SmartBike

An Automatically Shifting Bicycle with Other Embedded Features

Frank Pernice (CE) and Dylan Dreisch (EE)

Advisor: Professor Kenneth Short

ESE 441 Spring 2014

Design Document

Abstract

The automobile was invented in the late 19th century by Carl Benz (the co-founder of what is today known as Mercedes-Benz). Since its inception, it has undergone numerous innovations. Some of the only similarities between Carl Benz's original prototype and a modern Mercedes Benz are the fact that they both have four wheels and an engine (both of which have also significantly evolved in the last 100 years). Modern automobiles include every comfort imaginable. GPS, climate control, ambient lighting, autonomous driving, premium sound systems, televisions, on-board diagnostics, automatic transmissions, and even cup holders are just a few of the innovations introduced since the original automobile.

Conversely, the bicycle has undergone very little innovation since its introduction nearly 60 years prior to the automobile. While there has been some innovation in this field, the bicycle has remained pretty much the same for nearly 200 years; it still consists of little more than two wheels, a frame, two pedals, and a chain. The SmartBike will be our attempt to incorporate some of the innovations of the automobile into the bicycle. The automatic transmission proved to be a major breakthrough in automotive technology, making cars easier to operate and therefore allowing more people to drive (for better or worse). Our design centers around an automatic set of derailleurs which will shift positions based on a number of factors to be mentioned later. Vitals such as speed, cadence, current gear ratio, and temperature, are displayed on the rider's Android cell phone, which communicates with the bike through Bluetooth. The system is all powered by a battery which is charged via an on-board bottle dynamo. It should be noted that we have no intention of contributing to America's obesity epidemic; the rider will still need to pedal in order to use the bike.

Table of Contents

Goals and Impacts

Daily bicycle use around the world has been on the rise as the cost of transportation continues to increase. Roughly half of American households own at least one bike, 21% own 2 bicycles.[1] In major European countries like Germany, the UK, France, Italy, and Spain new bicycles are outselling new cars. In recession hit countries such as Greece, new bicycles outsell new cars five-fold.[2]

Despite these statistics, bicycle ownership also "declined rapidly [among those] beyond [the] age [of] 55.[1]" Arthritis is a common problem among senior citizens. For those seniors with arthritis in the fingers or wrist, it may be painful for them to shift gears on a multi-speed bicycle. Young children may also struggle with this task due to a lack of strength and immature motor skills. In addition, there are several factors (to be discussed later) that the rider must consider before shifting gears. Plain and simple, many riders want the comfort of a multi-speed bike but are uneducated about these factors, therefore often damaging the bike and discouraging new riders from continuing in the sport of bicycle riding. Like a manual automobile transmission, there is a steep learning curve associated with proper gear shifting on a bicycle. For aerobic exercise enthusiasts, there is much importance associated with maintaining a constant heart rate while riding (which can be achieved, in part, by proper gear ratio across various terrains). By automating this process, we can minimize or eliminate these issues, making the riding experience more pleasurable, safer, and easier. Not only have we made the process of shifting gears easier, but we have also increased the performance of the bike's derailleurs. By replacing the manual wire and pulley shifting mechanism with a servo and pushrod mechanism, we have achieved quick and precise gear changing. Under ideal conditions, our servo of choice can rotate 60^{o} in 170 milliseconds, much faster than a manually controlled derailleur.

The SmartBike is an inexpensive microcontroller based solution to eliminate the need to shift a bicycle automatically. Of course, one is able to override the automated shifting should they choose. In this case, the rider will be taking advantage of the superior electronic derailleurs. Through the use of an Android-powered device (with hands-free mount), the user will be able to view statistics such as current gear ratio, speed, cadence, and temperature. These measurements (albeit temperature) can be used to teach new bicycle riders how to properly shift gears on their own should they one day choose to use a conventional multi-speed bicycle.

In general, multi-speed bicycles offer distinct advantages over their single-speed counterparts. They outperform single-speed bikes at high speed and inclines; however, single-speed bicycles are simpler, have less moving parts, and are therefore more reliable.[3] Single-speed bicycles do not present the risk of cross-chaining which the multi-speed cyclist must be conscious of. Efficient gear changing on a multi-speed bicycle allows the rider to pedal at relatively the same pace for the duration of their ride, thus, maintaining a constant rotation per minute (also known as cadence) on the pedals. When a hill or decreased speed is detected, the SmartBike will shift down to ensure relatively constant cadence. Likewise it will shift up with increased speed. This arrangement will allow for all the benefits of a multi-speed bicycle with the simplicity and ease of riding that comes with a single-speed bike.

Overall the SmartBike could result in a safer, more enjoyable bicycle commute. It could increase the desire for many to bike to work rather than drive. Automated shifting will allow both senior citizens and new bicyclists alike to enjoy their bike ride care free. In today's day and age, energy and fuel prices have skyrocketed and "going green" is all the rage among many. By encouraging more people to ride a bicycle when possible, we can help curb energy costs and help environmentalists "go green." With a bottle dynamo and battery for power, the SmartBike is self-sufficient and clean.

Background

Survey

When we began researching this project, we first stumbled upon the Autobike.[4] The Autobike is relatively new, having just reached the market this past spring. It is the first commercial automatic bicycle. However, rather than use a standard derailleur, it relies on a continuous variable transmission (CVT). Although superior to a conventional derailleur in almost every way, the CVT is much more complicated and prohibitively expensive at around $800. At this particular moment in time, we did not have a bicycle in hand and were weighing our options. A standard derailleur was the only viable option.

Our next option came from Shimano, one of the most respected bicycle equipment manufacturers in the world. They offer an electronic drivetrain package called the Dura Ace 9070 Di2.[6] This package consists of electronic front and rear derailleurs, hand shifters, a battery, and wiring. This design uses a battery that needs recharging before use. It allows for electronic shifting with push-buttons rather than levers or grip-shifters. They use an "automatic trimming front derailleur so you never have to hear the sound of chain rub again.[6]" It offers fast and accurate shifting in all conditions and has a protective outer cage. The cost of this package is $2,499, although it is currently on sale for $2,199. It is a very expensive package for something we had hoped to complete on a $230 budget. Although it would be nice to use this derailleur package and focus solely on the electronics, it is obviously cost prohibitive.

Next we found an IEEE electronic bike derailleur prototype built by David Schneider.[7] His design consisted of servos mounted to the derailleurs, both of which were controlled by an Arduino Uno microcontroller development board. Two pushbuttons per derailleur (four in total) were fed into the Arduino to allow the user to control the servos. This design also used a battery that needed recharging before use. His design is heavily based on an earlier design made by Preston Fall, a bike mechanic from Oregon. His approach was to modify a standard derailleur by removing the spring inside and replacing the bike cable with a servo and pushrod. The included pictures and videos gave great visuals on the mechanics of the system. This seemed like the most viable option and is ultimately how we approached the problem of cost-effective electronic shifting, albeit with our own modifications for our specific derailleur and overall needs. One of the key differences between Schneider's and Fall's designs is that Fall opted to disengage the servos after each shift while Schneider did not. Schneider chose to keep the servos engaged so that the derailleurs would stay in place should they bang against

something during a bike ride. Fall was mainly concerned about power consumption, which is why he opted to rely on friction instead. We opted to mix these two philosophies for reasons described later; our front derailleur is only engaged as needed while the rear derailleur remains engaged for the duration of the ride and is only disengaged when the rider shuts down the bike.

Our research soon brought us to YouTube as we sought a way to better visualize these do-it-yourself (DIY) electronic shifters. One video of particular interest was filmed by YouTube user GroundPepper.[8] He attached the servos to the derailleurs in a manner similar to Schneider's and Fall's designs. The servo is attached to one side of the derailleur while a pushrod is attached to the other side where the tensioning cable used to be attached. The video demonstrates the operation of the device as it steps through the gears at the push of a button. His pushrod design consists of two ball joints: one on the edge of the servo arm and another on the derailleur's pivot point. Of particular interest was the fact that he used ball-bearing joints attached to the servo and derailleur and a rod to connect the two joints. The ball-bearings he used seem to work perfectly, rotating freely as the derailleur is flexed in and out by the servo. Mostly all derailleurs have two screws which can be loosened or tightened to restrict or widen their range of motion. It seems as if he removed these screws and replaced them with the ball joint stud. Since the derailleur is being electronically controlled, the high and low motion limiting screws are not needed (though keeping them is still a good idea to prevent accidental damage). The microcontroller should be calibrated to ensure that the servo stops at the correct gear with no overlap. The comments on the video stated that the spring on the derailleur had to be removed to lessen the load on the servo just as Schneider did.

GroundPepper also has a follow up video[9] where he shows off the electronic controls of the bike. He has both the front and rear derailleurs attached to the bike with servos for electronic control. In the video he switches through the gears on the bicycle and also takes the bike out for a test ride as a proof of concept. His project documentation can be seen on bikeforums.net.[10]

None of these previous designs, with the exception of the Autobike, have a self-generating power source which is another key component of our design. Our research in this area brought us to the Do-It-Yourself (DIY) website Instructables.com. Here we found an article[5] which outlines the steps required to build a generator for a bike. It was intended to act as a USB power supply for the rider's phone or other USB-powered device. The design consists of a small wheel which is mounted perpendicular to and rubs against the rear wheel of the bike. As the cyclist rides, friction between the two wheels causes the small wheel to rotate. The small wheel is attached to a stepper motor, which is used as a power source instead of actually controlling anything. While this provided us with an intuition on how to best generate power, we opted for an off-the-shelf bicycle dynamo typically used in conjunction with bicycle headlights and taillights.

Project Planning

Obviously, this project was very electromechanical in nature; since neither of us are mechanical engineering majors, the mechanical aspect of the project is somewhat daunting. New territories included learning how to properly design, machine, assemble, and mount parts as needed. We took an educated trial and error approach to these mechanical problems. The final design is workable prototype; of course a production model would require significantly more mechanical research and modifications.

Turning electronic signals into physical motion was a key component of our design. The servos had to be precise and powerful enough to ensure smooth and efficient shifting. The servo mounts had to be sturdy and resist the servos' rotation. Without properly mounted servos, the servos can unintentionally rotate their base rather than rotate the push rod. Proper positioning of the servos ensures a straight line between the servo arm and derailleur; their position should allow for no obstacles or extreme angles between it and the other end of the push rod. Of course, in order to do this both derailleurs' ranges of motion had to be studied. To ensure for a proper shifting system, every pivot point and mounting point of the derailleur had to be accounted for.

In order to study how the derailleurs operate, they were removed from the bike. This allowed us to examine their full range of motion and observe all moving parts that needed to be accounted for. First, all forces acting on the devices were examined and accounted for in the servo mount and pushrod design. Second, pushrod design and possible connection points were established. Third, the connections in between the pushrod, derailleur, and the servo were assessed. Finally, the servos had to be mounted in an effective position. Once the physical to electrical part of the project was completed, the rest of the project was highly electrical and software –based in nature.

Of course this project required quite a bit of knowledge in embedded systems design. Prior experience in courses such as ESE 380 and ESE 381 as well as individual research, personal projects, and internship experience in this field were of great value as the project progressed. Since this is a real-time system, it is important that we chose the correct microcontroller for the application. However, even the fastest microcontrollers are limited by the ability of their programmers to optimize code. Frank's internship experience at Texas Instruments revolved around DSP library development and code optimization, which proved to be invaluable in the software portion of the project. Outside of code, this project required that we be possess other general microcontroller and electrical knowledge. Knowledge of built-in and external components such as USART and I^2C, were essential. Knowledge of timer/counters; Bluetooth; Hall-effect sensors; inertial measurement units (IMUs) and others were critical. We were familiar with some of these, thanks to prior course work, but others were completely new. The analytical skills gained during our undergraduate career helped us to learn these most of these new peripherals and interfaces with ease. Due to poor documentation and massive amount of code required, the IMU was mostly dropped from the final design; future improvements to the

system would perhaps incorporate a better or alternative solution as will be discussed later. Nonetheless, communicating with these devices was similar to work done in the past.

System Implementation and Testing

Implementation Problems

From the very beginning, our project was plagued with implementation problems, the first of which were mechanical. In order to ensure proper operation, the servos needed to be mounted in a way which provided full control of the derailleurs; the mounts themselves needed to be strong enough to resist the forces exerted on the servos as they moved the derailleurs. Originally we built these mounts from a piece of aluminum sheet metal mainly because it was pliable using the rudimentary tools we had on hand in the electrical department. However, as we soon found out, the aluminum was too pliable and would not suit our needs. It was then that we fabricated similar mounts out of steel as shown in F**igure 1** below. As you can see, the front mount hugs the bike frame and is therefore very sturdy. However, the rear servo only has one point of contact with the bike frame: a single bolt that is normally used just for connecting the derailleur to the bike. With one point of contact, it was imperative that it be made of a strong metal like steel rather than aluminum.

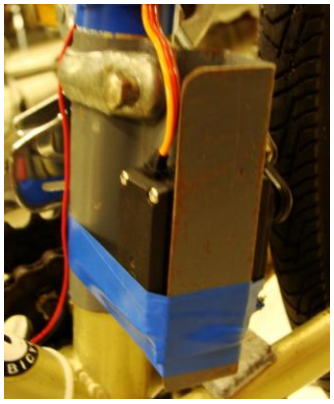

Figure 1a: Front mount

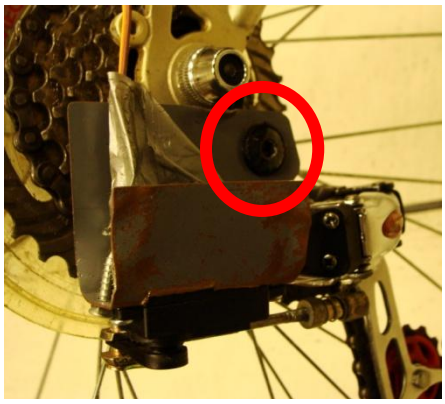

Figure 1b: Rear mount: note single point of contact

Once the servos were mounted to the bike, we of course had to connect them to the derailleurs themselves. Before settling for servos, we considered using worm drives and stepper motors. After deciding on servos, mainly due to speed concerns, we still needed to construct a mechanism to attach them to the derailleurs. As shown in the **Figure 2** below, the design consists of a ball-joint, a clevis rod end, and a threaded rod for both derailleurs. We used existing hardware on the derailleurs to connect them to the ball-joints; the clevises were heated up and then pushed through the plastic servo horns which resulted in custom-fit holes. The threaded push-rod was then cut to size and ties the two together a shown.

