures and lack the environmental feedback that the VCR Project will provide.

2 Design

2.1 Design Overview

In order to achieve our goal of a responsive and low latency virtual reality experience, we have segmented our project into the high level modules shown in figure 1. This model of our project shows the relationship between the various subsections of our design. Our intention throughout the design process was to keep the experience of riding the bike to feel as natural as possible. In order to maintain this, our design included mounting sensors and feedback to an actual bike elevated on a bike trainer. The sensing unit consists of the wheel infared sensor, handlebar encoder, and the phone accelerometer which track the wheel speed, handlebar position, and head orientation respectively. The control unit then probes the sensors attached to the bike and relays the information to the VR module which handles the data processing. The user wears VR goggles that allow him/her to view their location in the simulation as well as look around in the environment. Depending on the current speed of the user as well as their current acclivity in the simulation the Unity program sends a change of state message back to the microcontroller. The microcontroller then alters the userfeed back systems on the bike, which are the fan and resistance systems.

2.2 Top Level Block Diagram

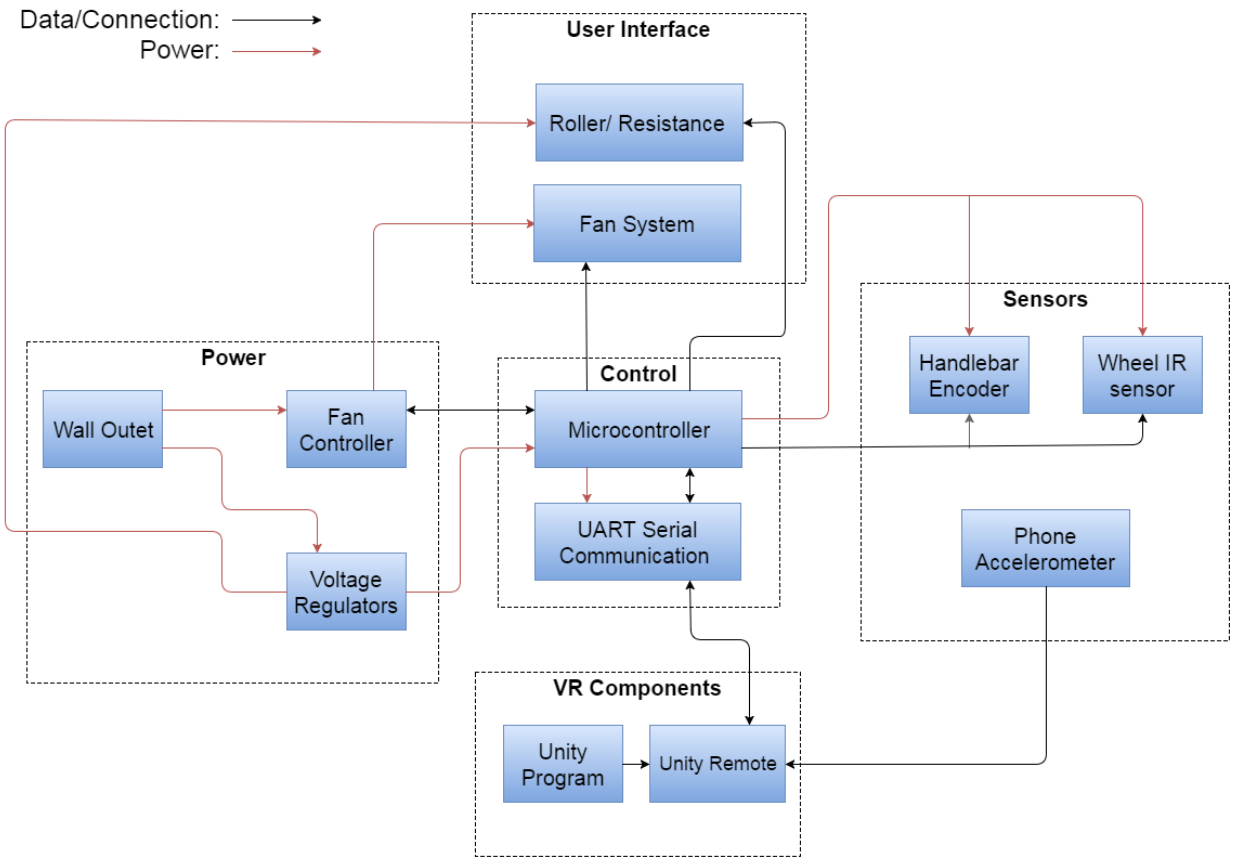

Figure 1: Block Diagram

2.3 Power Module Unit

2.3.1 Voltage Regulator

Power for this project is acquired directly from the standard wall outlet. A 5:1 step-down transformer was originally considered to step down the line voltage for the regulator circuits. The resulting supply voltage of about 34 volts was too high and stretched the thermal limits of the circuit. In order to reduce generation of waste heat and bring the supply voltage closer to the intended 12 V an 8:1 step-down transformer was used in the final design. This transformer lowers the 120 $Vrms$ outlet potential down to 15 $Vrms$ where it is rectified into dc. An output capacitor then filters the pulsed dc signal. The dc voltage is approximately at the peak voltage of the 15 V_{rms} signal ($15 \times \sqrt{(2)} = 21\ V$). Figure 2 shows the schematic of this rectifier.

From the rectified supply voltage two fixed voltage two voltage regulators supplied the circuit with two 12 V supplies. The original design for these regulators was focused around the ACT4514 switching regulator. This device accepts an input voltage up to 40 V and can output up to 12 V and 1.5 A. The circuit for this regulator is shown in Figure 3.

Parameter values for the converters were determined using the datasheet. The feedback resistor divider is

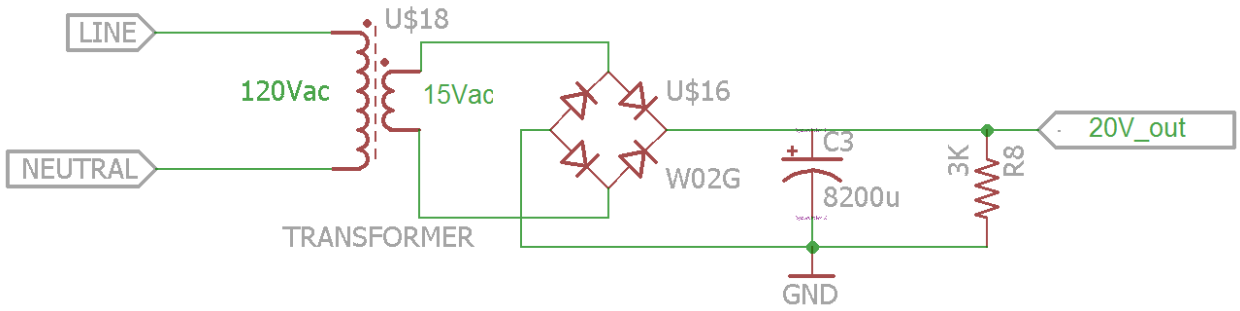

Figure 2: Stepdown transformer and rectifier circuit.

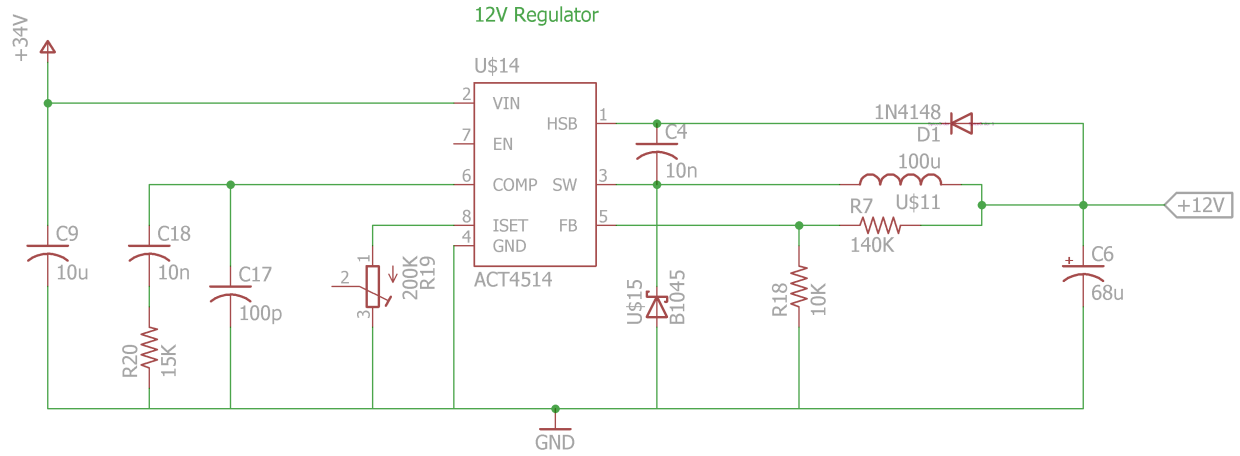

Figure 3: 12 V switching regulator circuit.

used to set the output voltage. This also acts to correct for any changes in the output load that would change the output voltage. Equation 1 is used to calculate the second resistor in the feedback divider. R_{FB2} is recommended to be 10 $K\Omega$. A 140 $k\Omega$ resistor is used as a close value to 138.5 $k\Omega$

$$R_{FB1} = R_{FB2}(\frac{V_{out}}{8.08} - 1) = 10k(\frac{12}{8.08} - 1) = 138.5\ k\Omega[1] \tag{1}$$

The inductor calculation is shown in Equation 2. The calculated value is for the 12 V regulator. The datasheet recommends using $K_{ripple} = 0.3$ and indicates that the switching frequency is 210 kHz. The calculated value of 88.04 μH was substituted with a 100 μH inductor since they are much more common. Capacitor values were chosen as recommended in the datasheet.

$$L = \frac{V_{out} \times (V_{in} - V_{out})}{V_{in} f_{sw} I_{loadmax} K_{ripple}} = \frac{12 \times (34 - 12)}{34 \times 210 \times 10^3 \times 1.4 \times 0.3} = 88.04\ \mu H[1] \tag{2}$$

Due to the nature of surface-mount components, the he ACT45 relies on the ground plane to dissipate any heat it generates. During testing of this circuit it was found that the ground plane was insufficient and the component would enter into thermal protection. As a result output voltage would drop after a few seconds. In order to correct for this problem an alternate regulator arrangement was constructed. A pair of LM317

adjustable linear regulators were chosen to supply the circuit for the final demo. The regulators were set to output 12 V as shown in Figure 4. The resistor R2 was set experimentally to achieve 12 V output.