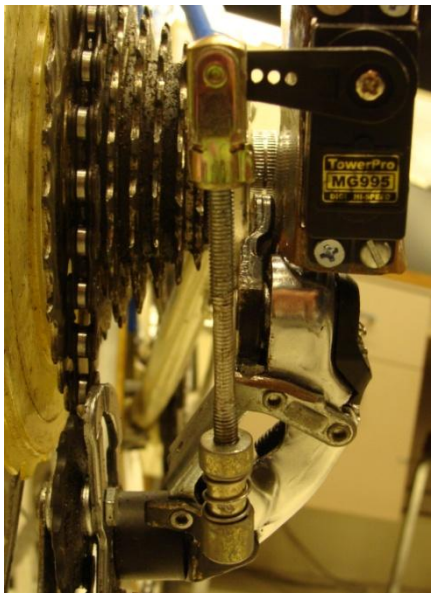

Figure 2a: Rear Pushrod

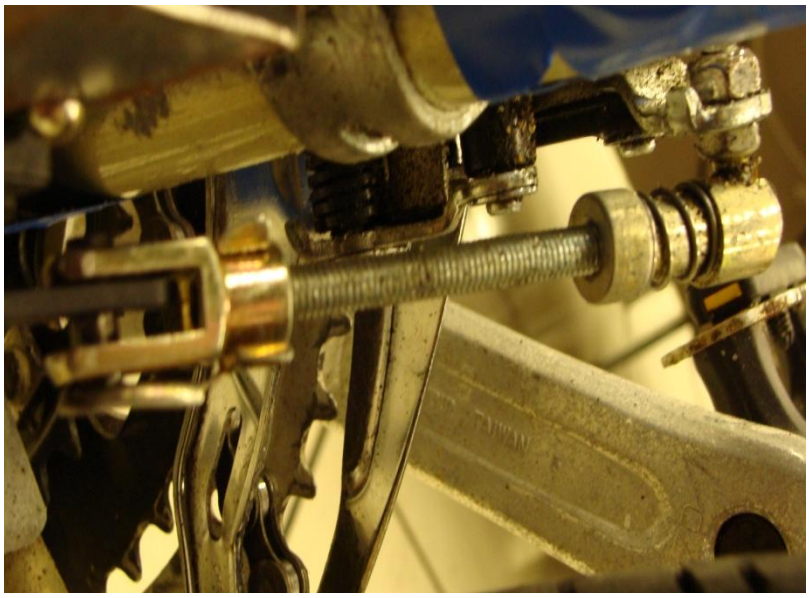

Figure 2b: Front Pushrod

Once these mechanical issues were sorted out, we could finally focus on the electrical and software aspects of the project. Servos are controlled using pulse-width-modulation; we had considered doing this with our microcontroller's timer/counter units as was done in our ESE-381 semester project. However, the ATmega128 only has two 16-bit timers. At the time we thought we needed to use all four timers (two for the servos and two for our Hall-effect sensors) and we were unsure if an 8-bit timer would be adequate for either application. Rather than use all of these resources early on in development, we decided to use an external servo controller to free the timers for other purposes. We opted for the Pololu Micro Maestro (pictured below), a 6-servo controller which communicates over UART.

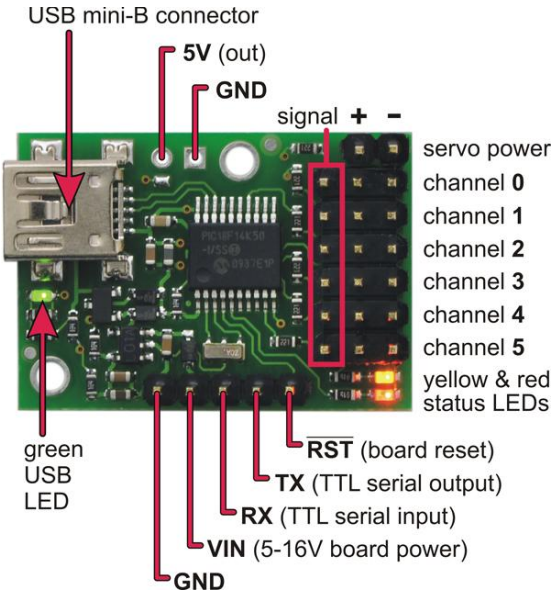

Figure 3: Pololu Micro Maestro Servo Controller (Courtesy of Pololu Robotics)

Since we had no hands-on experience with UART, we had to learn the interface and figure out how to use it on the ATmega128. This, in conjunction with learning I^2C later on, proved to be a valuable learning experience. Essentially, the controller allows us to send a series of commands serially over UART which specify pulse-width and a servo address for any of the (up to 6) attached servos. This was rather convenient as we didn't have to worry about modifying timer settings every time we wanted to move a servo – which could have been time-consuming as we have 42 different positions per servo, many of them with somewhat awkward pulse-widths. The other benefit to using the servo controller is that Pololu provides a Windows application, shown below, which enables users to control and program the device over USB.

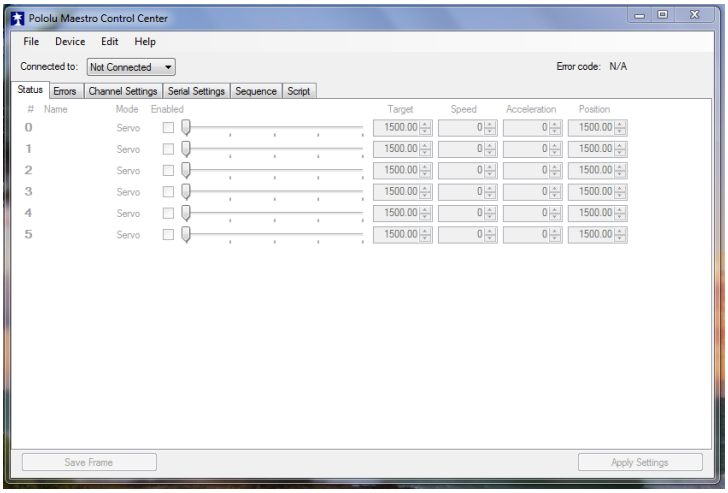

Figure 4: Pololu Maestro Control Center

This allowed us to control the servos in real-time by simply dragging a slider across the screen to obtain the corresponding pulse-width. This was much faster and easier than using a blind guess-and-check method; the values obtained in this software were then used in our program. However, after using these values in our code and sending them to the controller over UART, we noticed that the servo positions were somewhat different. So we were still required to use a guess-and-check procedure to fine-tune the previous pulse-widths, but at least we were able to perform coarse adjustments with the provided software.

Due to budget constraints, we were forced to buy sub-par servos for our project (though we have learned that it might have been better to buy better servos from the beginning). This, in conjunction with the size of the front gears, led to a series of burnt servos. Our first servo spontaneously combusted after a long day of testing; we were unable to cut power before smoke began piping out the sides of the case. Already short on time, we ordered a better servo with 40% more torque. This servo seemed to work fine; it was much more responsive and stronger than the original. Then one morning we set up our laptops in the lab to work and powered up our circuit only to discover that the servo was completely unresponsive. We inspected the servo for internal damage with Tony Olivo's help and found nothing out of the ordinary. We presume it may be an issue with the internal microcontroller. Nonetheless, the

servo was not willing to answer our cries of despair and we were forced yet again to buy another servo.

While we were waiting for both servos to arrive, we dedicated time to other parts of the project, namely our speed and cadence measurement system. This system consists of two digital Hall-effect sensors. Originally, we purchased the Melexis US1881 which, as we soon discovered, is a latching sensor. In other words, after a magnet passes by the sensor, it does not de-latch until an opposite-polarity magnet passes by. We tried to use it anyway, but it was not worth the aggravation. We returned these to Sparkfun and instead bought the Optek OH090U, which acts like a digital reed switch. Another nice feature of this sensor is that it is Schmitt-triggered so that no false-positives are generated at the output while the magnet is still in-phase with the sensor. Nonetheless, we still had problems with the sensor when we mounted it on the bike. Either the sensor had to be placed dangerously close to our magnets or we needed magnets with a stronger field to trigger the sensor. Professor Gouzman was kind enough to donate rare earth magnets which provided the necessary magnetic field. These magnets were Gorilla glued to the front crank and zip-tied to the rear wheel spokes to aid in the computation of cadence and speed respectively.

Once this hardware issue was resolved, we began to write software to calculate speed and cadence with the sensors. Originally, we tied the outputs of the sensors to the input capture pins of the 16-bit timer/counters and used them to count the time between magnet passes. These times were placed into a circular queue and averaged over 8 samples. With these averages we performed some calculation to obtain speed and cadence. However, these computations were slow so we set a flag in our interrupt service routine and moved the computation outside. Nonetheless, this method required that we be conscious of timer overflow so as not to make false computations and also required that we make assumptions about what the cyclist is doing. Instead, we decided to change our algorithm. Rather than compute the time between events we instead decided to count the number of magnet passes every second and then perform computations on these numbers. While this obviously results in a more coarse reading, the resolution was increased with the addition of more magnets such that it is insignificant to both the cyclist and the rest of the code.

In order to give the cyclist conventional manual shifting, we replaced the original grip shifters with four pushbuttons mounted on the handlebars (two for the front gears and two for the rear). For testing purposes we used these buttons without debouncing them (and never had issues with this). However, in order to be safe, we debounced them later on as the project matured. Rather than add more circuitry to the board, we decided to debounce them in software. We modified an algorithm provided by Jack Ganssle[16] for handling multiple inputs. Essentially, an interrupt is triggered every 500ms to poll the button inputs and place the result in a circular queue. When the program needs the state of the buttons, it calls another function which performs a *logical and* on the contents of the queue, essentially “averaging” out any noise. This proved to be essential as we added more functionality to the bike. In order to not overwhelm the user with buttons, multiple buttons can be held down simultaneously to trigger

different operations. For instance, holding down both *down* buttons allows the user to switch between automatic and manual operations, pressing the two *up* buttons instructs the bike to shift into a low gear ratio for hill climbing, and pressing all buttons simultaneously kills the servos, saves the gear positions to EEPROM, and prepares the system to be shut down. Without the button debouncing, these multi-button combinations could possibly be mistaken for a shift and catch the cyclist by surprise. To ensure that this never happens, we went one step further and wait until the user releases all the buttons before acting.

Even in the late stages of development, we were unsure if we would be able to finish our Android app. But providing feedback to the user was still vital. In order to give the rider the best automatic shifting experience possible, we had to alert them when the bike wanted to switch gears. We discussed several ways to do this including placing a small DC motor in the handlebars to produce a vibration or use a buzzer to produce a sound. The motor was ruled out due to extreme current draw relative to its effectiveness. The buzzer was ruled out so as not to annoy or startle the rider. Eventually we settled on mounting a super bright LED on the handlebars near the buttons, but within view of the rider. However, despite using a super bright LED, it is still somewhat difficult to see outdoors. However, since the app is fully functional, this is no longer an issue, the LED is currently used as a secondary form of feedback.

Originally, we had hoped to use an IMU to detect the presence of a hill for automatic shifting. For this we chose the Invensense MPU-6050 which is an advanced, yet inexpensive IMU popular in the hobbyist community. This chip has many features including a gyroscope, accelerometer and thermometer and communicates over I^2C (which we learned for the first time along with UART). Raw data can be read from these sensors on an individual basis and math performed externally on the microcontroller. However, the MPU-6050 has a nice feature which attracted us to it: a digital motion processor (DMP) which is supposed to perform the computations locally and send useable data back upon request, thus freeing the microcontroller for more important tasks. Receiving the raw data was relatively easy. However, using the DMP proved to be a much more difficult task. Invensense provides virtually no documentation on it to the general public. It has been reverse engineered by several hobbyists who graciously developed Arduino libraries for it. Originally we planned on porting these libraries from C++ to C, but this proved to be too daunting of a task. We contacted Invensense to see if we could get more documentation and possibly a library. In order to do so, we had to register on their site with a corporate email address (our Stony Brook email worked fine) to gain access to their developer section. We downloaded the library, which was written for the Texas Instruments MSP-430, and again sat down and tried to port it to work with the ATMega128. However, after looking at the library properties, we saw that it was almost 4Kbytes in size, which would have limited our ability to work from home with the Kickstart version of IAR Embedded Workbench and trying to port it would have been prohibitively time consuming provided the lack of documentation. While reading their code, we noticed several snide, unprofessional comments alongside certain functions. We decided that between the company's lack of documentation, support, and ethics that they were not worth our time. We instead tried to make use of the raw

data instead. However, this required extensive use of trigonometric functions in *math.h*, which slowed down our performance. Additionally, the IMU was too sensitive to vibrational noise and we were unable to obtain useable data even after setting the built-in noise filter to its highest setting. As we continued to race against the clock to finish the project, we decided to drop the IMU from the project almost entirely. After researching alternatives much later into the project we concluded that a barometer or an inclinometer would have been a much better, simpler solution to this problem. Fortunately, Sparkfun has a barometer that communicates over I^2C with a similar footprint. This means that changes to the PCB would not be necessary.

Our power generation system provided several implementation problems. Originally we had planned on using a hub dynamo, a newer technology which is capable of providing more power than an old fashioned bottle dynamo. However, this was not only cost-prohibitive, but would have required that we remove the front wheel, remove the spokes to install the dynamo, and then put new spokes on or cut the old ones to size. So we opted to use a bottle dynamo instead. The bottle dynamo uses spring force to push against the sidewall of the tire and friction is used to rotate a stator in a field of coils, producing an AC signal. This required us to adjust it several times until we found an optimal position. If it became loose it would produce a loud rattling noise rather than produce a good signal. Since the dynamo was intended to power head and tail lamps, some modifications needed to be made to prevent it from interfering with the rest of the system. By design, it originally used the bike chassis as a wire so that only one wire needs to be run between the dynamo and the head and tail lamps. Obviously since we were putting other electronics on the bike, we had to ensure that the AC signal on the chassis would not generate in-system noise. So we isolated the dynamo from the chassis with several layers of electrical tape and connected another lead to it with a ring terminal. Both the negative and positive leads were then fed into our rectifier circuit.

Obviously, since our system is meant to be mobile, leaving our project on a breadboard was simply not an option. We had to design a printed circuit board and have it printed (again, a skill we learned on our own and one which required several days of trial, error, and patience). Initially, we had hoped to design a stackable design similar to the popular Arduino “shields” shown below in **figure 5** which make it easy for users to add components to their design.

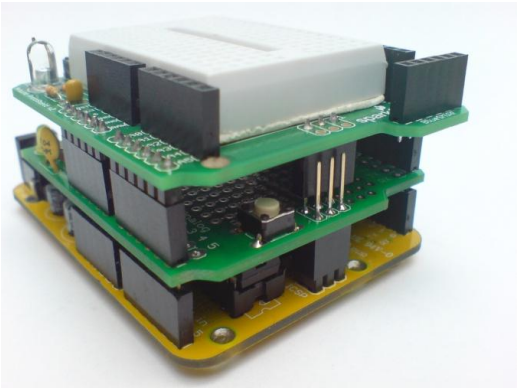

Figure 5: Stackable Arduino shields

Such a design would have been very forgiving to first time PCB designers because adding new components is mostly a matter of adding another stack. It would have also given us room laterally so that we could mount the board inside the frame without the risk of it being kicked by the user. One such board was produced. However, the rapid-prototyping PCB milling machine we used (courtesy of Professor Gouzman) does not line the thru-holes with copper. As a result, vias must be placed on the board before a trace can connect to a component on the solder-side of the board. EAGLE's auto-route tool did not take this into account and therefore the first board was useless. Regardless, the required headers were prohibitively expensive at almost $5 for a measly 16 headers – it would have cost us over $30 for 100 headers. So we opted for a single board design whose traces were routed using a combination of EAGLE's auto-routing tool and hand-adjustments. Its design will be discussed in greater detail below. Once the board was manufactured, populating it became an issue. Because the board is milled from a copper plate rather than etched, it has several drawbacks. Namely, it is not as durable as a board made at a fab-house. We noticed that some traces and pads peeled away due to heat as we began to populate the board. As a result, some traces separated and caused electrical issues which required hours of visual debugging under a magnifying helmet. Ultimately, jumper wires had to be soldered between the disconnected traces to correct these issues. Furthermore, some of the traces and holes were not adequately milled out by the machine. This required about a half-hour's worth of time drilling out holes manually on a drill-press and manually scraping away some copper bridges left between traces. Once the entire board was populated we realized the reset button for the Bluetooth module was placed too close to its antenna and so it would not fit on the board. A breakout board was made which moved the button slightly south. Also, a connection from the Bluetooth module's GPIO7 pin to 3.3V was left out accidentally and so a jumper wire was soldered on afterwards. These two design issues have been corrected in the provided PCB layout.