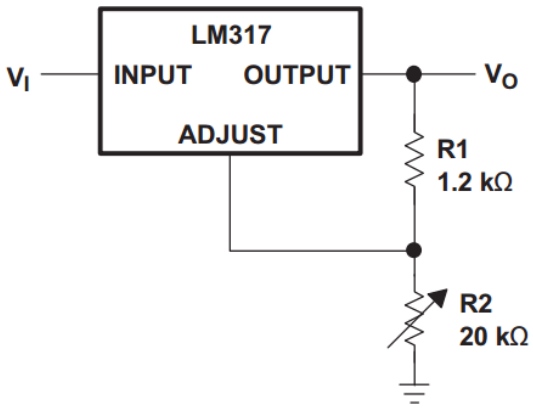

Figure 4: LM317 linear regulator circuit.[2]

2.4 Control Module Unit

2.4.1 Microcontroller

The microcontroller in this project acts as the connecting bridge between sensors and the simulation. An Arduino Uno is used for this purpose. The power source for this unit comes from the voltage regulator and will be operating at 12 V. It takes input from the wheel IR sensor and the handlebar encoder. After probing the sensors the data collected is sent to the Unity simulation using Universal Asynchronus Receiveing/Transmiting (UART) serial connection. In the case of a changing terrain due to the user, the simulation sends a new resistance value to the microcontroller via the UART. In response, the microcontroller then alters the PWM signal sent to the Roller/ Resistance system in order to change the pedalling resistance. Or in the case of the Fan system, altering the triac delays or the DC fan pwm signals to change the speed of the fans.

When designing the microcontroller, the issue of latency was a large concern. Due to the nature of our project the critical path must be kept as short as possible. Based on the documentation given for our microcontroller , the execution time for an analog read from one of our pins is 100 μsec. Since we are polling from the potentiometer on the handlebar as well as the IR sensors attached to the wheel. Thus, reading from these sensors would take a combined 200 μsec. The UART module is sending data at a rate of 20 Hz . After reading from the pins a packet containing the sensor information will be sent to the simulation. The data rate was chosen as midpoint between sending too much data too quickly, overloading the simulation, and sending data to slowly, resulting in a laggy user response.

The data packets were chosen to be a byte in length. The speed of the bike was calculated using input from the IR sensors and is maxed at 30 kph, which is well above the average rider speed. The angle of the handlebars is then mapped to the remaining 325 byte variations and sent to the unity simulation. The data was segregated in this way as to make processing quicker an more efficient on the software side.

2.5 Sensor Module Unit

There are total of three sensors being used in the project in order to track the user's physical movement. Those sensors are the IR sensor tracking the speed of the bike wheel, the rotary encoder tracking the bike handlebar position, and the smartphone accelerometer tracking the user's head movement. The IR sensor and rotary encoder are connected to the microcontroller for communication and the smartphone accelerometer is directly communicating with the simulation program via UART serial communication.

2.5.1 Handlebar rotary Encoder

In order to effectively measure the angle of the handlebars, a 5k potentiometer is placed in the tube of the handlebar. The top section of the potentiometer is attached to the handlebar, and the button section of the potentiometer is attached to the front wheel. The handlebar and the front wheel are being disconnected which allows the handlebar to freely rotate. The potentiometer is configured as a voltage divider where the top and bottom pins are 5 V and ground respectively. This allows us to user the middle pin as a sense pin, where the microcontroller can perform an analog read of the voltage. The rotation action will lead to resistance changes for the system, and the voltage reading from the sense pin will be probed by the microcontroller and mapped to be sent to the simulation.

2.5.2 Phone Accelerometer

The built in accelerometer on the smart phone is used to track the user's head movement. Its input directly feeds into the Unity Program, which is being used to control the simulation field of view. Due to the limitation of the accelerometer, only rotation movement will be support, the linear movement will not be supported in our design.

2.5.3 Wheel IR Sensor

Two IR sensors (IR emitter and IR receiver) are installed at both sides of the rear wheel. A circle of duck tape are being applied around the wheel to block the IR rays with four openings which are evenly spread out.These opening serve as breaks in the sensor readouts probed by the microcontroller. Time diffrential between the breaks is measured and used in the speed calculation. This calculation is trivial since the radius of the wheel is known.

2.6 VR Module Unit

2.6.1 Unity5 Simulation program

The simulation program is built on Unity5 Engine. It contains functions for simulating environment rendering, shading, and texture. The Unity5 simulation is the center of graphical computation for VR simulation. It takes inputs from both smartphone accelerometer and microcontroller via USB connection, and integrates all input signals into simulation control.
The unity program uses the serial port to communicate with the microcontroller. It uses a try-catch loop for any serial port actions, with the reading timeout is set as 1 microsecond. The read in value from serial port is asssigned to speed and handlebar position of simulation bike. The slope of the bike is calculated by the bike orientation angle in y-axis direction and sent to the microcontroller.(See Appendix B)

2.6.2 Unity Remote 4

As the connection between the unity program and the smartphone graphic output, Unity Remote 4 is used to accomplish this task. Unity Remote 4 is being ran on the smartphone, which samples the image from the Unity simulation program and displays it onto the smartphone screen. It serves as a viewing device and does not handle any computation. [3]

2.7 User Interface Unit

2.7.1 Roller/Resistance System

The Rollers/Resistance system is located on the rear wheel of the bike, and gives direct feedback to user from the simulation. The rear dropout of the bike is supported by a bike stand while the roller system is applied to the rear wheel. The roller system is incorporated on the bike stand as a part of the bike trainer. The resistance works by moving a magnet in proximity to a metal roller. The position of this magnet is controlled by a steel cable which is extended using a linear actuator controlled by the microcontroller.

2.7.2 Fan Controller

Fan speed is determined by the rider speed in the program simulation. Speed control for the dc fans is achieved through a buck converter. The dc fan covers low rider speed situations. The buck converter accepts a PWM signal from the microcontroller and outputs a varying voltage to the dc fans. The circuit for this converter is shown in Figure 5. A high side mosfet driver is required for this circuit to operate. Originally this driver was chosen to be the IRS2124 but attempts to build the circuit revealed that this device did not perform the task we required from it. Instead the UCC2701 driver was used.

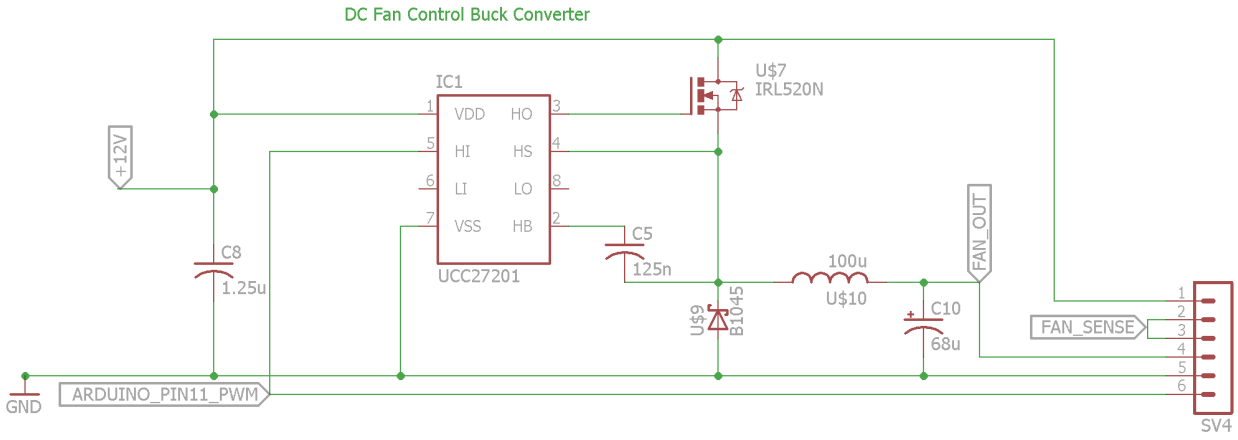

Figure 5: Fan control buck converter circuit.

The equation used to calculate the buck converter inductor value is given in Equation 3. This value was calculated for a minimum output voltage of $V_{out} = 6\ V$ which corresponds to a duty ratio $D = 0.5$. The switching frequency was chosen to be the maximum PWM output frequency of the Arduino Uno. This microcontroller can output at 62.5 kHz in fast PWM mode with 8-bit resolution.[4] The inductor ripple current Δi_L was chosen to be 40% of the maximum current of 1.2 A. Equation 4 shows the result of this calculation. The output capacitor value was similarly determined. The equation for this calculation is show

in Equation 5. The ripple voltage was arbitrarily chosen to be 1% to find an appropriate capacitance. Equation 6 shows the result of this computation.

$$L = \frac{(V_{in} - V_{out}) \times (D \times \frac{1}{f_{sw}})}{\Delta i_L} \tag{3}$$

$$L = \frac{(12 - 6) \times (0.5 \times \frac{1}{62500})}{0.4 \times 1.2} = 102.86\ \mu H \tag{4}$$

$$C = \frac{\Delta i \times (1 - D) \times \frac{1}{f_{sw}}}{\Delta V_{ripple}} \tag{5}$$

$$C = \frac{0.467 \times (1 - 0.5) \times \frac{1}{62500}}{0.06} = 62.2\ \mu F \tag{6}$$

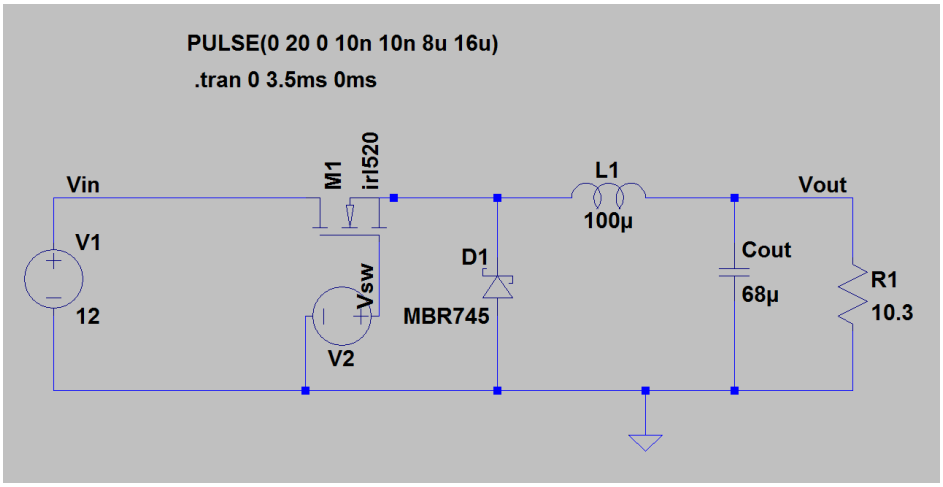

Figure 6: LTSpice buck converter circuit.

The calculated values for the capacitor and inductor were used to choose the values of the inductor and capacitor appearing in the circuit in Figure 5. Since 100 μH inductors are common this was chosen as the inductance value. A very low ESR 68 μF capacitor was chosen for this design. A simulation of these parameters was performed using the LTSpice circuit in Figure 6. Figure 7 plots the output values at various duty ratios showing dc voltages close to expected values.

The ac fan is intended for high rider speed situations and is controlled by a triac circuit. This circuit is shown in Figure 8. The triac blocks the ac voltage until a control signal is provided to the gate. The voltage can then pass through the triac until the signal reaches zero. The gate is activated with a pulse from the microcontroller. The MOC3012 is used to isolate the high voltage ac signal from the microcontroller. The resistor and capacitor values in Figure 8 are derived from the MOC3012 datasheet.[5]

In order for the microcontroller to know when to trigger the triac a zero crossing circuit must be used. The zero crossing circuit is shown in Figure 9. When the ac signal drops below the optocoupler's turn on voltage the diodes no longer conduct and the optotransistor no longer conducts. A brief 5 V signal appears at the

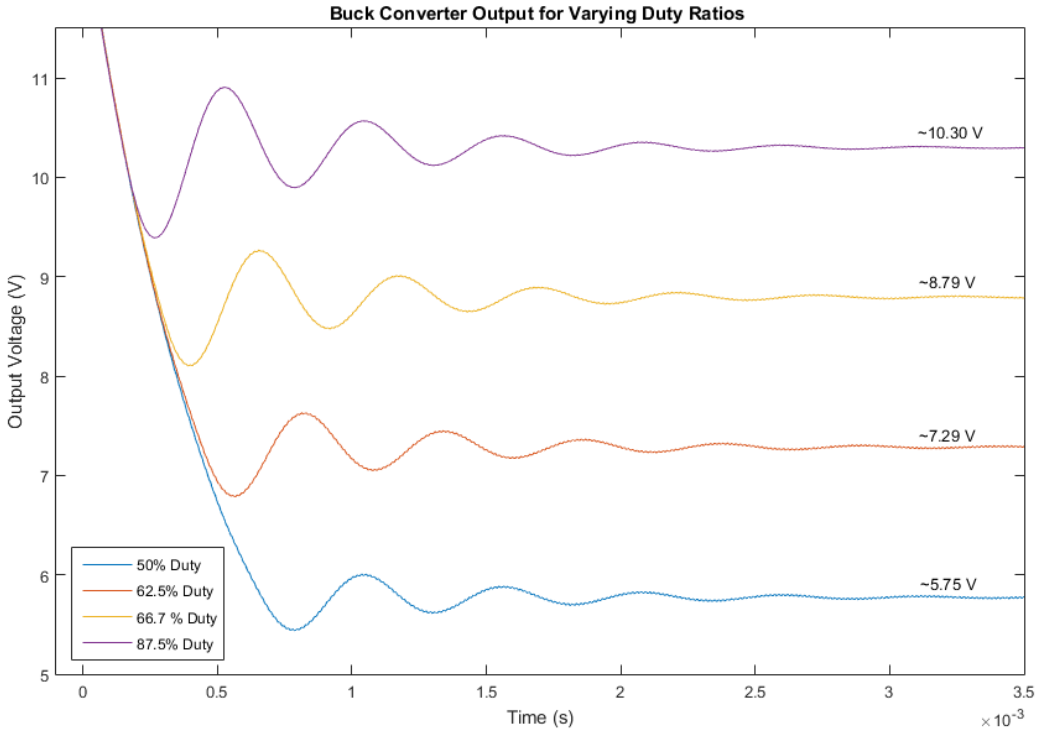

Figure 7: LTSpice simulation results showing expected output voltages.

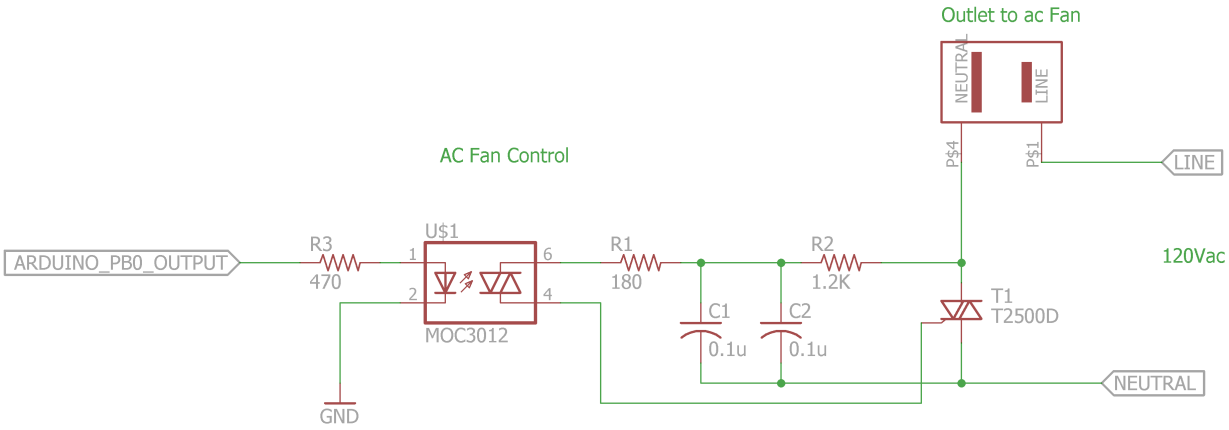

Figure 8: Control circuit for ac fan.

microcontroller input.[6] This behavior confirmed in an LTSpice simulation. The LTSpice circuit is shown in Figure 10.

The results of the LTSpice simulation are shown in Figure 11. The figure shows that at each zero crossing of the input signal there is a brief 5 V peak at the output to the microcontroller to indicate the zero crossing. Once the zero crossing has been located, the microcontroller waits a certain amount of time depending on the desired speed of the fan and then triggers the triac. A shorter time delay gives a faster fan speed.