Originally, we had thought that it would be straightforward to develop an Android application (and it is, provided the developer has ample time to learn). In the interest of time, we decided to use MIT's App Inventor to build the application rather than learn the Android SDK. App Inventor is a web-based drag-and-drop programming environment similar to National Instruments' LabView.

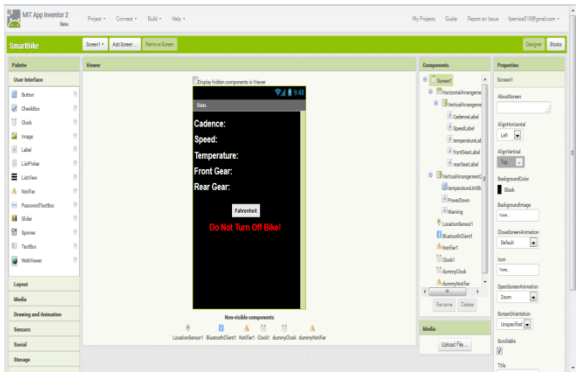

Figure 6a: App Inventor Designer View

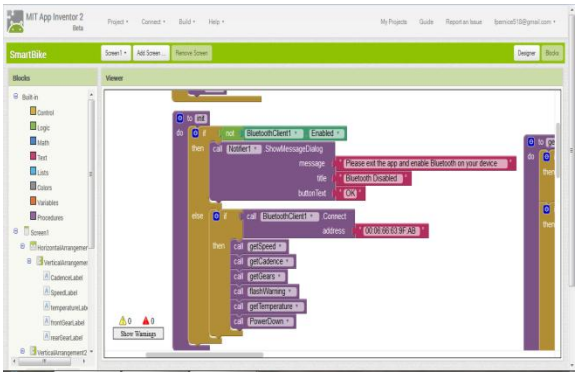

Figure 6b: App Inventor Blocks View

Originally developed by Google, it was placed in the care of MIT when Google abandoned the project several years ago (at this moment in time, it is still under active development, a new beta version was just released for testing in April 2014). While this made rapid development extremely easy, it still had a steep learning curve (especially for an experienced programmer who is not used to coding in this fashion) and a variety of limitations that naturally come with high levels of abstraction. For instance, in an attempt to make concepts easy for new programmers, numerical data types are non-existent. The environment does not distinguish between floating point numbers and integers. This became a problem when we tried to send 4-byte IEEE floating-point numbers to the phone over Bluetooth. Normally one could just typecast the received data and perform the necessary bitwise manipulations to concatenate the bytes of data. Unfortunately, due to the lack of these operations, this had to be done in a complex manner. While App Inventor does support Bluetooth, tutorials in this subject are sorely lacking and so the app contained numerous bugs during the early stages of development. Its Bluetooth features are rather rudimentary; it does not allow the developer to automatically turn on Bluetooth when the app is opened as we would have liked. Instead, the user must do this manually. Finally, there is no way to prevent the screen from dimming and locking after periods of inactivity. As a result, backdoor methods had to be used to provide this essential functionality. In short, App Inventor was very successful at making hard things easy and easy things prohibitively difficult. It is difficult to judge whether or not we would have been better off making an app with the SDK after all.

We had originally planned for the app to be more robust and feature-packed than it currently is. GPS tracking was dropped in the interest of time. In-system calibration and settings modification were dropped due to mechanical constraints. Our servo calibration required extensive hand-tuning and lots of guesswork. It would have been impractical to allow the user to calibrate the system without breaking something. In the bike's current form, this would need to be done in-shop at a dealer.

As we tested the system, we realized that there was a lower and upper limit on cadence in order to properly shift gears. For instance, below 60 RPM, it is very difficult to jump into a new gear (as such we do not allow the user to shift at lower cadences and instead display a warning on the phone when the try to do so). At high cadences the same is true. So generally the user should stay within the range of 60-90 RPM (with 60 RPM being the norm for a casual cyclist and 90 RPM being the norm for an athlete) in order to shift. Further, we decided not to adjust gear positions relative to cadence, but relative to speed. There is a linear relationship between speed and cadence. So based on factors such as tire diameter, we determined the optimal speed for each gear position and developed an algorithm to intelligently select a gear ratio based on speed. Cadence is automatically maintained by linearity, thus still keeping true to our original goal.

Finally, we originally planned to sense pedal pressure to ensure that the bike does not shift when the rider is standing on the pedals (which could break the chain). However, there were numerous mechanical problems with this and so it was dropped. Namely, the pressure

sensor (most likely a piezoelectric pressure sensor) would have needed to connect to the microcontroller. However, in order to prevent wires from becoming tangled in the pedals, some sort of free-rotating connector would have had to be machined. The other option was a wireless connection such as Bluetooth, RF, or Xbee. Of course power would still need to be supplied and so the issue of running wires to the pedals is not completely eliminated. Nonetheless, either solution would have been beyond the scope of the project as the project would have become even more mechanically complex.

Final Implementation

While our final design was different than our original hopes and goals, as is the case with most senior design projects, it was not significantly different. As a whole, the system is fully-functional and incorporates all the necessary features with some extras. Of the few dropped features, none of them were essential to the overall goal, they were merely extras. The final design, from a user's perspective, consists of their android phone, four pushbuttons, an LED, and a power-switch – the simple layout enables the rider to keep their eyes on the road at all times. These feedback peripherals will be discussed in further detail in the following paragraphs.

The four pushbuttons mounted on the handlebars allow the user to control the system. When in manual mode, the buttons on the port side of the bike control the front derailleur while those on the starboard side control the rear as shown in **Figure 7** on the following page. In automatic mode, single button-presses do not do anything as the microcontroller takes full control; only button combinations work in automatic mode. Pressing the two *up* buttons simultaneously (buttons 1 and 2 in the diagram) places the bike a *hill mode*; this rapidly moves the bike into gears 1 in the front and 6 in the rear for optimal hill climbing. This was necessary due to the aforementioned issues with the IMU of choice; we unfortunately have to rely on the user to warn the system of an upcoming hill so that it can adapt accordingly. Pressing buttons 3 and 4 simultaneously toggles between automatic and manual modes and pressing all four buttons simultaneously disengages the servos and saves the current gear positions to EEPROM. In an ideal world of ultra-fast, eternal EEPROM, this would not be necessary because we would save our values after every shift. Unfortunately, this is not the case; we must rely on the user to ensure both the speed of our system and the longevity of the EEPROM.

Digital Hall-effect sensors are used to count the number of quarter rotations of the rear tire and fifths of a rotation of the crank to calculate speed and cadence respectively. Five rare earth magnets are glued to the inside of the crank and four more are zip-tied and taped to the spokes of the rear tire. They are placed equidistant from each other to ensure accurate data collection. The sensors output a logic one normally. As the magnets pass by the sensors, a logic zero appears at the output and an interrupt is triggered to increment the count for the respective sensor. These counts are observed for a 1 second interval, after which the counts are reset to zero and simple math is performed to convert the data into MPH and RPM.

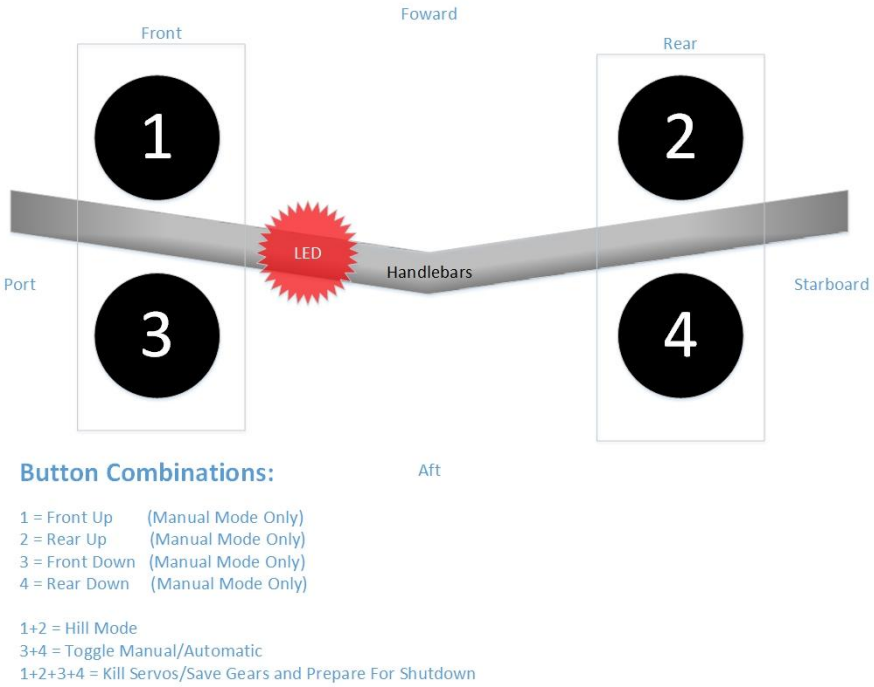

Figure 7: Button Combination Diagram

The super bright LED positioned on the port side of the bicycle, as previously mentioned, is used for secondary feedback. In manual mode, the LED is off by default and lights up while a shift is occurring. In automatic mode, the LED is on by default and will blink just before a shift occurs.

The Android application is used as a primary form of feedback. It shows the current gear positioning, speed, cadence, and temperature values; temperature can be displayed in either degrees Celsius or Fahrenheit by tapping the corresponding button in the app. The app will also warn the user if he is pedaling too slowly to shift (cadence must be greater than or equal to 60 RPM to ensure that the chain can solidly move into position) and also tell the user to not turn off power to the bicycle until they enter shutdown mode. Once the user enters shutdown mode, it will tell the user that it is in standby mode as it disengages the servos and saves data to EEPROM, after which it will alert he user that it is safe to shutdown with the power switch (**Figure 8**) located near the battery.

Figure 8: Power Switch

The app and the bike communicate using the Bluetooth Serial Port Profile (SPP). A Bluetooth profile defines the way a set of devices communicate over Bluetooth. SPP, in particular, is designed to be a wireless RS-232 replacement. The phone acts as a master while the bike acts as the slave. Every second, the phone polls the bike for new data. That is, it sends ASCII character commands to the bike and expects to receive the corresponding data in return. When the command is sent to the bike, the onboard RN-42XV Bluetooth module receives the command and then relays it to the ATmega128. Once the 128's UART data register is filled, an interrupt is triggered which reads the command and consequently sends the requested data to the RN-42XV which in turn forwards it to the phone. This relationship is further illustrated by **Figure 9** below.

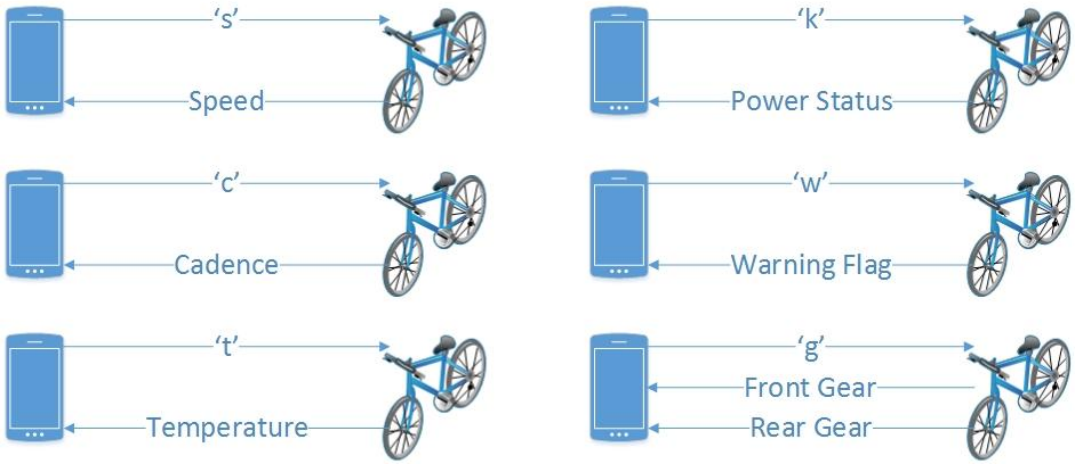

Figure 9: Bluetooth Command Diagram

Because neither the Android SDK nor App Inventor allow us to set the baud rate or parity for Bluetooth communication (they both auto-detect the baud-rate), the Bluetooth module is configured to communicate at 9600 baud to minimize data transmission errors relative to our clock speed of 16MHz.

As previously mentioned, the servos are located near each derailleur. They are both mounted to the bike frame with custom-made steel brackets. A threaded rod, a clevis rod end, and a ball-joint were used to create a set of pushrods (see **Figure 2**) that connect the servos to the derailleurs at their pivot points. When the servo arms rotate, they push or pull the rods and consequently change the derailleur positions. Essentially they convert rotational motion into the linear motion originally provided by the tensioning cables.

Servos are controlled by variations in the duty-cycle of a square wave. The up time of the square wave determines the position of the servo arm. In order to keep the servo in the specified position, this signal needs to be repeated every 20 milliseconds. That is, the frequency of the square wave should be 50Hz. The servos of choice can rotate from 0° to 180° (which corresponds to a duty cycle of 1000 microseconds and 2000 microseconds respectively). The Micro Maestro allows us to control the servos with 0.25 microsecond precision.

Since our goal was to move the chain across the two gear cassettes, each gear had a set of corresponding pulse-widths. Originally, we thought we could get away with using a single look-up table for this. But as we progressed through the gears, we noticed that the servo positions were different depending on the gear ratio and the direction of the servo. For instance, in order to shift from gear 4 to 5 in the rear and then back to gear 4, different servo duty-cycles were required. This is because it is easier to shift down than up, so more force is required to get the servo into a higher gear. Also, if we shifted between gears 2 and 3 in the front, our positions in the rear would change and vice-versa due to the angle of the chain. So what was originally supposed to be one one-dimensional look-up table turned into four two-dimensional tables. Two tables (up and down) are dedicated to each gear cassette. The tables for the front cassette have 7x3 entries while the tables for the rear have 3x7 entries. So the position for the next gear is determined by the gear number in the opposite cassette and the desired direction.

Our servos are connected directly to our 6V battery. Looking back it may have been wiser to use 6V regulators and/or analog servos, but there are benefits to this approach as well. The battery provides roughly 6.2V, but when coupled with the dynamo, the voltage can often spike to almost 7V. Though not done intentionally, this has the added benefit of increasing servo torque. Also if the battery were to discharge and drop below 6.2V the dynamo picks up the slack: it charges the battery and provides the needed power to the system. In order to protect the servos from burnout when under stress, low-side drivers were placed between the servos and ground. During early testing, we noticed that sometimes the chain would not jump into gear (particularly in the front), but the servo would keep on trying until it burnt out. The low-side driver allows us to stop this behavior after a fixed period of time (1.5 seconds). Though the issue of not jumping into gear is now minimal, this feature provides an extra layer of protection. Additionally, because the cogs in the front gear cassette are so large, the servos tend to overshoot when shifting up. Once in gear, the front servo is disengaged and the chain falls into proper position automatically. However, since the rear cogs are much smaller in comparison, the rear servo is always engaged to prevent it from meandering into another gear unintentionally. This is the hybrid between Schneider's and Fall's approaches mentioned earlier. Both low-side drivers are disengaged immediately when the bicycle enters shutdown mode.

As previously mentioned, the IMU proved to be too much of a hassle to be of any use to us. Instead, it is being used as a glorified thermometer. This information has no impact whatsoever on shifting, but it adds a nice additional feature to our system. This information is collected as needed when requested from the app.