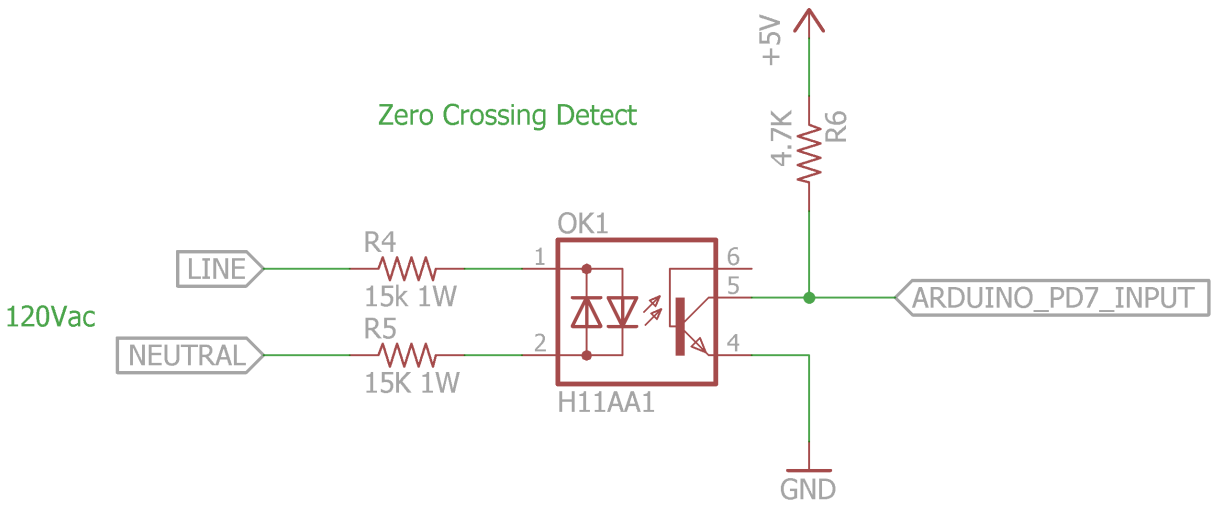

Figure 9: Zero crossing circuit schematic.

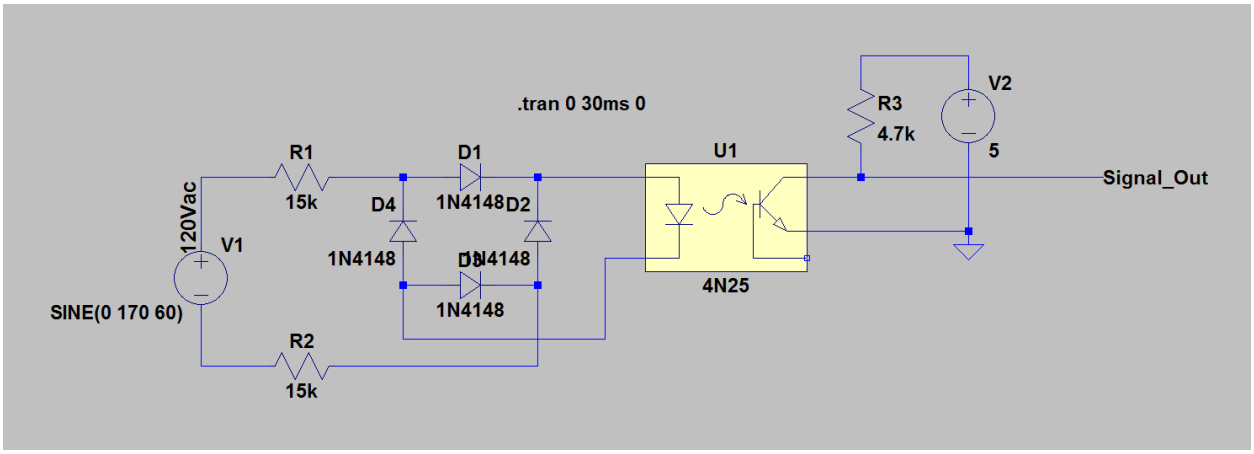

Figure 10: LTSpice Zerocrossing simulation circuit.

2.8 PCB Layout

The design for the PCB focused on incorporating all circuit elements onto the same board. This requirement made it necessary to ensure that there was sufficient isolation between the high voltage sides of the board and the low voltage components on the other. In order to achieve this effect the high voltage components were collected at one side and the remaining were collected on an opposing side. In the region between these two sections, where the optoisolators sit, the copper backing was completely removed in order to make short circuits difficult to occur. The final PCB layout design in shown in Appendix D. The sizing of the ground plane around the voltage regulators was later found to have been insufficient and resulted in excessive heating of the chip. As a result of this excess heating, linear regulators were used instead. The linear regulator circuits were assembled on a separate prototyping board with connections to the existing PCB made with wires connected to open vias.

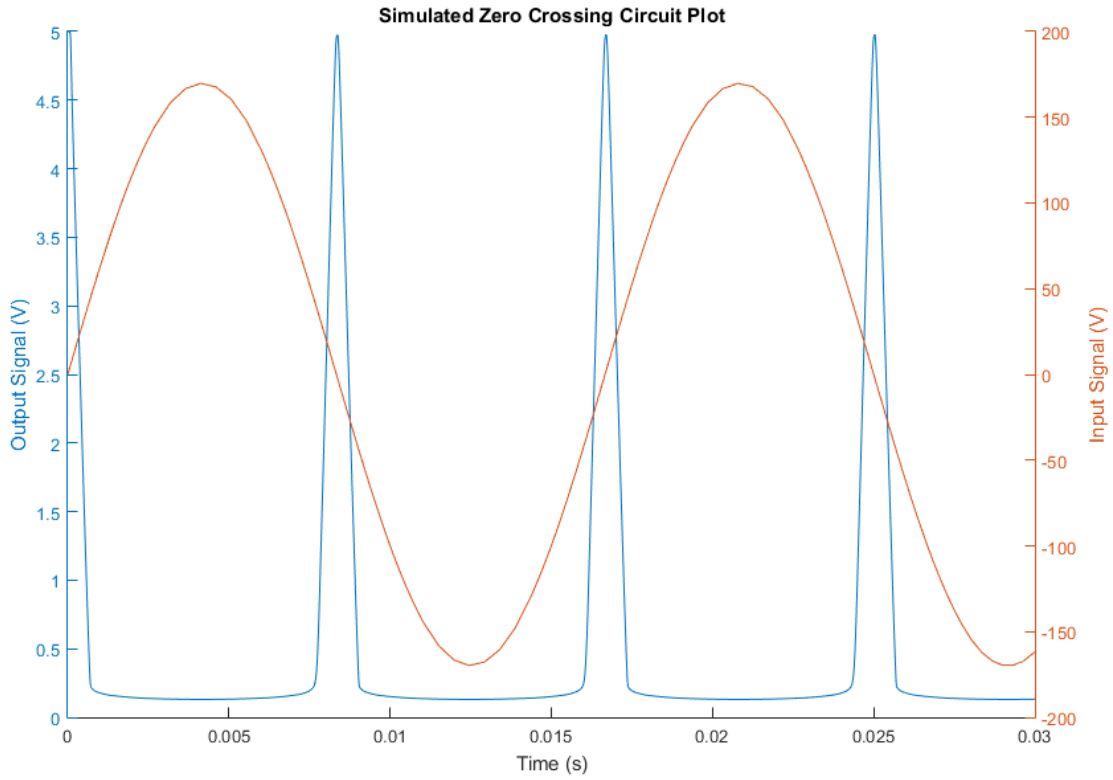

Figure 11: Plot of zero crossing pulses.

2.9 Design Revisions

During the time working on our original design, there were several revision that we made in order to improve the functionality of the final project. These changes were heavily researched and only made in bottleneck situations where our project was hindered by a few select components.

Most notably, in our original design we used Bluetooth Low Energy (BTLE)[7] as a method of communication between the microcontroller and the Unity Simulation. BTLE was chosen due to its quick data transfer speeds, reliability, and wireless nature. However, when running tests on the simulation on the smartphone the frame rate dropped to around 30 fps. This measure was much lower than our design requirement of 60 fps, which is the standard for virtual reality applications. In order to accommodate this shortcoming we moved the graphic processing of the simulation onto a laptop with higher graphical processing power, allowing us to push our simulations up to the required fps. The simulation is now instead streamed to the phone. As a result bluetooth communication with the phone was no longer possible, thus Universal Asynchronous Receiver/Transmitter (UART) Serial communication became the preferable method for data transfer.

In future iterations of this project BTLE communication would be an integral part of the design allowing for more flexibility for the user. Nonetheless, before BLTE can be implemented a more efficient graphical application for the smartphone would have to be developed.

3 Design Verification

3.1 Power

3.1.1 Voltage Regulator

The original switching voltage regulator was constructed and tested for input voltages up to 25 V (the limit of the bench power supplies). In order to be certain that thermal issues were causing the failure of these regulators, the circuit was built on the circuit board and tested as outlined in Appendix A. At 25 V the voltage would drop from 12 V nearly instantly. At 15 V and full verification power the regulator would sit at 12 V for a few seconds before dropping to. The replacement linear regulators were tested exactly as the original regulators were to be tested. These regulators performed well but extra care was required to ensure that they would not produce excess heat since they produce their regulation by converting the excess voltage to heat. Output voltage for the linear regulators was precisely 12 V with no measurable ripple. At 10 Ω output load the measured current was exactly 1.2 A.