Our printed circuit board, shown on the following page, was designed to be as compact as possible. All power components were placed underneath the ATmega128 stamp, which itself was elevated over the components by two rows of female headers. This not only conserved board space, but it allowed us to add the microcontroller to the circuit without soldering it to the board directly. Though we never had a problem with the microcontroller, this would have allowed us to quickly replace it should something go wrong. This was of particular importance

because, as new PCB designers, many problems did arise and nobody knows how many more could have arisen and how damaging they could have been. Alongside the ATmega128 stamp is an additional row of 50 headers on each side. This allows easy access to the pins for debugging and expansion purposes. The other electronic components such as the servo controller, IMU, and the Bluetooth module surround the ATMega128. On the starboard side of the stamp 10 male headers were soldered onto the board for the JTAG connection and on the other side another set of 10 headers for a ribbon cable. The ribbon cable connects to other off-board peripherals such as the buttons and LED on the handlebars and the Hall-effect sensors. This allowed us to attach these components to the board in a neat, compact, easy to debug fashion. Other off-board components such as the servos, battery, and generator are attached in a similar fashion with either three or two pin quick-connects. The PCB itself is mounted to the frame with cable ties opposite the rear gear cassette and out of the way of the rider. Of course a protective case (and more compact circuitry) would be a requirement for a production model so that no components are damaged by road debris. However, in the interest of our (already-over) budget, this was omitted. Of course, the solder-side of the board (not shown) was coated in electrical tape to ensure no solder joints come into contact the bike frame.

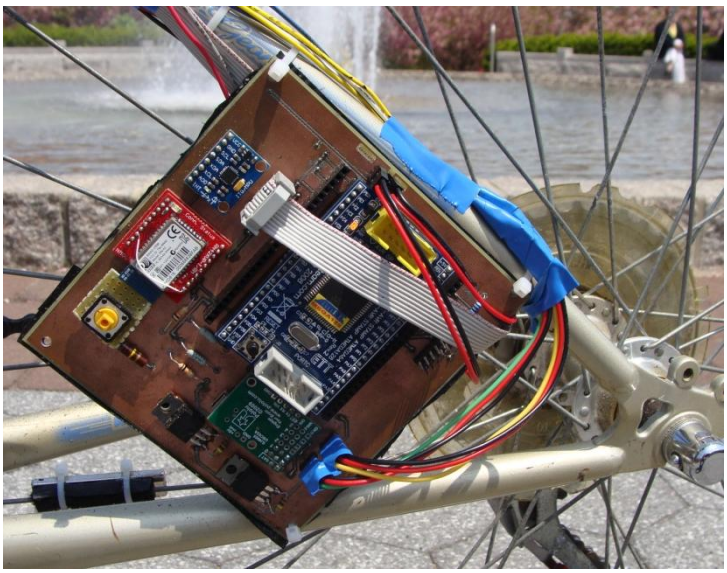

Figure 10: Printed Circuit Board (PCB)

The bottle dynamo is placed under the seat with the rotor placed along the top of the rear tire. The AC signal is then sent through a full-wave bridge rectifier and filtered with a simple RC filer before being sent to the battery, voltage regulators, and servos. Of course the dynamo does not charge the battery at a constant rate; the rate is entirely dependent on how hard the cyclist is pedaling. In general, the dynamo can send about 700mA into the battery. At rest, current draw remains relatively constant at 100mA. However, during gear shifts, the current draw will briefly spike to as much as 2A for a split second or two.

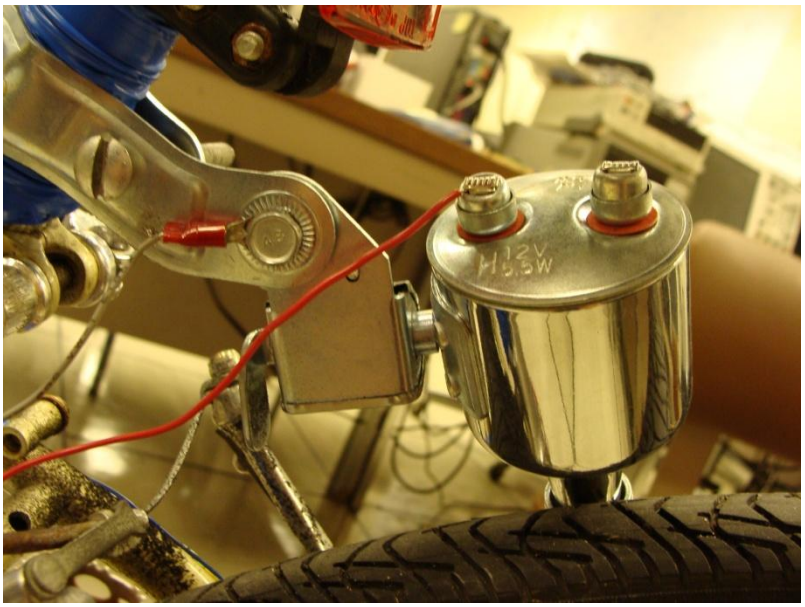

Figure 11: Bottle Dynamo against Rear Tire

Our sealed lead-acid battery (**Figure 12**) is located in what was originally the bicycle's water bottle holder (**warning**: lead acid has not been approved by the FDA as a substitute for electrolyte sports drinks, drink at your own risk). The battery fits snugly in the holder; this position gives us full access to the terminals while preventing it from vibrating or falling off the bike while in motion.

Figure 12: Battery in Water Bottle Holder

The flowcharts on the following pages illustrate some of the main parts of our algorithm.

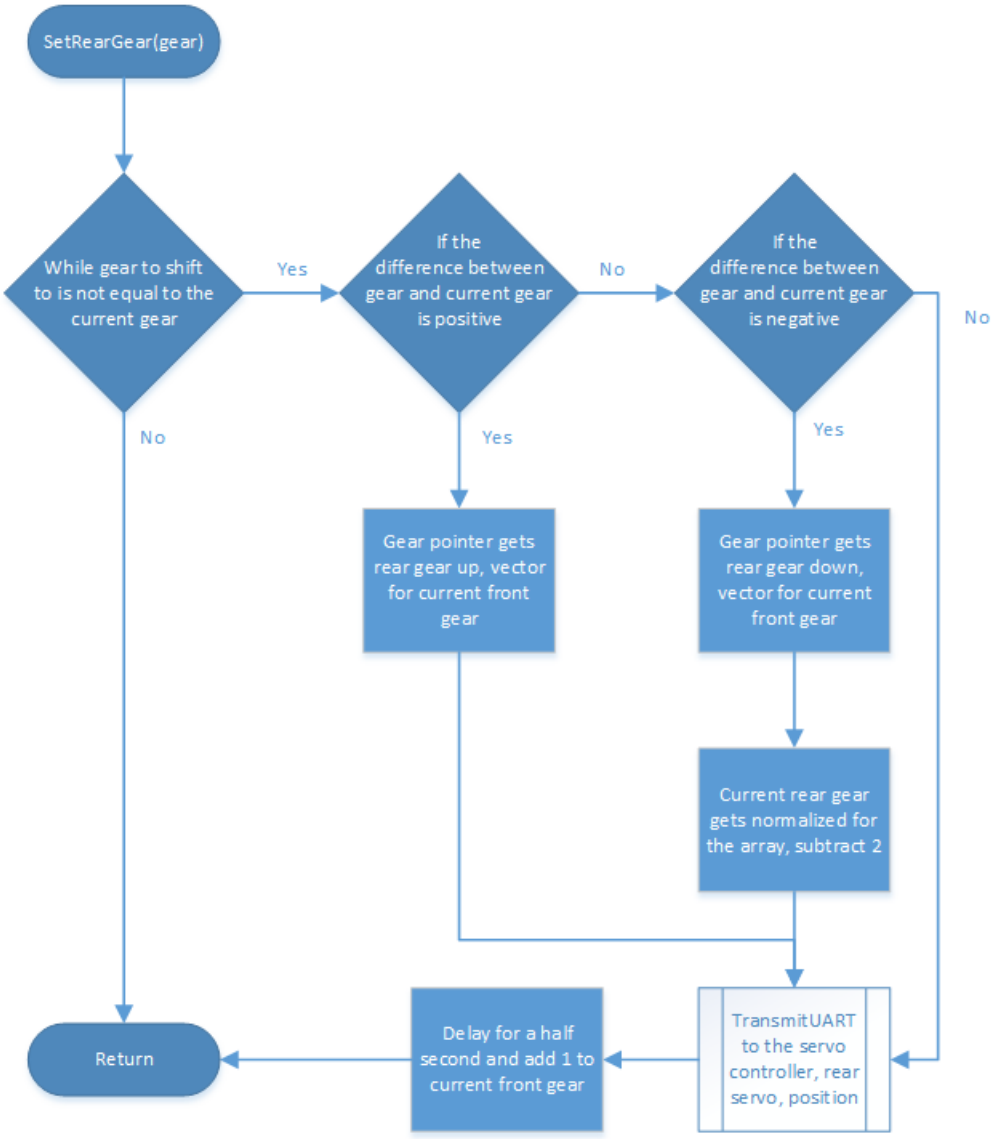

Figure 13: A function which takes in a rear gear value and shifts to that gear one step at a time. The function uses two different arrays, one for up shifting and another for down shifting. We decide our direction, then chose the appropriate array and shift.

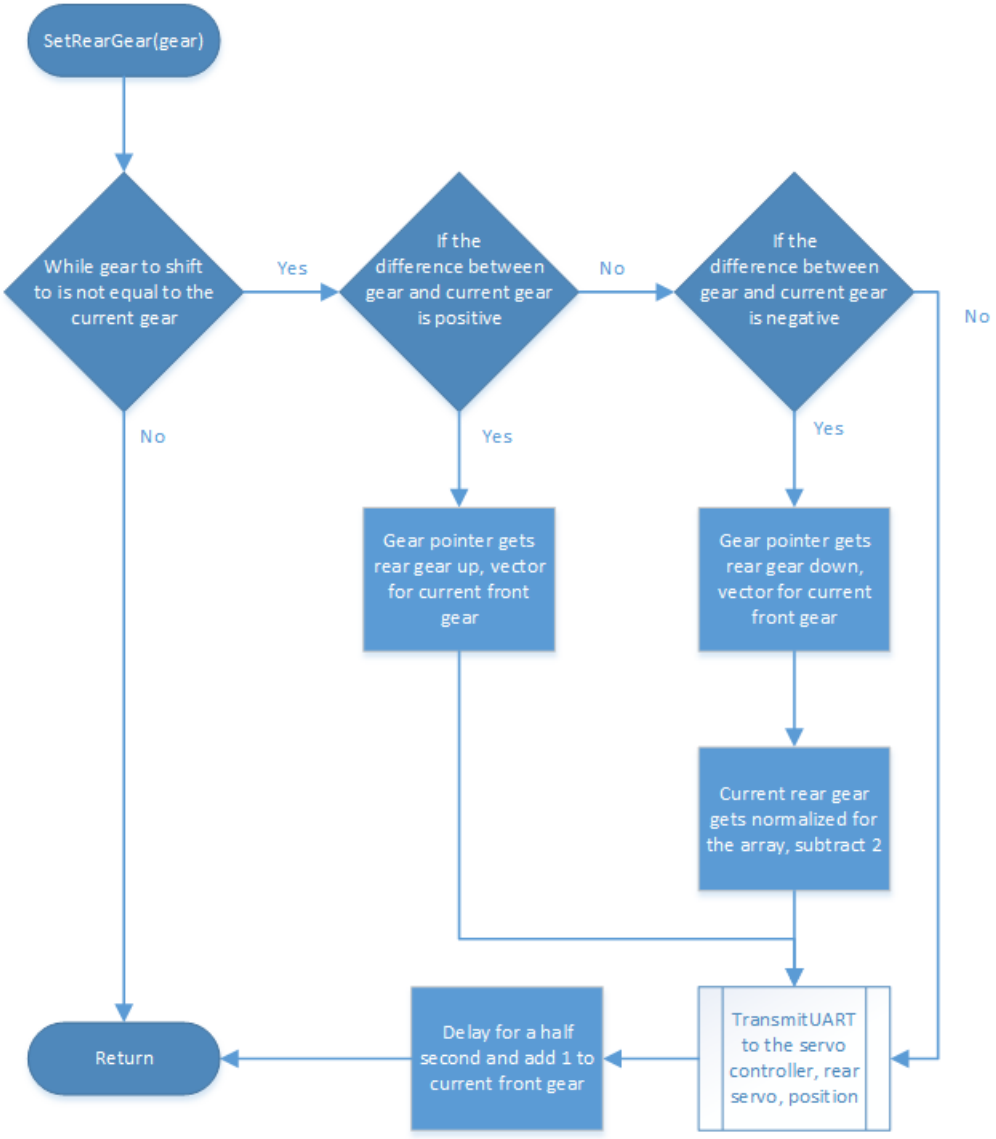

Figure 14: A function that takes the requested front gear value and shifts to it one gear at a time. This function also only engages the front servo for 1.5 seconds, then cuts power once it is in gear to save power along with allowing the derailleur to trim itself instead of staying in a position that produces contact with the chain. Once the front servo shifts, the rear will readjust back into the gear it was in previously. This readjustment was necessary due to the chain jumping gears when the front gear was shifted. It is minimized in rear gear 3 and 5 but still occurs. We use our direction to choose between two different arrays for the servo placement. When readjusting the rear gears, the array direction is opposite of the front array's direction.

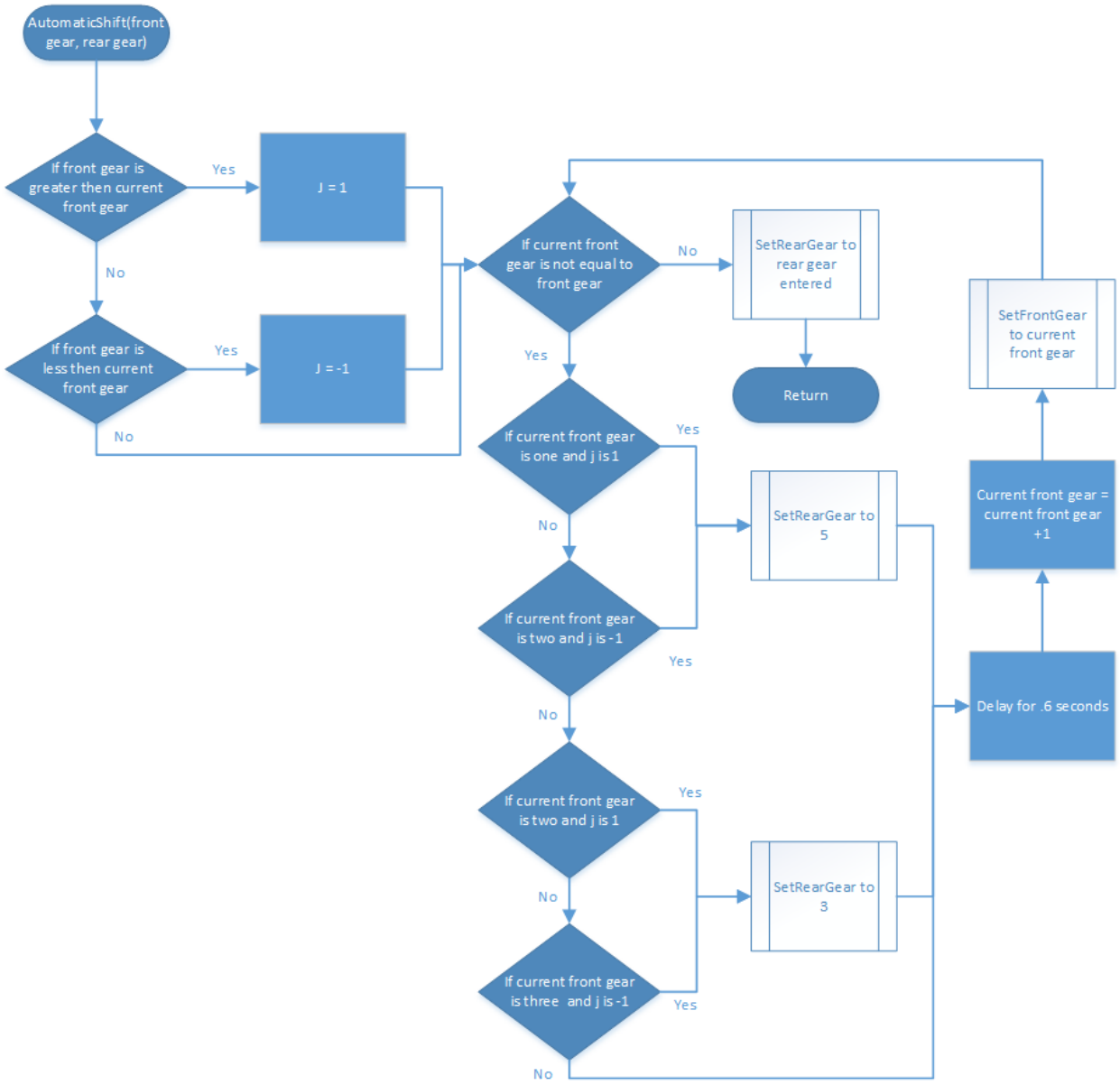

Figure 15: A function used for general shifting in the front and rear. Based off the function used for HillShift, except this is multi-directional and allows for input of gears to shift to. As stated before, this function works the same way as HillShift above, the front gear will only shift once the rear is in either 3 (if shifting between 2 and 3 in the front) or 5 (if shifting in-between 1 and 2 in the front). After the front shift, the rear will make its way one gear at a time to its final destination. In this functions use, the front gear doesn't move more than one gear at a time, but it contains protections in case it were to happen.

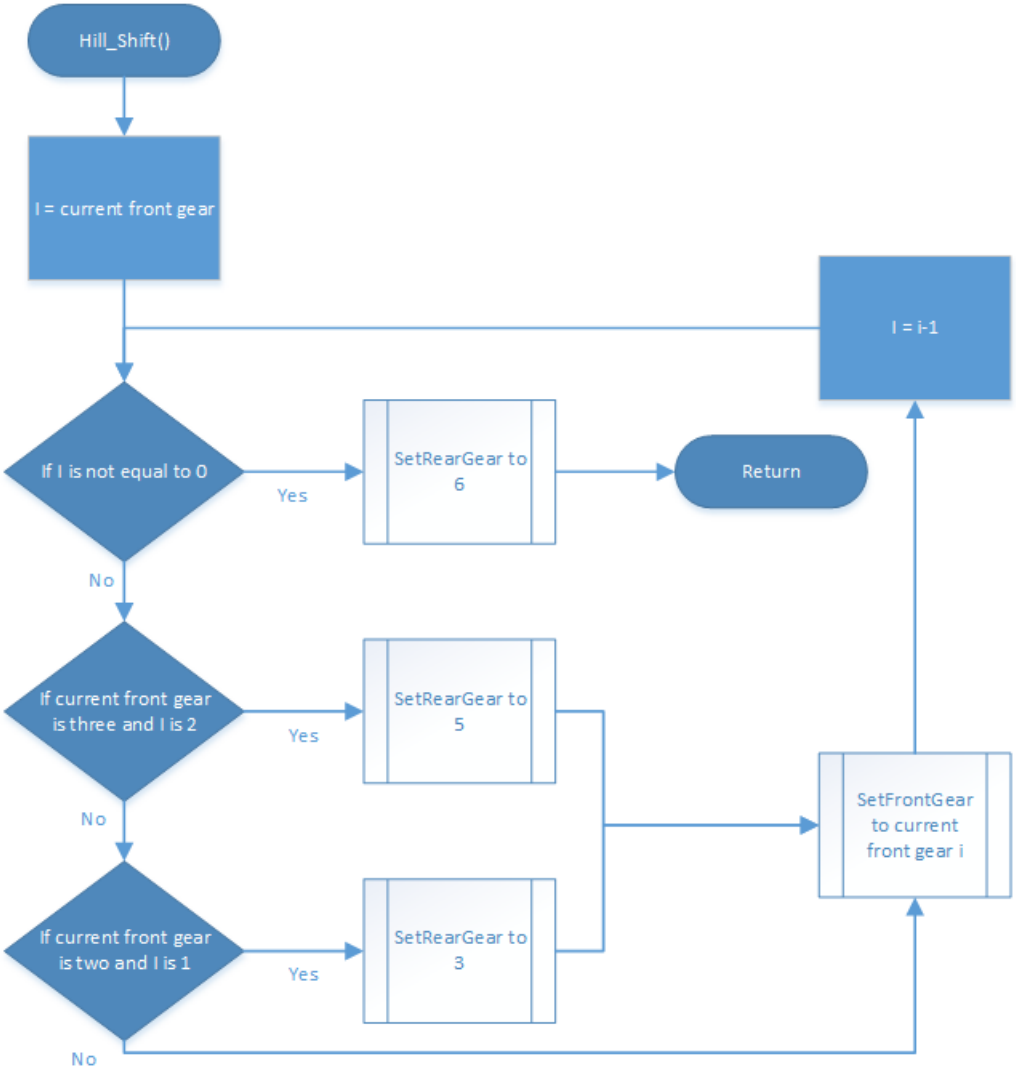

Figure 16: A function used to automatically change the gearing from its current position into gear one in the front and six in the rear. This function is meant to be able to shift into our hill climb gearing from any gear. Also it will only shift the front once our gears in the rear are in the proper transition gears.

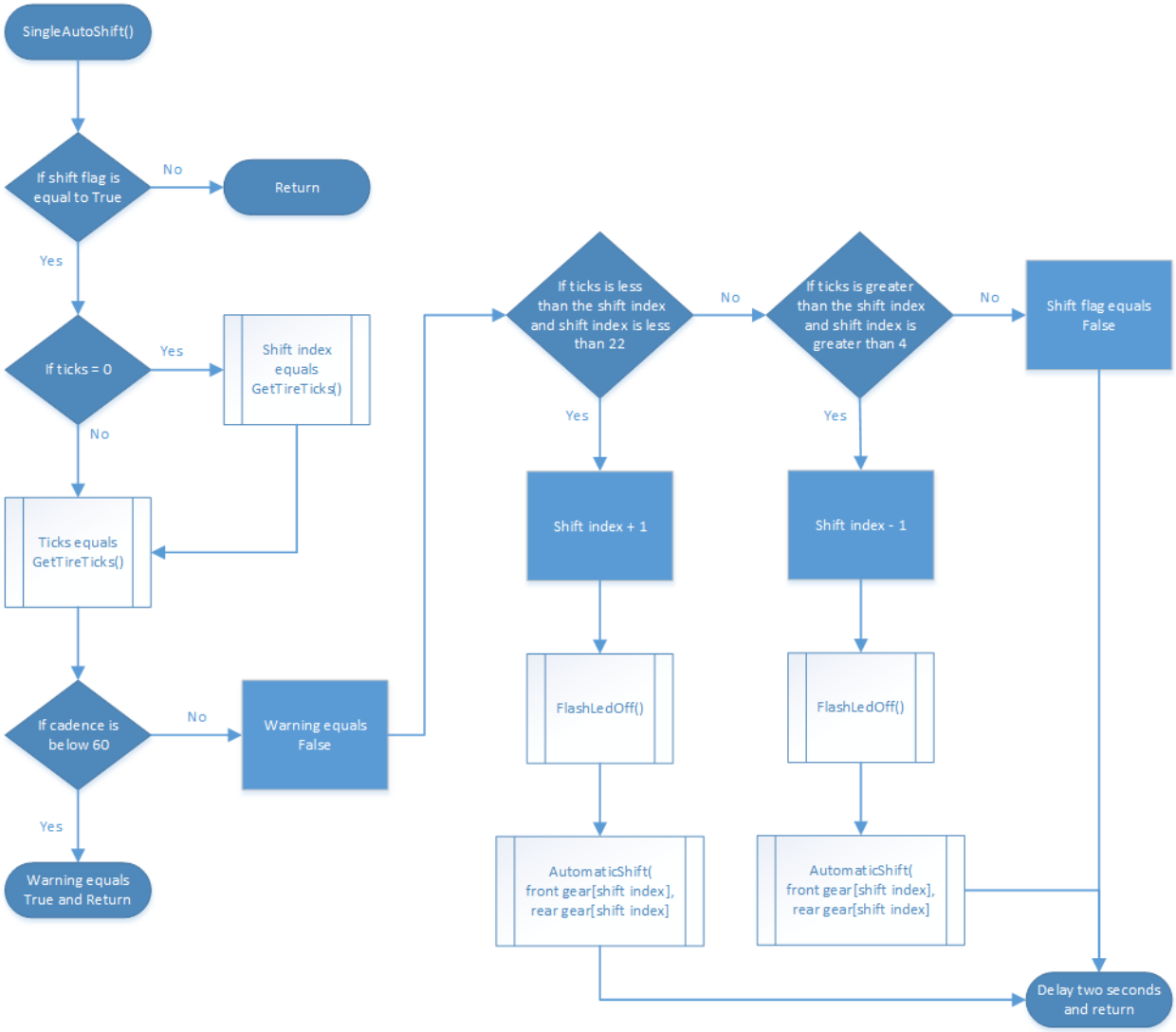

Figure 17: The Autoshift function used to control our gearing in automatic mode. Gets a tick value and compares this to the current Shift index value. If the ticks is larger, then the shift index will increment by one. If the ticks is lower, then the shift index

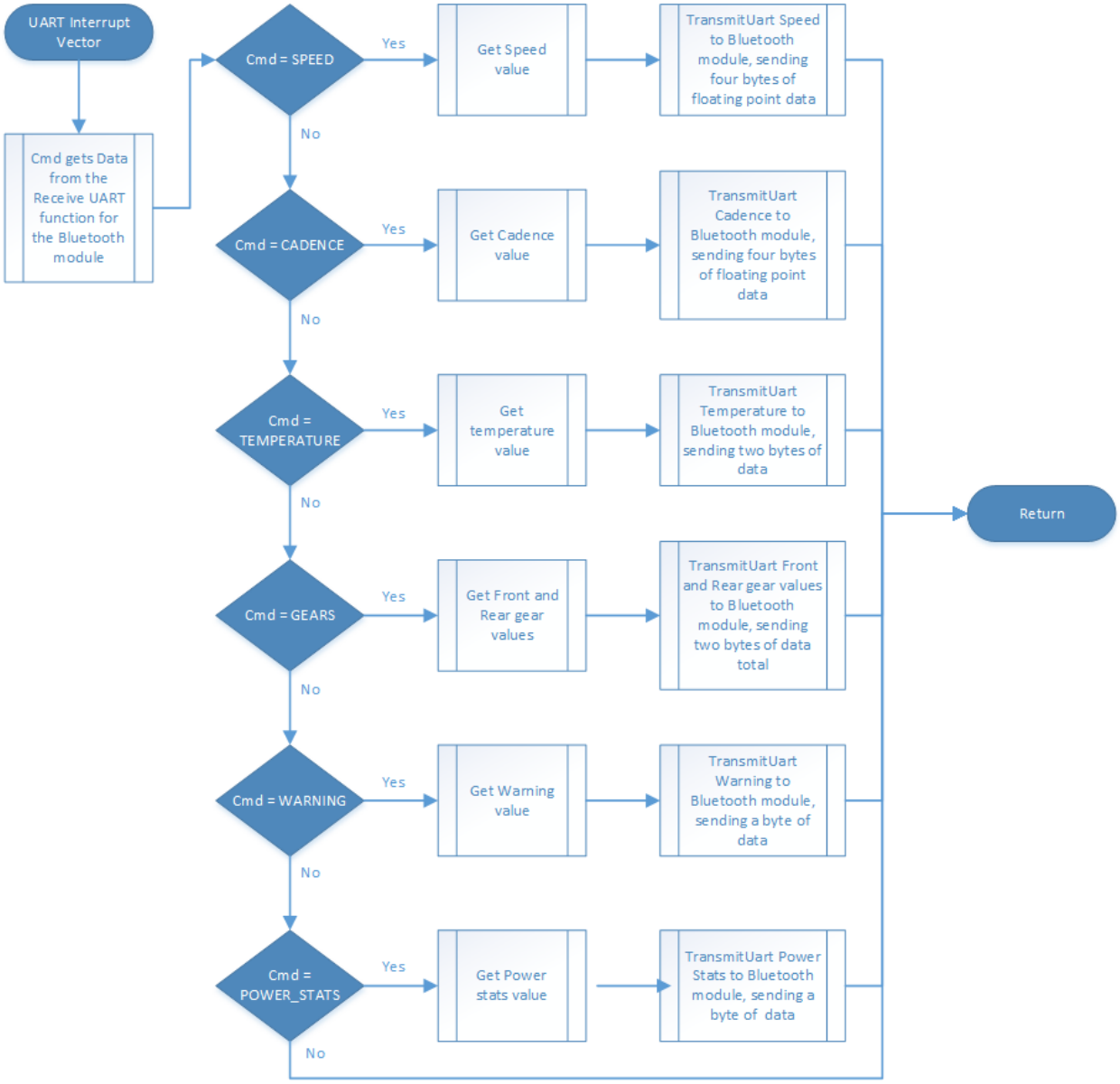

Figure 18: Interrupt service routine used for when Bluetooth sends a command to the ATmega128. The module will send a character relating to our values of SPEED, CADENCE, TEMPERATURE, GEARS, WARNING or POWER_STATS. Once received, the routine will switch to the correct case and send back the data requested.

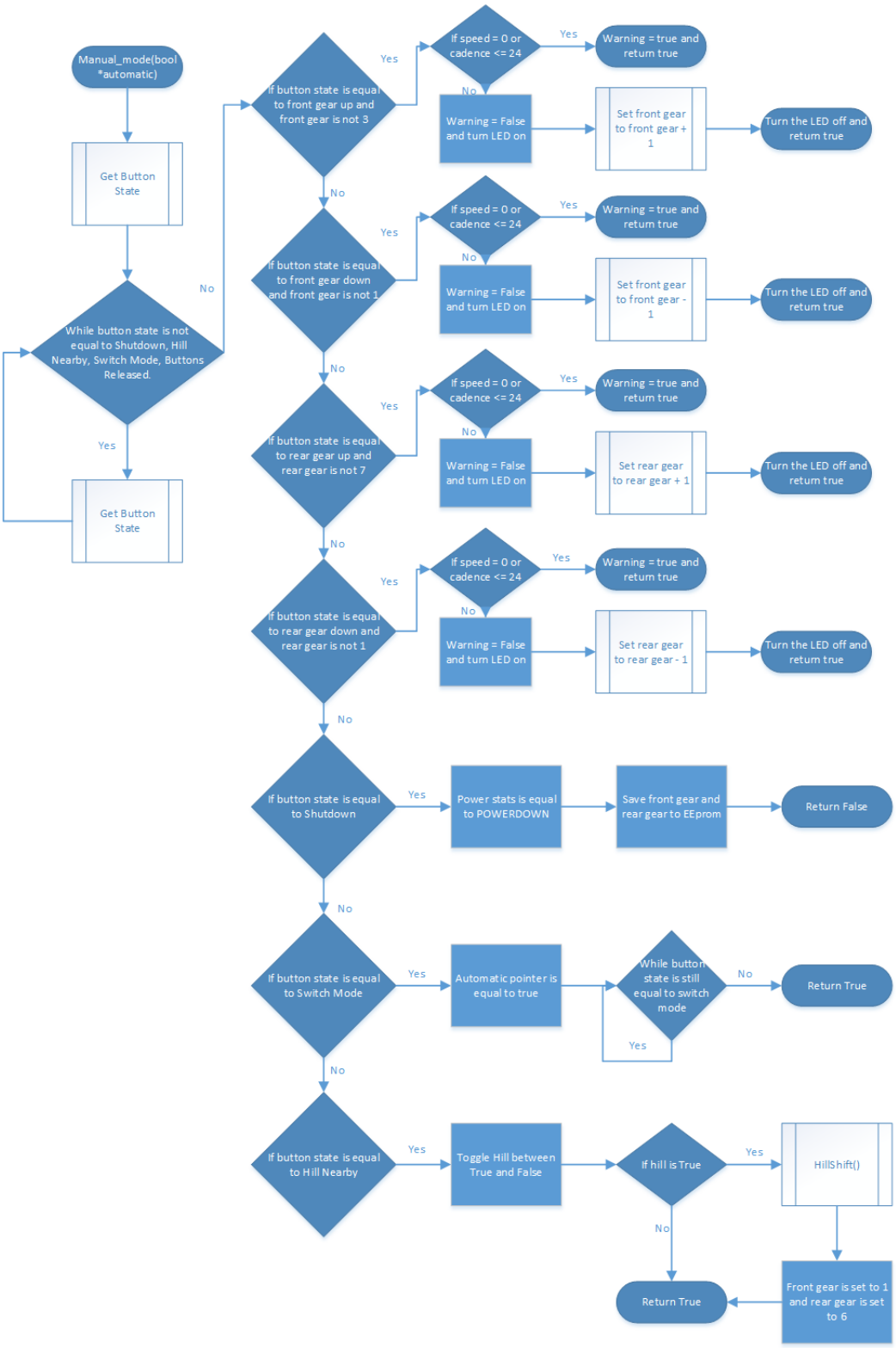

Figure 19: A function for manual mode control. This will wait for a button input and react in one of seven ways: Front Gear Up, Front Gear Down, Rear Gear Up, Rear Gear Down, Shutdown, Switch Mode and Hill Incoming.