The buck converter that supplies power for the fan controller was tested similarly to the voltage regulator. At a 95% duty ratio the voltage ripple measured at 0.448 V which is within specifications. The duty ratios between 95% and 50% were confirmed at intervals of 10%. Across this range the ripple was unchanged and remained at about 0.448 V

3.1.2 Fan Controller

The zero crossing circuit was tested in discrete steps that lead toward full implementation. Since the zero crossing is detected on full line voltage it was important to confirm correct behavior at lower voltages in order to prevent damage to the circuits. This was done by first providing the circuit with a test 10 V_{pp} 60 Hz ac waveform and 5 V from the function generator. This generated expected pulses 1.134 ms long. With the circuit behavior confirmed the full line voltage was provided and the circuit was fully verified as outlined in Appendix A.

Testing on the ac fan was conducted using a test program. The test program accepts an interrupt from the verified zero crossing circuit and records the time using the micros function. The test program then waits a desired amount of time before sending a high pulse to the triac gate pin. To determine appropriate times corresponding to the ac fan speeds this delay time was experimentally varied. The power output to the fan was connected to a power meter and the rms voltage and power were recorded. The results of the chosen delay times are shown in Table 1. From the table the result shows a lower voltage for a longer delay as expected. The 5000 ms time established a lower limit on fan speed while the 3000 ms time established the upper limit. The rms values between these points are within the 15 V difference established in the Requirements and Verification table. In order to create a fourth fan speed a value of 3500 ms was chosen. When this circuit was integrated into the main control code even narrower divisions between the delays were established experimentally in order to eliminating pulsing of the fan speed and create a smooth transition between the different speed levels.

Table 1: Measurements of AV fan voltage and current.

Delay (μs)	V_{rms}	Power (W)	Current (A)
5000	79.86	8	0.21
4000	100.59	17	0.34
3000	115.64	26	0.44

Verification of the dc fan speeds was conduced by providing the buck converter circuit with a signal from the function generator and connecting the fan load to its output. The fan speed sense pin was observed on the scope to determine fan speed. The fan speeds corresponding with the requirements in Appendix A were determined experimentally by varying the duty ratio until the desired speed was found. This resulted in three fan speeds available to the simulation that correspond to low medium and high speeds. These duty ratios were found to be 95%, 75%, and 50% respectively.

3.2 Control

3.2.1 UART Serial Communication

Our requirements for the UART states that the round trip time for a data packet sent from the microcontroller to time simulation and back to be less than 10 ms. In order to test this we loaded a test program onto the microcontroller and sent out a test byte. After this test byte was sent out the microcontroller started a timer to verify our requirement. When the simulation received the byte it immediately sent back and acknowledgement signal. Upon arrival, the micontroller stops the timer and prints this output to the screen. Experimentally we acheived values all below our requirement.

3.3 Sensors Module Unit

3.3.1 Smartphone Accelerometer

Both tracking accuracy and drifting errors were tested and verified for the smartphone accelerometer. For the tracking accuracy test, the simulation ran with the orientation of the "head" GameObject printed out to the unity console. The smartphone was held in the Google Cardboard, with testing orientation being performed in all three axis. The angle of testing orientation was measured with protractor, and the rotation angle was compared with the rotation angle being outputted onto the unity console. The result of the rotation accuracy test was significant. The rotational accuracy was able to achieve an average of $\pm$ 1.5 degrees accuracy.

In order to test and verify the drifting error of smartphone accelerometer that is being used on this project, the simulation program ran with the orientation of the "head" Gameobject being recorded and outputs into analysis file every 10 seconds. The simulation testing ran for 700 seconds continuously, with the smartphone orientation keep constant. The C script that extract the Gameobject orientation during testing. (See Appendix C)

The drifting error test was designed as a continuous 700 seconds static test. The test was performed three times with each time testing the drifting error of a specific axis (x,y, or z axis). The data that produced

during the drifting error tests were collected and analysed. Figure 12 below shown the accelerometer drifting error along the x-axis over a period of 700 seconds.

error.png

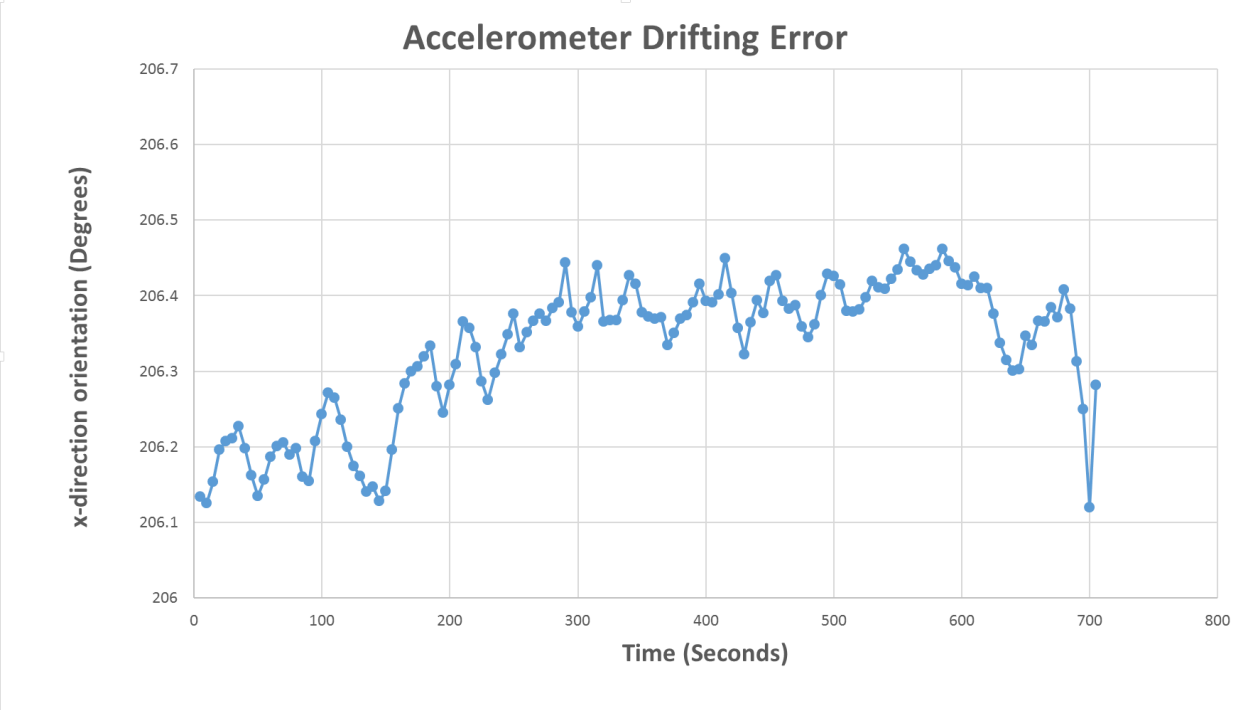

Figure 12: Accelerometer drifting error testing data

As shown in Figure 12, the drifting error in x-axis of the smartphone accelerometer was limited in 0.5 degrees over a period of 700 seconds. This result meets the requirements for smartphone accelerometer for this project.

3.3.2 Handlebar Encoder

The handlebar encoder was tested after the fully installation of the handlebar encoder, and the finishing of the handlebar position code in microcontroller. A protractor was used for the testing of handlebar encoder. The protractor was centered with the rotation center of the handlebar, and it is kept stationary with respect to the front wheel. A wooden stick was taped on the handlebar which points onto the protractor in order to show handlebar rotation degrees. The handlebar position in the simulation was printed out to the unity console. The maximum rotation of the handlebar was tested with the handlebar starting at the neutral position (when the handlebar is orthogonal to the front wheel). It was turned left 120 degrees and turned right 120 degrees, and the reading from unity console output was checked and compared. The resolution and accuracy of handlebar was tested with turning the handlebar left or right with certain testing values, and the value was compared and verified with the unity console output. The handlebar encoder was tested and verified that it satisfies all requirements for this projects. The accuracy and the resolution of the handlebar is able to achieve under 2 degrees, and the maximum rotation of the handlebar achieved more than 120 degrees in both clockwise and counter clockwise direction.

3.3.3 Wheel IR Senor

The IR sensor was tested with the input from a consumer bike speedometer. A consumer version of bike speedometer was installed on the rear wheel of the bike. A testing group member was riding the bike to perform the test. The speed of the bike was printed out to the unity console for compression. To test the minimum speed, the testing member was riding the bike with speedometer shown of 3km/h constantly. The speed reading output from unity console was read and compared. To test the maximum speed, the testing member was riding the bike with speedometer shown of 30km/h constantly. The speed reading output from unity console was read and compared. To test the speed accuracy of the IR sensors, the testing member was riding the bike range from minimum speed to maximum speed. The speedometer readings was constantly being compared with the unity console readings. The performance of the IR sensors satisfied all requirements for this project, which it can measure the minimum speed of 3km/h and maximum speed of 30km/h and has the accuracy of ± 2km/h.

3.4 VR Module Unit

3.4.1 Simulation

The frame rate of the simulation is a critical measurement for the VR experience. In order to test and verify the frame rate performance for the simulation of this project, a specific sections of C scripts were being written in order to calculate the simulation frame rate at current time and display the frame rate information onto the simulation screen. (See Appendix D).
As the testing result, the unity simulation program for this project was assigned with frame rate upper limited of 200fps (frames per second). The real time testing without smartphone graphic output, the frame rate of the simulation was able to achieve a static 150fps ± 20fps. With the smartphone graphic output being integrated, the real time testing frame rate performance of the simulation was able to achieve 80fps ± 20 fps. This performance satisfied the requirements for the simulation program for this project.

3.5 User Interface Unit

3.5.1 Resistance System

The resistance system was verified using experimentally derived positions. Instead of using positions one inch apart as indicated in the Requirements and Verification table in Appendix A the positions were determined based on duty ratio provided to the linear actuator. The linear actuator was able to pull the resistance cable to 5 positions as the requirement table desired but one inch positions were not realistic due to the actual range of movement of the cable.

4 Cost

4.1 Parts

Table 2: Parts Costs

Description	***Manufacturer***	***Part Number***	***Cost***	***Number Ordered***	***Total Cost***
Bike	Dynacraft	n/a	$109.97	1	$109.97
Samsung Galaxy S5	Saumsung	n/a	$467.99	1	$467.99
IR Reflectiv e Sensor	Adafruit	2167	$1.59	2	$3.18
Google Cardboard	Knox Lab	KNOX V2	$15.00	1	$15.00
Arduino Uno	Aruduino	n/a	$24.95	1	$24.95
ATMega328	Amtel	n/a	$3.70	2	$7.40
Bluetooth Adaptor	Adafruit	nRF8001	$19.95	1	$19.95
Bike Trainer	FDW	n/a	$79.99	1	$79.99
Servo	Hiteck	hs311	$7.99	1	$7.99
Fan(4 pack)	Coolermaster	R4-S2S-124K-GP	$12.99	1	$12.99
Rotary Encoder	Allied	RV4NAYSD103A	$7.81	1	$7.81
High side driver	Infineon Technologies Americas Corp.	IRS2124STRPBF	$2.31	2	$4.62
Voltage regulator	Active-Semi	ACT4514SH-T	$0.72	5	$3.60
Transformer	White Rodgers	90-T40F3	$11.99	1	$11.99
N Channel power mosfet	Fairchild	FQP30N06L	$0.95	3	$2.85
10 uF capacitor	TDK	FK20X7S1H106K	$1.06	2	$2.12
8200 uF capacitor	Nichicon	LGU1H822MELB	$3.89	2	$7.78
140 kOhms resistor	Yageo	MFR-25FBF52-140K	$0.10	3	$0.30
Optocoupler	Everlight	H11AA1M	$0.55	2	$1.10
Diode	Fairchild	1N4148	$0.10	3	$0.30
68 uF capacitor	Nichicon	RR71C680MDN1	$0.66	5	$3.30
Power entry module	Qualtek	762-18/002	$9.46	1	$9.46
100 uH inductor	Bourns	6100-101K-RC	$0.77	4	$3.08
Total Components:					$807.72

4.2 Labor

Table 3: Labor Costs

Labor	**Cost**
Wage Per Hour * 2.5	$78.75
Hours Per Week (hours)	20
Number of Weeks (Weeks)	16
Number of Workers (Person)	3
Total	$75,600.00

4.3 Grand Total

Table 4: Total Costs

Components	$807.72
Labor Cost	$75,600.00
Grand Total	$76,407.72

5 Conclusion

5.1 Accomplishments

At the end of the project, all requirements and expectations have been achieved. A smooth VR experience was achieved with stable frame rate performance of 80fps. The latency of the user input was minimized within 100ms. All sensors are able to track the physical actions with required precision. Specifically the speed of the bike is able to be tracked within 2km/h accuracy and minimum of 3km/h and maximum of 30km/h. The handlebar rotation is able to be tracked with accuracy of 1.5 degrees, and the accelerometer drifting error is being limited within 0.5 degrees over 700 seconds. In additional, the power unit is able supply power for all modules, and all user feedback units are able to provide reasonable physical feed backs to user in real time.

5.2 Ethical considerations

1. All designs and functionality of this project were designed with the complete physical safety of the consumers in mind. We have, to the best of our ability, reduced the risks of all physical harms.
2. The VR simulation was designed with safety and the user experience in mind. It was tested fully to minimize all possible uncomfortable experiences to the consumers.
3. VR simulation contains user friendly information, the simulation was designed to be suitable for users from variety backgrounds.
4. Due to the nature of VR simulation, people that are under age 12 are not being recommended to use the VR simulation. The maximum time usage of the simulation is recommended remain under 20 minutes per usage.

5.3 Future work

Although we have created a working design, the are various improvements/considerations that should be examined. As mentioned previously, changing the communication method from UART to BTLE is a convent change that makes the users riding experience feel more natural. In addition, using a microcontroller with more timer interrupts would allow for us to perform some of the heavier microcontroller functions in parallel. Some minor improvements to the PCB layout would likely allow us to use the original regulator chips as well. As a stretch goal we considered the possibility of mounting the bike on a stand that would allows the user to lean on the bike in any direction. While this would greatly improve the immersion and realism of the system, safety concerns and difficulties with realizing this design left us to only consider the bike in the fixed stand. With more outside help and additional mechanical design such a system could be designed as a future improvement.

References

[1] "Act4514 datasheet." [Online]. Available: http://www.alldatasheet.com/datasheet-pdf/pdf/430740/ACTIVE-SEMI/ACT4514.html

[2] "Lm317 datasheet." [Online]. Available: http://www.ti.com/lit/ds/symlink/lm317.pdf

[3] "Unity remote 4 application." [Online]. Available: http://docs.unity3d.com/Manual/UnityRemote4.html

[4] "Arduino forum." [Online]. Available: https://forum.arduino.cc/index.php?topic=310753.0

[5] "Moc3012 datasheet." [Online]. Available: http://www.ti.com/lit/ds/symlink/moc3012.pdf

[6] "Zero-crossing detectors circuits and applications." [Online]. Available: http://www.bristolwatch.com/ele2/zero_crossing.htm

[7] "Bluetooth low energy." [Online]. Available: https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics/low-energy

Appendix A Requirement and Verification Table

Table 5: System Requirements and Verifications

Requirement	Verification	Verification status (Y or N)
1. Voltage Regulator (a) 12 Volt regulators supply current up to 1.2 A with maximum ripple of $\pm 0.6\ V$. 36 V maximum input. **(4 pts)** (b) Buck converter supplies load up to 14 W at 95% duty ratio and operates down to 50% duty ratio. Maximum voltage ripple of $\pm 3\%$ **(3 pts)**	1. Verification (a) i. Attach $10\,\Omega$ power resistor as regulator load. ii. Attach oscilloscope probe across load. iii. Supply 36V to regulator from DC supply. iv. Verify voltage remains between 12.6 V and 11.4 V. (b) i. Attach $10\,\Omega$ power resistor at converter output. ii. Attach oscilloscope probe across the load. iii. Supply 12V to converter from DC supply. iv. Provide 50% duty ratio 62.5 kHz from function generator. v. Verify voltage remains between 6.2 V and 5.8 V.	Y
2. Resistance System (a) Driving linear actuator pulls resistance systems into 3 positions with 1 inch variance between each position. **(2 pts)**	2. Verification (a) i. Using microcontroller and write several testing positions difference to the actuator. ii. Measuring the distance of the wire pulled by the actuator, using a ruler, to ensure correct position.	Y

Continued on next page

Table 5 – continued from previous page

Requirement	Verification	Verification status (Y or N)
3. Microcontroller (a) Microcontroller running final demo code able to achieve a sensor probing with period less than 2.5ms. **(3 pts)** (b) UART communication with PC has a round trip time less than 10ms.**(3 pts)**	3. Verification (a) i. Upload verification code to the microcontroller. ii. User operates bike controller while the simulation is running. iii. Measure the time between probes using the microcontrollers timing functions. iv. Print period to the screen. Verify that the period is not greater than 2.5 milliseconds. (b) i. Upload verification code to the microcontroller, and the PC simulation. ii. Mark the time a UART signal is sent to the PC. iii. Have PC send a response message back to the microcontroller, record the time that it returns. iv. Verify that the round trip time is not more than 10ms.	Y
4. Bike (a) Support user with maximum of 120kg body weight. **(0 pts)** (b) Support maximum biking speed of 30km/h. **(3 pts)**	4. Verification (a) i. Measuring out 120kg of weight with a scale. ii. Place the weight directly onto the bike with weight mount. iii. Check the stability of the system. (b) i. Installing bike speedometer in order to measure the wheel speed of the bike. ii. Testing member riding the bike, increasing the wheel speed gradually until reaches 35km/h. iii. Check the stability of the system.	Y

Continued on next page

Table 5 – continued from previous page

Requirement	Verification	Verification status (Y or N)
5. Fan Controller (a) Zero crossing circuit detects all zero crossings and provides pulse width of at least 0.05 ms. **(2 pts)** (b) Ac fan operates at four distinct speeds seperated by 20 Vrms ± 5 Vrms. **(3 pts)** (c) Dc fan operates at three distinct speeds. Maximum speed is at least 1100 RPM and each speed setting is 200 RPM ± 50 RPM from the last setting. **(3 pts)**	5. Verification (a) i. Attach oscilloscope probe across digital output. ii. Provide wall power to circuit. iii. Confirm on oscilloscope that time between pulses are consistent and are about 8 ms apart. iv. Measure width of pulse on oscilloscope and verify that it is at least 0.05ms across. (b) i. Attach power meter probes across. ii. Provide circuit with wall power. iii. Activate microcontroller test routine to test 4 different time delays. iv. Record the rms voltage supplied to the fan for each time delay setting. v. Verify each speed setting provides between 15 and 25 Vrms difference from the last. (c) i. Attach oscilloscope probe across dc fan pulse output setting. ii. Provide 12V to buck converter circuit. iii. Activate microcontroller test routine to test 3 different duty ratios. iv. For each setting record fan sense pin pulses on oscilloscope. v. Measure time between pulses and invert the time and multiply by 60 to get the RPM. vi. Verify that the RPM is at 1100 at maximum output and verify each setting is between 150 and 250 RPM different from the last.	Y