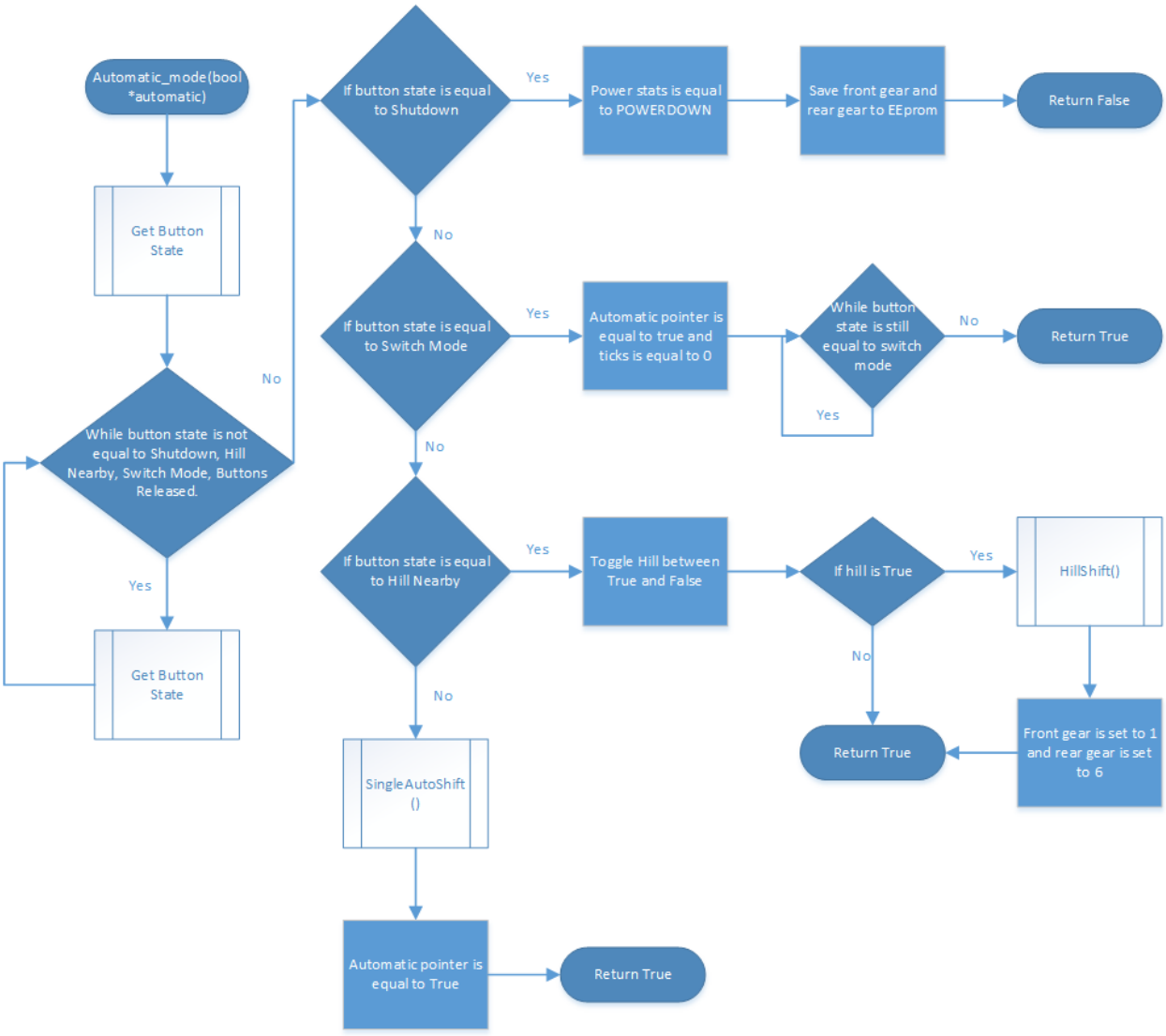

Figure 20: A function for automatic mode control. This will react when a button is input in one of three ways: Shutdown, Switch Mode and Hill Incoming. Once the function determines no buttons were pressed, it runs the SingleAutoShift function to determine which gear the rider should be in.

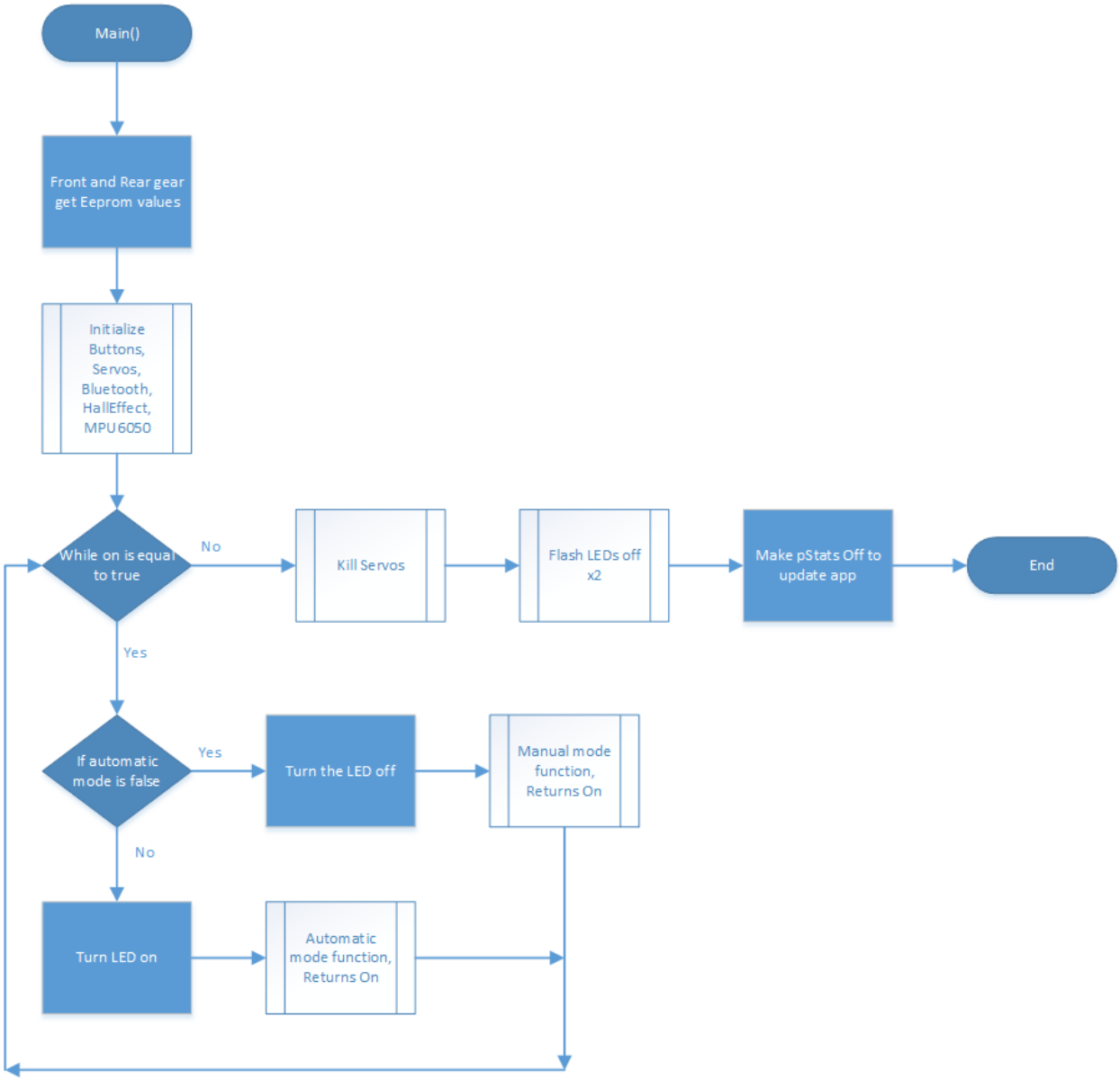

Figure 21: The main function used to initialize our control values like: pStats, on, automatic, front and rear gears. This will also call our initialization functions for each device (buttons, servos, Bluetooth, Hall Effect, MPU6050). The function will then loop while the value of on is true. In this loop, the pointer automatic will decide what mode we are in. Once shutdown is pressed, the servos will be killed and the LEDs will flash to signify shutdown was activated.

The following schematics represent the complete system as it currently stands.

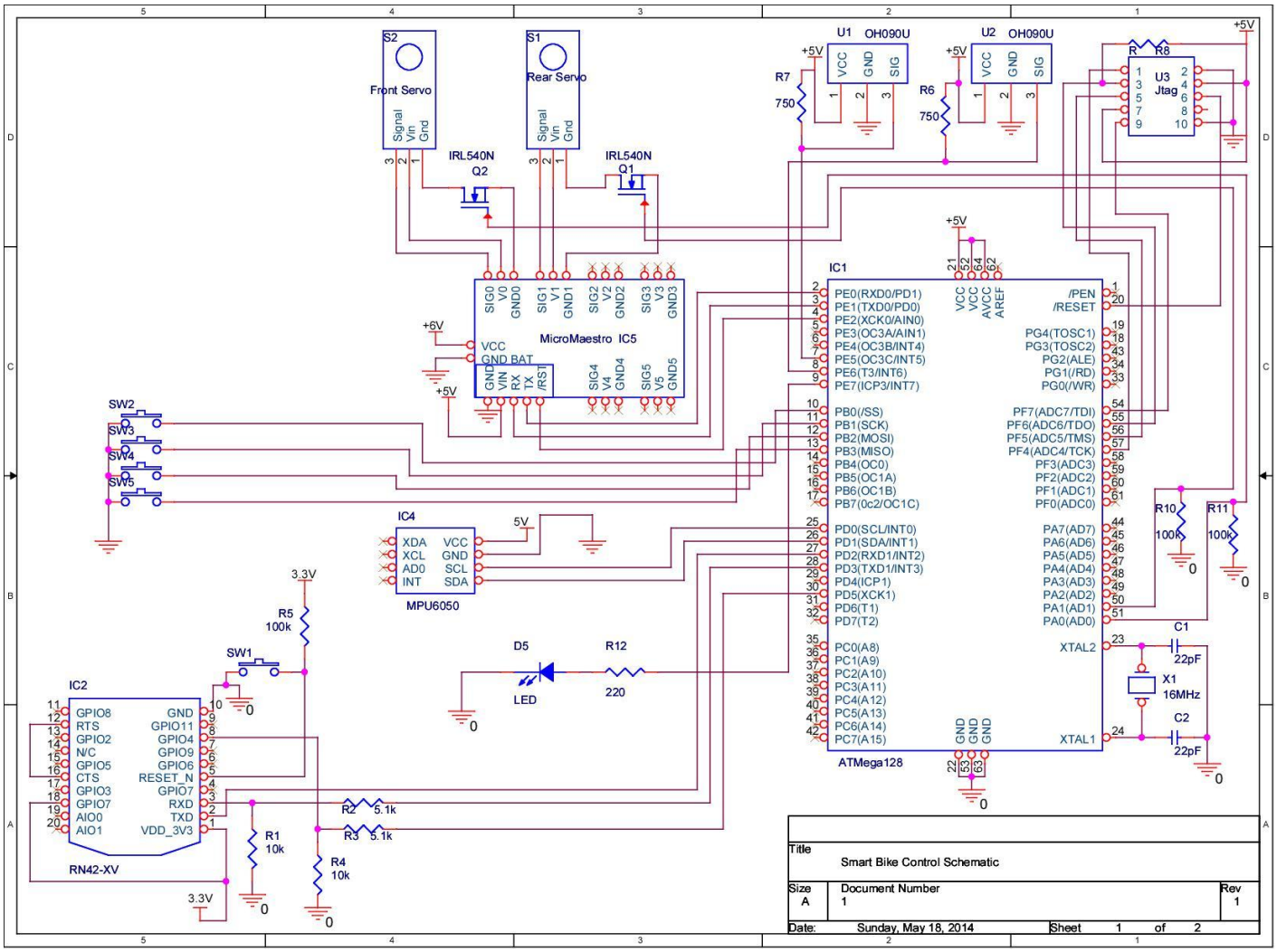

Figure 22: SmartBike Control Schematic

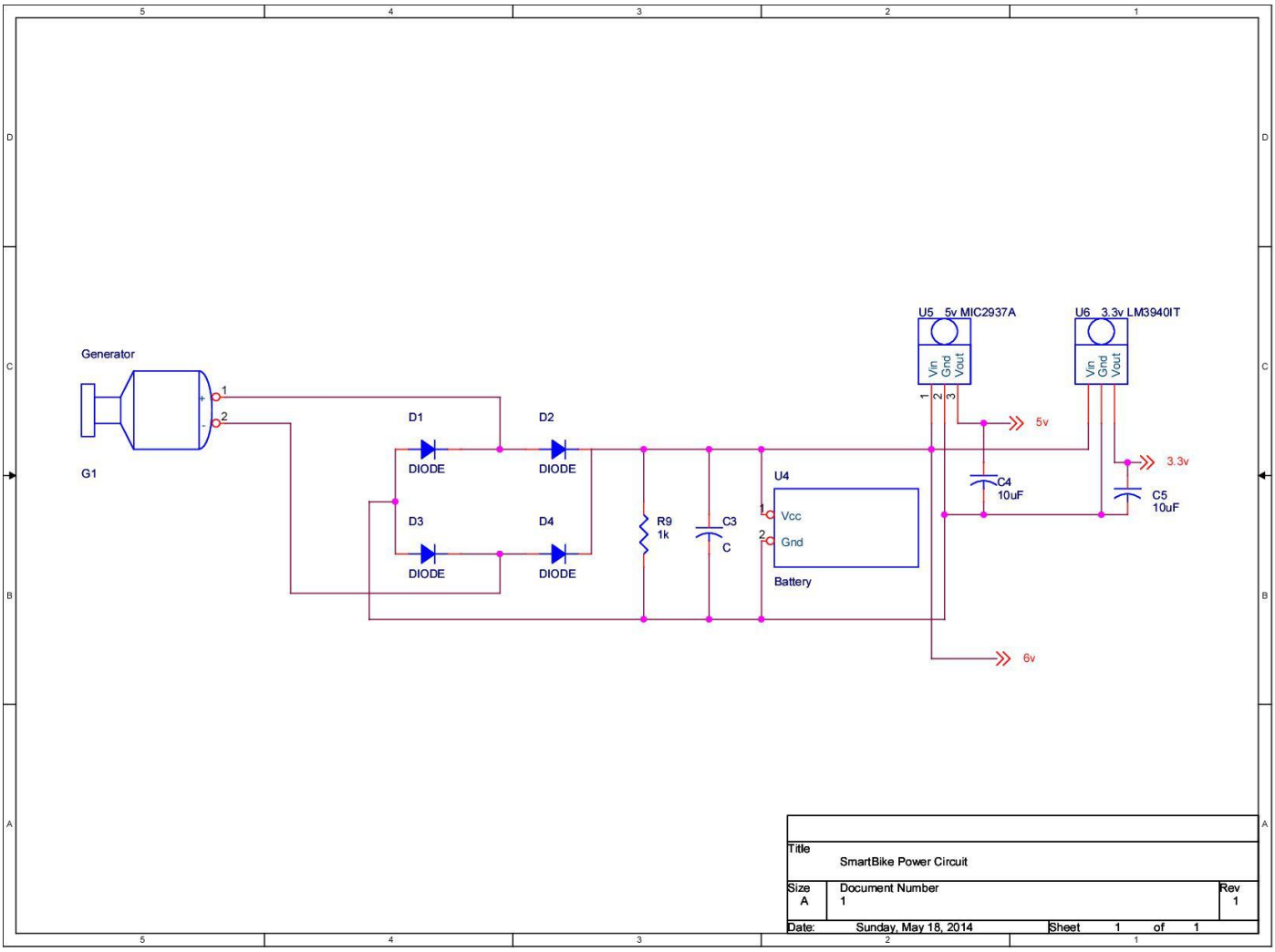

Figure 23: SmartBike Power Schematic

The bill of materials, including shipping, for the entire design is shown in the table below