Continued on next page

Table 5 – continued from previous page

Requirement	Verification	Verification status (Y or N)
6. Unity Simulation Program (a) Achieve minimum of 60 frames per second. **(8 pts)**	6. Verification (a) Run Unity Simulation program with testing personal riding the bike. (b) Verify with frame rate displayed on the screen.	Y
7. VR System (a) Achieve 1 hour of continuous usage with fully charged battery. **(2 pts)**	7. Verification (a) Fully charge the test phone. (b) Run Unity Simulation program and start the timer. (c) Record the time when the test phone turns off due to lack of power.	Y
8. Smartphone Accelerometer (a) Detect rotational acceleration with error of 5% in all three axis. (Yaw, Pitch, Row) **(1 pts)** (b) Drifting error within 25 degrees in a period use of 20 minutes. **(2 pts)**	8. Verification (a) i. Running testing program, set the default orientation. ii. Rotate phone 360 degrees of the x-axis, verifying with default orientation. iii. Rotate phone 360 degrees of the y-axis, verifying with default orientation. iv. Rotate phone 360 degrees of the z-axis, verifying with default orientation. (b) i. Running testing program, set the default orientation. ii. Run the testing program for 30 minutes. iii. Return to default orientation, measure angle drifted.	Y

Continued on next page

Table 5 – continued from previous page

Requirement	Verification	Verification status (Y or N)
9. Handlebar Encoder (a) Maximum rotation of 120 degrees in both directions (clockwise and counter clockwise) in reference to neutral handlebar position. **(1 pts)** (b) Angle resolution of 5 degrees. **(2 pts)** (c) Angle accuracy with ± 3 degrees. **(2 pts)**	9. Verification (a) i. Calibrate encoder using neutral handlebar position (Neutral position is when the the handlebar is orthogonal to the frame, and the front and back wheels are in line). ii. Rotate handlebar 120 degrees clockwise. iii. Read voltage across potentiometer using analog read from microcontroller. iv. Quantize voltage reading into angle and verify with protractor. v. Repeat steps c and d turning the handlebar 120 degrees counterclockwise. (b) i. Calibrate encoder using neutral handlebar position ii. Rotate handlebar 5 degrees in both clockwise and counterclockwise direction. iii. Read voltage across potentiometer using analog read from microcontroller. iv. Quantize voltage reading into angle and observe microcontroller readout changes. (c) i. Calibrate encoder using neutral handlebar position. ii. Rotate handlebar 5 degrees in both clockwise and counterclockwise direction. iii. Read voltage across potentiometer using analog read from microcontroller. iv. Quantize voltage reading into angle and verify with protractor, ensure ± 3 degrees accuracy . v. Rotate handlebar 120 degrees in both clockwise and counterclockwise direction. vi. Read voltage across potentiometer using analog read from microcontroller.	Y

Table 5 – continued from previous page

Requirement	Verification	Verification status (Y or N)
10. Wheel IR Sensor (a) Detect wheel speed of the bike with accuracy of ± 2 km/h. **(2 pts)** (b) Minimum speed detection achieve 3 km/h. **(2 pts)** (c) Maximum speed detection achieve 30 km/h. **(2 pts)**	10. Verification (a) i. Install bike speedometer to measure the speed of bike wheel. ii. Testing personal ride the bike, with speedometer reading of 2 km/h. iii. Verifying bike wheel speed reading from microcontroller. iv. Testing personal ride the bike, with speedometer reading of 30 km/h. v. Verifying bike wheel speed reading from microcontroller. (b) i. Install bike speedometer to measure the speed of bike wheel. ii. Testing personal ride the bike, with speedometer reading of 3 km/h. iii. Verifying bike wheel speed reading from microcontroller. (c) i. Install bike speedometer to measure the speed of bike wheel. ii. Testing personal ride the bike, with speedometer reading of 30 km/h. iii. Verfying bike wheel speed reading from microcontroller.	Y

Appendix B Port Communication Character Control Code

The unity simulation contains C scripts that controls all simulation objects. The main functionality are the simulation character control and microcontroller communication. The code that shown below is the main update functions for bike control, which contains both functionality of character control and microcontroller communication.

The Code below contains all variables that being initialized in the beginning of the program

```
using UnityEngine;
using System.Collections;
using System.IO.Ports;

public class BikeController : MonoBehaviour {

    //check every frame for player input, and apply the input to bike movement
    //checking input from BT in the furture.
    //current input from keyboard

    public static float speed; //speed of the bike
    public float turn; //turning speed of thie bike
    private char[] buf = new char[5] ;
    private float slop; //the slop of the bike


    SerialPort sp = new SerialPort("COM1", 9600); //creating the port that communicate with Arduino

    private float translation; //the variable input from arduino

    public static float angle; //the angle of thrust with respect to foward position
    private float port_read; //variable to read from the port
    private float angle_read; //the angle reading value
```

The function below is the starting function, which being ran once at the beginning of the simulation. The starting function initialize all variables that will be used during the entire simulation.

```
/** Start()
  *     The start function being called once when simulation begins
  */
    void Start()
    {
        angle = 0; //angle value initial as 0
        port_read = 0; //port read initial as 0
        translation = 0; //translation starting as 0
        sp.Open(); //open the serial port
        sp.ReadTimeout = 1;
        buf[0] = '\0';
        buf[1] = (char)1;
        buf[2] = (char)2;
        buf[3] = (char)3;
        buf[4] = (char)4;


        Application.targetFrameRate = 200; \\setting maximum frame rate cap to 200
    }
```

The function below Update() which is being called during every frame being performed. The function contains code to communicate with microcontroller using serial port, and setting the input data value to character control. It also contains code for simulation character control, which include linear action and x-direction rotation.

```
/**
  * Update()
  *     The update function being called every frame
  */
    void Update()
    {
        // check communication with Arduino
        sp.DiscardInBuffer();   //discard USB buffer to reduce latency
        sp.DiscardOutBuffer();
        if (sp.IsOpen)
            {
                try
                {
                    port_read = sp.ReadByte(); //reading 1 byte from port
                    if (port_read < 30)
                        speed = port_read ;
                    else
                        angle_read = port_read;

                }
                catch (System.Exception) { }
            }

        //sending slop variable to microcontroller ever second
        if((int)Time.time % 1 == 0)
        {
            slop_translate(); //output slop to USB port
        }



        //calculating turnning angle and forwarding speed
        angle = (angle_read - 152) * 0.7f;
        translation = speed;


        translation *= Time.deltaTime;

        transform.Translate(0, 0, translation); //linear translation
        if (translation != 0)
            transform.Rotate(0, angle*0.1f, 0); //Rotation for turning


    }
```

```
/** OnTriggerEnter(Collider other)
  *   Code that will be executed when the bike is running into other gameobjects
  *  Current function indicate running into coin and catch the coin, increase score
  *  destroy the coin
  */
    void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("coin"))
        {
            other.gameObject.SetActive(false);
            fps_display.score += 10;
        }
    }
/** slop_translate
  *  function to map the bike orientation into code that can be read by microcontroller
  *  in order to set the resistance by linear actuator
  */
    void slop_translate()
    {
        slop = transform.eulerAngles.x;
        if (slop > 5 && slop <= 10) //down hill 1
        {
            sp.Write(buf, 1, 1);
        }
        else if (slop > 10 && slop < 200)
        { //down hill 2
            sp.Write(buf, 0, 1);
        }
        else if (slop < 355 && slop > 345)
        { //up hill 1
            sp.Write(buf, 3, 1);
        }
        else if (slop <= 345 && slop > 200)
        { //up hill 2
            sp.Write(buf, 4, 1);
        }
        else
        {
            sp.Write(buf, 2, 1);
        }
    }
}
```

Appendix C Accelerometer Drifting Testing Code

```
using UnityEngine;
using System.Collections;
using System.IO;
using System.Text;
using System;

public class Testing : MonoBehaviour {
    public GameObject GO;
    private FileStream F;
    private bool check;
    private int cur_time;

// Use this for initialization
void Start () {

       F = new FileStream("E:/VCR/Testing files/output2.csv", FileMode.Create, FileAccess.ReadWrite);
        check = false;
        cur_time = 0;
    }
// Update is called once per frame
void Update () {

        if((int)Time.time % 5 == 0 && Time.time <= 1200 && check == false)
        {
            Debug.Log(transform.eulerAngles.y);
            byte[] toBytes = Encoding.ASCII.GetBytes(Convert.ToString(GO.transform.eulerAngles.y) + ","
            F.Write(toBytes, 0, toBytes.Length);
            check = true;
            cur_time = (int)Time.time + 1;
        }
        if((int)Time.time == cur_time)
        {
            check = false;
        }
        if (Time.time > 1200)
            F.Close();
    }
```

Appendix D Frame Rate Testing Code

```
public class fps_display : MonoBehaviour {
    //frame rate calculation variable
    int m_frameCounter = 0;
    float m_timeCounter = 0.0f;
    float m_lastFramerate = 0.0f;
    public float m_refreshTime = 0.5f;
    public GameObject display; //the gameobject of the display panel
    private TextMesh number;  //the textmesh that being on the display
    private int fps;
    private float time;
    public static int score;

    void start()
    {
        time = 0; score = 0;
    }
    void Update()
    {
        fps = (int)m_lastFramerate; //convert fps to int
        time += Time.deltaTime;

        //display.GetComponent<TextMesh>().text = fps.ToString(); //display onto screen
        display.GetComponent<TextMesh>().text = ("fps " + fps +
                                "         " + "Score : " + score +
                                "         " + "km/h : " + BikeController.speed).ToString();
        framerate(); //calculating the frame rate
    }
    //function to calculate the frame rate
    void framerate()
    {
        //average framerate calculation, in 0.5second
        if (m_timeCounter < m_refreshTime)
        {
            m_timeCounter += Time.deltaTime;
            m_frameCounter++;
        }
        else
        {
            //This code will break if you set your m_refreshTime to 0, which makes no sense.
            m_lastFramerate = (float)m_frameCounter / m_timeCounter;
            m_frameCounter = 0;
            m_timeCounter = 0.0f;
        }
    }
}
```

Appendix E PCB Layout

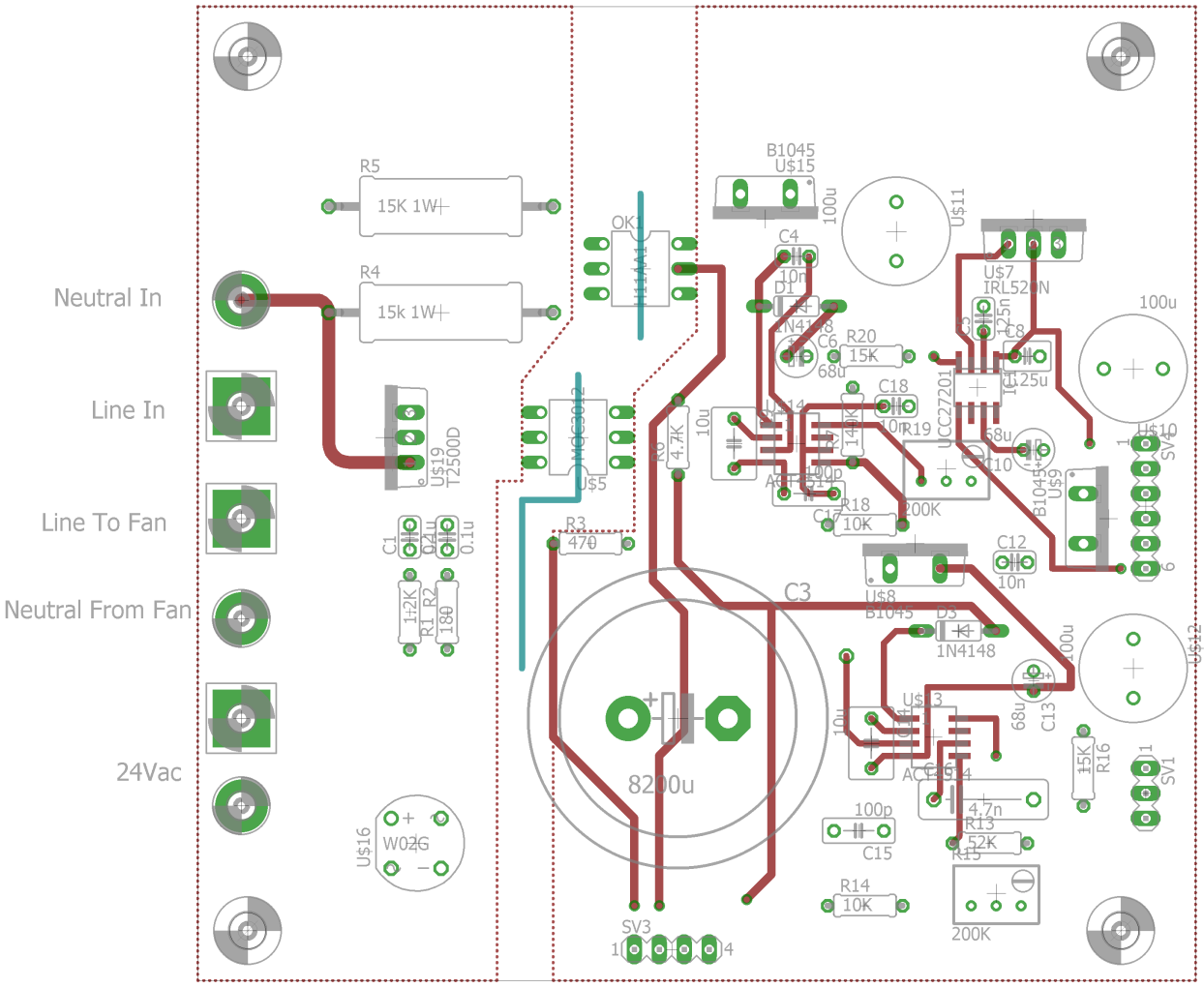

Figure 13: PCB top layer layout.

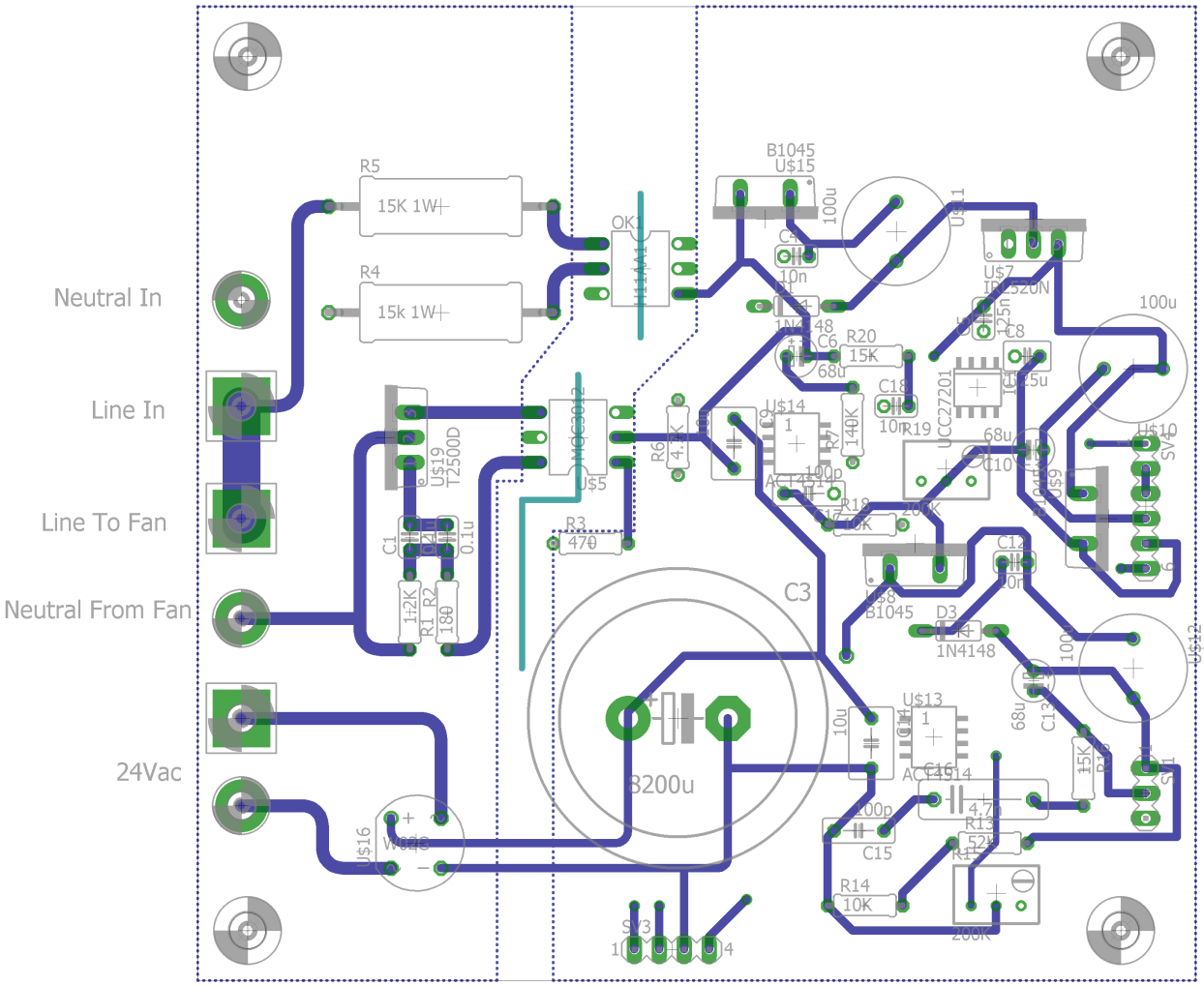

Figure 14: PCB bottom layer layout.