Vendor	Item	Price	Quantity	Subtotal	Shipping	Total
SBU	Printed Circuit Board	$0.00	1	$0.00	$0.00	$0.00
Amazon	Kootek MPU-6050	$6.27	1	$6.27	$0.00	$6.27
Amazon	Sunkee Tower-Pro Servo	$14.28	2	$28.56	$0.00	$28.56
Amazon	6V Battery	$10.36	1	$10.36	$0.00	$10.36
Amazon	SMAKN MG996R Servo	$13.50	1	$13.50	$3.99	$17.49
Craigslist	Bicycle	$30.00	1	$30.00	$0.00	$30.00
Digikey	Hall Effect Sensors	$1.75	3	$5.25	$2.86	$8.11
Ebay	Bike Trainer	$25.01	1	$25.01	$0.00	$25.01
Ebay	400 female headers	$4.18	1	$4.18	$1.99	$6.17
Ebay	Bottle Dynamo	$19.50	1	$19.50	$0.00	$19.50
Hobbyking.com	Aerostar Servo	$23.74	1	$23.74	$9.99	$33.73
Home Depot	6x18” aluminum sheetmetal	$8.97	1	$8.97	$0.00	$8.97
McMaster-Carr	Ball-joint linkage	$5.48	2	$10.96	$0.00	$10.96
McMaster-Carr	Hex Standoff	$0.99	2	$1.98	$0.00	$1.98
McMaster-Carr	1-meter Threaded Rod	$6.23	1	$6.23	$0.00	$6.23
McMaster-Carr	Clevis Rod end	$6.36	2	$12.72	$5.13	$17.85
SBU	Assorted Resistors	$0.00	11	$0.00	$0.00	$0.00
SBU	Assorted Capacitors	$0.00	3	$0.00	$0.00	$0.00
SBU	Steel Sheet Metal	$0.00	1	$0.00	$0.00	$0.00
SBU	Male Headers	$0.00	30	$0.00	$0.00	$0.00
SBU	Voltage Regulators (5V and 3.3V)	$0.00	2	$0.00	$0.00	$0.00
SBU	Pushbuttons	$0.00	5	$0.00	$0.00	$0.00
SBU	1N4002 Rectifier Diodes	$0.00	4	$0.00	$0.00	$0.00
Sparkfun	Xbee Breakout Board	$2.95	1	$2.95	$0.00	$2.95
Sparkfun	Pololu Micro Maestro	$19.95	1	$19.95	$3.95	$23.90
Sparkfun	RN42-XV Bluetooth Module	$20.95	1	$20.95	$4.09	$25.04
Walmart	27” innertube	$4.96	2	$9.92	$0.00	$9.92
			Subtotal:	**$261.00**	**Total:**	**$293.00**

Table 1: Bill of Materials

While we failed to stay within our allotted budget of $230, we did come close. Had we not experienced two servo failures we would have saved $51.22, which would have brought us down to $241.78. While this is still over budget, it is much closer to the allotted $230 and much further away than our original estimate of $309.19. The drastic price difference was mostly due to the choice in dynamo. By choosing a $19.50 bottle dynamo over a $70+ hub dynamo, we saved quite a bit of money (not including service fees for installing the hub dynamo).

Testing

As in any large system, components were first tested on an individual basis. Then, as the system grew, it was tested intermittently until the final product and desired results were achieved. Our methodology will be discussed in the following paragraphs.

Our electronic derailleurs, the most important part of the system, were rigorously tested to ensure smooth, hassle-free shifting. They had to, as much as possible, emulate the original mechanical derailleurs. That is, they had to shift into every gear flawlessly. Originally we tested the bicycle upside-down so that we could pedal and shift. However, we realized that gravity was now working in the opposite direction (and probably affecting our calibration), so we needed a better solution. We purchased the magnetic bicycle trainer shown below to allow us to test in-place.

Figure 24: Indoor Cycling Trainer

This tool allows cyclists to train indoors when they cannot ride outside; it's essentially a treadmill for bicycles. The magnetic rotor at the bottom of the trainer can be adjusted to add or remove resistance and change the difficulty of the ride. This proved to be invaluable. Once the servos were mounted (but not yet connected to the derailleurs) and we had the trainer in hand, we attached servos to the Pololu servo controller and began testing in the provided Windows client. Using this method we determined how to best attach the servo horns to ensure that the derailleurs retained their maximum range of motion across all their respective gears. Next, we had to calibrate the derailleurs themselves using the high and low limiting screws. These ensure that the derailleurs do not extend beyond the top and bottom gears. Once the screws were tightened accordingly, the servos (still not attached to the derailleurs to avoid damage) were rotated with the Windows client until they appeared to line up with minimum and maximum positions of the derailleurs. Once the approximate maximum and minimum servo positions were discovered, we could safely attach the servos to the derailleurs and fine-tune these values. Next, the most logical thing to do was to find the required servo positions of the other gears and

record the values in our look-up-table using a trial-and-error approach. As we tested, we noticed that these positions were slightly different depending on the direction the servos were moving across the gears. This was due to the difference in torque required for the servo to shift the chain into gear. So we recorded two positions for each gear, again using a trial-and-error approach. We would move the positions slightly up and down until it could easily climb into the gear under test. We did this for every possible gear combination ($3\ front\ gears\ \times 7\ rear\ gears = 21\ combinations$). As we did this we performed a couple stress tests. The first was to try and shift across all the gears, one by one, from one extreme to the other and back. However, once the bike passed this test, we developed a more rigorous test which we dubbed the "snake," for reasons which will soon be apparent. We would choose a starting gear and "slither" across the gears. That is, we would choose a starting gear, let's say 7 in the rear for argument's sake. We would shift from 7 to 6 and back to 7. Then from 7 we would shift down to 5, up to 6 again, then down to 4 and back to 5 and so on. This allowed us to ensure that the chain could move not only linearly across the gears, but in a somewhat unstructured fashion. This is important because it mimicked how an average cyclist would shift. For instance, if starting in gear 7 in the rear, a cyclist wants to shift to gear 3 and then sometime later to gear 4. They would not go down to 3 and then all the way down to 1 in order to go up to 4, this is inefficient and a waste of time. They would merely go from 7 down to 3 and then up one gear to 4. In the computer science world, a good analogy would be a linear search versus a binary search. Once we were able to shift into all gears with ease, we considered the test complete.

The next most important component is our Hall-effect sensors. Since these are used to calculate speed and cadence, it is critical that these be fully operational at all times. We had to make sure that all magnets are sensed as they pass by the sensors. Of course the most logical way to do this was to observe the sensor outputs on an oscilloscope and count the number of pulses on the screen. As we did this, we noticed that some of the magnets were too far from the sensors. As a result, we had to adjust their positioning or stack more magnets on top: we took the former approach for the tire and the latter for the crank. Yet, some magnets were still not being detected. Since the sensors require the presence of a magnetic south pole to trigger, we had to ensure that all the magnets were positioned so that their south poles face the sensors. Once all the magnets were successfully detected, we needed to ensure that we were obtaining accurate results. We did this using a combination of our IDE's debugger and a stopwatch. We set the watch to count down for a number of seconds and counted the number of tire and pedal rotations. We compared our manual count with the results from our debugger to ensure that they were in fact the same.

As mentioned, we did not originally debounce our buttons for simplicity. Once we refined the code and added a debounce routine, we had to ensure that it worked. Again, the debugger was used to ensure that the debounce routine returned the correct button press. Breakpoints were placed in the block of code under question and when the corresponding button was pressed we waited for the debugger to pause the program. If it did not pause then

we knew something was wrong and needed to be fixed. We did this for both single and double button combinations.

Once we were able to shift and control the bike, we could begin to test manual shifting. We added some resistance to the trainer to make the test more realistic. We repeated our "snake" test to ensure that the bike shifted into the proper gear smoothly under the added stress and under microcontroller control.

Next we implemented and tested the dynamo and the charging circuit. We used the oscilloscope to view the waveform produced by the dynamo as we pedaled. We attached a second probe to various points in the circuit to ensure that it was operating properly relative to the input sine wave. We began by placing the probe inside the bridge rectifier to ensure that our sine wave was being fully rectified. Next we tested the output of the rectifier and observed the ripple on this new signal. We then measured the time constant and adjusted the RC filter accordingly to minimize the ripple as much as possible. However, this wasn't deemed to be too much of an issue as the battery essentially acts as a giant capacitor and filters out any remaining ripple.

Once we had a DC output from our generator, we could charge our battery. Since we are using a lead acid battery, it is very tolerant to voltage and current fluctuations. In order to charge the battery, approximately 6.8 volts needs to be applied across the terminals. Once the cyclist pedals fast enough to produce this voltage, the generator circuit becomes a constant voltage source rather than a constant current source. It is then able to feed the battery until it reaches this threshold or the cyclist stops pedaling. In order to test this, we placed the oscilloscope probes across the battery terminals and a digital multi-meter in series with the battery to measure current. We then pedaled for extended periods of time and observed the voltage increase. Through these tests, we determined that a cyclist can produce up to 700mA with enough speed – more than enough to charge the battery.

Once these systems were operational, we were able to move our circuit from the breadboard to the PCB. This was somewhat of a gamble because any major problems could have delayed the project for weeks. Much time was spent testing the board for short circuits and proper connections before anything other than the vias were soldered to the board. This was largely done with visual inspection and the continuity function on the multi-meter. Excess copper was shaved away and short circuits were eliminated. Since via pads are rather small, there was an inherent risk of excess solder reaching undesirable areas of the board. This had to be eliminated. Once the unpopulated board was free of problems, we began to populate it, all the while continuing to test for shorts and erroneous connections. However, now we were also interested in ensuring that proper connections were made. Again, using the continuity tester, we ensured continuity between the ends of all traces. In some cases, heat from the soldering iron lifted traces off the board and thus the continuity tests failed. Jumper wires were soldered between the broken traces to fix these problems. Once everything was validated, it was safe to apply power to the system.

Once the PCB was completed, our system was portable. This allowed us to test outdoors; we were no longer completely dependent on the trainer. At this moment in time, we had simultaneously finished our automatic shifting algorithm. We were now able to test both automatic and manual modes outdoors. Again, this was done through stress tests. The bike was ridden over rough terrain such as grass and dirt and was also ridden through the academic mall at various speeds. We needed to ensure that the bike could shift properly under all of these conditions. If not, values in our lookup table would have to be updated again. Automatic mode, in particular, was tested by intentionally raising and lowering our speed and checking if the bike maintained our cadence via the Android app (which still had some Bluetooth bugs at this point, but was functional enough to act as a display). Once the bike met our expectations, we allowed peers, who have never used our bike before, to ride it. This ensured that the bike was in fact comfortable to ordinary riders and helped to eliminate any bias from our testing procedures.

Around the same time we implemented our hill-climbing mode, which helped us to refine both transmission modes. In order to test this, we both manually and automatically shifted into many gear ratios other than 1 in the front and 6 in the rear. We then pressed the respective button combination and watched the chain move back to gears 1 and 6, one gear at a time, in sequence. We of course tested while the system was already in 1 and 6 too. This ensured that we didn't move to some arbitrary gear ratio and that we remained in 1 and 6.

Before we were able to complete the above testing, the Android app had to be functional so that we could verify our gearing was correct for specific speeds and cadences. So our Bluetooth module had to be tested to guarantee strong communication with the bike. In order to independently test the Bluetooth module, a freeware terminal emulator called CoolTerm (shown below) was used.

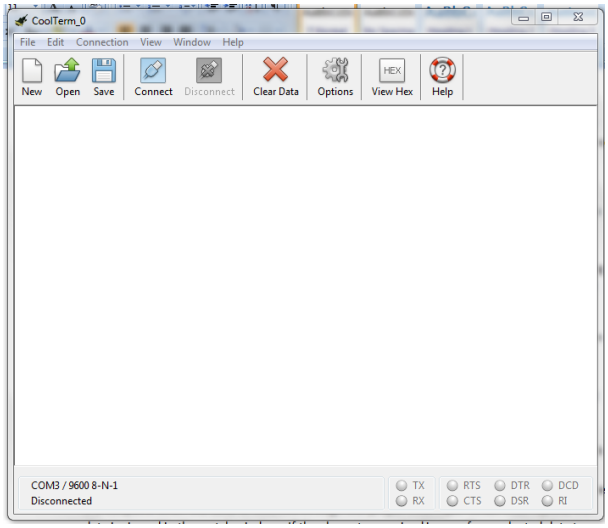

Figure 25: CoolTerm Terminal Emulator

It allows users to send data serially to COM ports on their PC. After pairing the Bluetooth module with our computers, Windows 7 and 8 automatically installed drivers for it and assigned

it a COM Port. We then proceeded to connect to it with CoolTerm and sent data to it. We emulated the Android app by sending the same command characters shown in **Figure 9**; the response was verified against variables in the debugger. Then the Android app itself needed to be tested. Using the debugger, we verified that the app was in fact sending the correct commands to the bike. We then allowed the bike to return the requested values and verified them on the app against the debugger.

Results and Discussions

Result Analysis

Our finished prototype is still somewhat rough around the edges and not very aesthetically pleasing. Exposed wiring and the lack of a case for the circuit board are some undesirable characteristics. However, our results far exceeded our expectations given the number of setbacks we encountered.

While electronic shifting does work, there are some flaws which could be improved. For instance, the derailleurs are somewhat sensitive to excessive motion and they are still somewhat fragile despite being made of steel. For instance, if the brackets (the rear in particular) unintentionally hit something and bent, the whole system would need to be recalibrated. In an ideal world, the brackets would be machined from a solid piece of metal and mounted on the bike.

After much debate and arguing among ourselves, we decided to grant the user full manual control of the bike. While this does not prevent cross chaining, it gives the rider the maximum freedom to do as they please (which may be desirable for an advanced cyclist). Therefore it is advisable for beginners to remain in automatic mode when possible. If time allowed, we would have liked to incorporate a semi-automatic mode as well which would work to eliminate cross chaining. Full manual mode would have to be unlocked using a password in the app. In summary, in order to shift between front gears 1 and 2, the chain should be in gear 5 in the rear. Likewise, in order to shift between gears 2 and 3 in the front, the chain should ride on gear 3 in the rear. So in the case that the user is not in gears 3 or 5 in the rear when shifting in the front, we cannot guarantee that the chain will jump into gear. Based on a sample size of 30 trials, we determined that we have a failure rate of 40% when shifting under these extreme conditions.

In automatic mode we don't run into any of these problems. Our microcontroller only allows the front gears to shift once our rear gear has entered the proper set-up gear. This means when shifting in-between gears 1 and 2 in the front, our rear gear must be in gear 5 and when shifting between 2 and 3 in the front, our rear gear must be in gear 3. This system gives us the least amount of error while shifting, although missed shifts can still occur. When a shift failure occurs (i.e. the front servo attempts to shift into gear 2 from 1 and fails), the rider will either be going too fast or slow and the system will correct itself on the next shift. This system works as planned: it adjusts for the users change in speed by shifting our gears to keep the rider's cadence in the preferred range.

Our application works well and displays all ride statistics to the user, updating every second. However, the only issue is that when the bicycle is prematurely turned off, the green "Safe To Power Down Bike" message appears. This was added very late into the design phase and should easily be fixed by adding a value to the enumerated types of power states or rearranging them. The reason this happens is because when power is cut from the bike, the app

receives no data. Presumably it thinks a zero was sent. Since the enumeration for this state has a value of zero anyway, it thinks the bike entered shutdown mode although it was shutdown prematurely instead.

Our speed and cadence sensors work well. The system we used counts the number of magnet passes (ticks) in one second. This system is not the most accurate and has its faults as we already mentioned. The resolution of each sensor is much lower and hence we get coarser measurements than had we used our original idea of counting the time between magnet passes. Again, the resolution can be increased by adding more magnets or increasing sample time. However, the fact that we have a fixed resolution aided us in our design. It allowed us to use the number of ticks as an index to a lookup-table. The lookup-table in question stores the gear positions corresponding to the number of ticks. As a result, we got a range of values which were easier to work with. So this implementation had both drawbacks and advantages.

Multi-Disciplinary Issue

By now, it should be very clear to the reader that this project is very multi-disciplinary. Not only does it consist of electrical hardware and software, but it also consists of mechanical hardware design; which we have little background and knowledge in. As a team, overcame these deficiencies so that we were able to progress through the mechanical aspect of the design. In order to do this, we consulted with mechanical engineering friends, cyclists, and machinists in addition to using some good old fashioned common sense and creativity which any decent engineer must possess. Through these means, we believe we have arrived at a more than adequate solution to one of the most difficult parts of the project.

Obviously, the electrical and software portions of this project were easier since these are areas which we have been educated in over the past four years. Of course, mechanical issues aside, the electronics themselves are quite multi-disciplinary. Clearly quite a bit of electrical engineering and knowledge of power systems was required to determine how best to power the system. Further electrical knowledge was needed to adequately layout the PCB and to design the circuit in general.

Of course the other third of the project was software-based; both low-level software (microcontroller) and high-level software (Android application) were taken into consideration. A modular approach was taken in the software development part of the project. That is, the code was organized in different files based on its functionality. For instance, all Bluetooth related functions made up one module while servo related functions made up another. This was something taught in various classes such as ESE 381 and CSE 219, but was heavily reinforced during Frank's internship with Texas Instruments. This allowed for agile, easy to debug code. As the code grew more complex, this became essential. Version control was also another important tool in our project. Version control (SVN) is something that was emphasized in computer science classes and was consequently used in this project. It allowed us to share code across multiple computers remotely and ensured that our code was always up to date. We did not have to deal with emailing code to each other or using other mediums like flash drives. It also allowed us to branch off and experiment with the code while always maintaining older versions in an online repository (CloudForge) which we could always revert back to if needed.

Professional/Ethical Issues

When dealing with any form of transportation (albeit walking or swimming), it usually involves human beings traveling at high, potentially dangerous, velocities. Automobiles have come under scrutiny for the large part of their existence. One of the most recent issues is Toyota's sudden acceleration issue. However, the automobile has been plagued by other disastrous design decisions throughout history. Other automobiles such as the Ford Pinto and the Chevy Corvair met much scrutiny for the failures of their designers.

Although there are much fewer fatal cycling accidents than automobile accidents, safety is still an important issue for cyclists. In today's modern world, technology is often times more of a curse than a blessing. Automobile drivers are constantly distracted by the very technology mentioned in the abstract of this document. Too often drivers pay more attention to their texting buddies, stereo system, and GPS devices than the road itself. It is no wonder Google, Mercedes-Benz, and others have developed self-driving cars; may people are narcissistically caught up in their own lives rather than taking concern and responsibility for their driving habits. Cyclists and pedestrians must deal with these incapable drivers on a daily basis. One of the goals of the SmartBike is to allow the cyclist to concentrate on their ride without having to worry about operating the bike. This allows the rider to pay more attention to the road and potential hazards in a world of those who otherwise cannot.

Of course, in order to ensure such a safe ride, it is essential that there be no major bugs in the system. The bike was put through rigorous testing over various terrains to ensure that the bike does not become stuck in a single gear, the chain does not become loose (or too tight so that it snaps under pressure) and unexpectedly falls off, and etcetera. A bike which advertises safety should be as reliable as possible to avoid disturbing the cyclist. Should the bicycle be mass produced, it will undergo testing with an independent agency such as Underwriters Laboratories (UL) or Dayton T. Brown to ensure that it meets and exceeds current cycling safety standards set forth by the Consumer Products Safety Commission (CPSC) and others.

Since we are not the first to implement this idea (although it is still a relatively new concept) we must be wary of intellectual property infringement. Through our research, we have stumbled upon a number of patents for similar bicycles; however, all of these patents have long expired (and our bike likely uses newer, more advanced technology than most of them). The only major concern would be the Autobike, which does not appear to be patented. Of course, a major difference between our bike and the Autobike is the transmission (conventional derailleurs versus a continuously variable transmission). Therefore the implementation of the system was considerably different.

Impact of Project on Society or Contemporary Issues

To quote Albert Einstein, "It has become appallingly obvious that our technology has exceeded our humanity." In today's society, many people are very content spending a beautiful summer's afternoon indoors watching television or surfing the web. Many adolescents choose to interact with their peers virtually rather than physically. Nowadays, it's not uncommon to see a young kid immersed in a videogame while sitting on a tropical beach or a teenager engrossed in a text message conversation at the dinner table, eyes glued to the screen. Many adults are guilty of these things as well. In this fast-paced technological world in which we live, many people are simply unaware of the beauty that surrounds them; they are willfully ignorant of the changing seasons; virtually unaware of the falling leaves in autumn, the first frost of winter; the smell of a freshly cut lawn in spring, or the sound of a crashing wave on a warm summer's day at the beach. Gone are the days of a neighborhood game of tag or hide-and-seek on a warm summer night, snowball fights on the snow-days of winter, or the bicycle ride to the local burger joint or ice cream parlor. These joys of life are of a bygone era and may never be experienced by future generations. To many, the outdoors is about as foreign as another country and about as undesirable as its tap water. Many are friendlier with people they've only virtually met on social media than they are with their own neighbors and family. In the workplace, many people email their co-workers across the hall rather than interact face to face. In the same way that drivers pose a threat to cyclists, technology is in many ways a curse, rather than a blessing, which sucks the life out of us. As a little known country artist, Bradley West, sings, "On the altar of technology, we sacrifice our souls."

Of course technology in and of itself is not life-draining. As a whole, technology has rewarded society. Nonetheless, the more people abuse it, the less human they become. Certainly we cannot un-invent these abusive technologies nor can we halt their advancement. As the technological snowball races downhill and grows larger, the few snowflakes that fall off will not make much difference in the race to the hill's bottom. However, that snowball may be redirected instead. This is where the SmartBike comes into play. People are almost always excited about new technology and new ways in which they can use their smartphones. People are fascinated by technology more so than nature. Future iterations of the SmartBike will be one of these fascinating technologies. Our goal is to get people outdoors again by bringing technology to the bicycle; our hope is that it will draw people out of their homes and back into nature; and our mission is to help people view technology as a tool rather than a way of life.

In other respects, by promoting bicycle use, we can help curb the price of oil and please environmentalists simultaneously. We can help overcome the childhood obesity epidemic and help people achieve longer-healthier lives through adequate exercise. Bicycling has been scientifically proven to be easier on the joints than running and most find it more enjoyable anyway.

Summary and Conclusion

Despite the many challenges, setbacks, and disappointments we had with this project, overall we are very satisfied with the results. It proved to be very challenging and was one of the few ESE projects which incorporated mechanical engineering into its design. It has tested our abilities to brainstorm and come up with solutions to various problems; a skill which will be useful in the future when we graduate and enter industry. The nice thing about this project is that there is still much room for improvement; the platform we built can be inherited by future senior design teams for many years to come in multiple departments, not just ESE.

For instance, an entire project could revolve around sensing what gear the chain is in. Our current system relies on blind faith that the chain successfully reached the desired gear. Instead, it would be desirable to use optics or an analog Hall-effect sensor to guarantee that the bike is in the correct gear. There is also much that can be improved with the app; this may be an inexpensive, software-based project for a team of computer engineers. Of course, the pressure sensors were not able to be implemented due to mechanical constraints; this would make an excellent project for mechanical engineers or as joint-departmental project. As we mentioned earlier, a semi-manual mode would be an improvement to the current design in order to prevent cross chaining while giving the user some control of the derailleurs. As was already made evident, the IMU was almost completely removed from the project. It would be a good idea to replace it with a barometer or inclinometer, which may be much easier to use and provide better results. Finally, another project could revolve around calibration. As evident by this paper, our calibration techniques were rather ad-hoc and would easily confuse anybody who did not take part in the design of this bike. We would like this to be greatly improved in future designs.

Clearly, this hands-on experience was a learning experience which we will not soon forget. We hope that this project is one the department remembers as well. We hope that this project will live on and continue to develop in the near future. Most importantly, we hope to look back on the many setbacks and tribulations we encountered several years from now and apply what we learned from them wherever we may end up in life.

Acknowledgements

The authors would like to acknowledge Professor Kenneth Short for considering our project and taking the time to coach us through the development cycle. His wisdom and practical knowledge has been priceless. We would like to thank him for his willingness to venture into this project, despite its mechanical nature, and his patience as we struggled to design the system as a whole.

We would also like to recognize Scott Tierno for his time, insight, and eagerness to help us along as we developed this project. Mr. Tierno has offered many design suggestions which were extremely valuable in the final product, namely the button combinations. He has been more than generous in offering his time and the use of his facilities to assist in this project.

Professor David Westerfeld's advice has been an indispensible resource throughout the project's development As an avid cyclist and an electrical engineer, his practical knowledge and constructive criticism has helped us to choose the best solutions to project issues that came up throughout the project development, both a cyclist's and engineer's perspective.

Professor Mikhail Gouzman has been more than generous in offering the use of his laboratory facilities for our project. This project would never have been possible without his assistance; without a PCB, this project would have never made it off the breadboard. His PhD. Students Joe Cheang and Vladimir were more than generous with their time; they helped us learn how to use the PCB mill from start to finish.

While we did not spend much time in the senior design lab, Tony Olivo was more than happy to aid us in debugging parts of our circuit, namely the second servo, and offered practical advice along the way.

Anthony Tom, a good friend of ours and a mechanical engineering senior who works in the machine shop, has been more than generous with his time and mechanical knowledge. He helped us get our mechanical struggles off the ground and allowed us to progress to the critical electrical part of the project. With his help and the use of the machine shop, we were able to quickly manufacture the needed mechanical parts with precision and ease. Good friends and good tools have made all the difference in this project.

Machinist Henry Honigman has also been an indispensable resource. His wisdom and craftsmanship have helped us to realize the flaws in our initial designs as he offered us better, wiser solutions. He has offered us many suggestions on how to strengthen our servo mounts simply by making the proper bends and cuts in the metal.

References

[1] Maness, Michael. “BICYCLE OWNERSHIP IN THE UNITED STATES: EMPIRICAL ANALYSIS OF REGIONAL DIFFERENCES”. http://www.academia.edu/1839374. Published: November 15 2011. Accessed: November 29 2013.

[2] Calamur, Krishnadev. “In Almost Every European Country, Bikes Are Outselling New Cars”. http://www.npr.org/blogs/parallels/2013/10/24/240493422/in-most-every-european-country-bikes-are-outselling-cars. Published: October24 2013. Accessed: November 29 2013.

[3] Dilthey, Max Roman. “Multiple Gears vs. Single Gear Bikes”. http://livehealthy.chron.com/multiple-gears-vs-singlegear-bikes-3495.html. Accessed: December 1 2013.

[4] Overholt, Zach. “Autobike Builds First Continuously Variable, Automatic Shifting Bike”. http://www.bikerumor.com/2013/03/20/autobike-builds-first-continuously-variable-automatic-shifting-bike/. Published: March 20 2013. Accessed: November 15 2013.

[5] dbc1218. “USB Bike Generator”. http://www.instructables.com/id/USB-Bike-Generator/?ALLSTEPS. Accessed: November 21 2013.

[6] “Shimano Dura Ace 9070 Di2 Drivetrain Upgrade Kit”. http://www.performancebike.com/bikes/Product_10052_10551_1147610_-1_400608__400608. Accessed: November 19 2013.

[7] Schneider, David. “DIY Electronic Bicycle Derailleur: Upgrade your bike with a microcontroller and servo”. IEEE Spectrum. ,

[8] GroundPepper. “OpenShift Prototype One. Servo Controlled Derailleur”. https://www.youtube.com/watch?v=m23kgfQll7E. Published: March 14 2013. Accessed: October 27 2013.

[9] GroundPepper. “OpenShift - Homemade Electronic Shifting (Bicycle) - Part 1”. http://www.youtube.com/watch?v=xXwDmX9Ly_U. Published: April 4 2013. Accessed: October 27 2013.

[10] “Road Cycling - Homemade Electronic Shifting” http://www.bikeforums.net/archive/index.php/t-626668.html. Accessed: October 27 2013.

[11] “Ball Joint Linkages”. http://www.mcmaster.com/#catalog/119/1186/=prafbk. Accessed: November 10 2013.

[12] “Clevis Rod Ends”. http://www.mcmaster.com/#catalog/119/1193/=prahvi/. Accessed: November 10 2013.

[13] “Fully Threaded Studs & Rods”. http://www.mcmaster.com/#catalog/119/3147/=pramzm. Accessed November 10 2013.

[14] “Standoffs”. http://www.mcmaster.com/#catalog/119/3238/=pranl3. Accessed November 10 2013.

[15] David A. Mitchell. “Bicycle Safety and Bicycle Standards”. ASTM International. Published March 2006. Accessed December 8 2013.

[16] Ganssel, Jack. "A Guide to Debouncing - Part 2, Or, How to Debounce a Contact in Two Easy Pages." Debouncing Contacts Part 2. Web. 19 May 2014. <http://www.ganssle.com/debouncing-pt2.htm>.