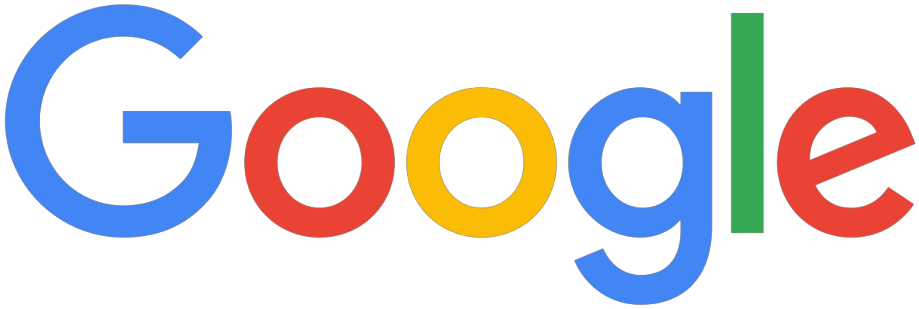


Introducing Google Go:
A New and Safer Electric Scooter Experience Powered by Google.

Samuel Bruner
Ming Hsieh Department of Electrical Engineering
University of Southern California
sbruner@usc.edu

Dae Kim
Ming Hsieh Department of Electrical Engineering
University of Southern California
daek@usc.edu

Juan P. Mejia
Ming Hsieh Department of Electrical Engineering
University of Southern California
mejiasil@usc.edu

Hyun-Bum Yang
Department of Computer Science
University of Southern California
hyunbumy@usc.edu

Table of Content

1. INTRODUCTION

1.1. Project Requirement

The objective given was to develop a "smart" device that could enhance a recreational activity. Our team chose to develop a "smart" scooter that increased the rider's safety. Our product was also required to consider its ergonomics, ease-of-use, expandability, and reliability. We were also asked to incorporate at least 3 unique inputs and outputs that captured information and communicated it to the main microcontroller of the system.

1.2. Product Overview

The Google Go aims to provide the safest riding experience on the market, in a way that is unique compared to its competitors. Existing scooter service providers largely utilize the same hardware, which is sourced from Xiaomi. These companies buy the scooters from the Chinese manufacturer and simply add their logos and networking technology. Google Go offers a new product with enhanced features, placing it ahead of its competitors.

1.3. Concept and Design

Our team decided to have a modular approach for the design of our product. Each module had its own specific function for adding safety to the scooter. Our original design intended to place the inputs and outputs of each module on various locations throughout the scooter as shown in figure 1. For example, the pressure sensors were placed over the grip for the handlebars and the proximity sensors were to be placed at the rear of the scooter. All the inputs and outputs for our modules would then be connected to the ATmega328 microcontroller shown in the following section. The microcontroller would be placed along the vertical steering bar of the scooter using a clamp as shown in figure 2. The overall design of the product was to have the modules incorporated unto the scooter in a way that was unnoticeable to the rider. The four safety features implemented were smart speed regulator, haptic feedback, automatic headlights, and a GPS display. How these features were implemented will be further explained in the following sections.

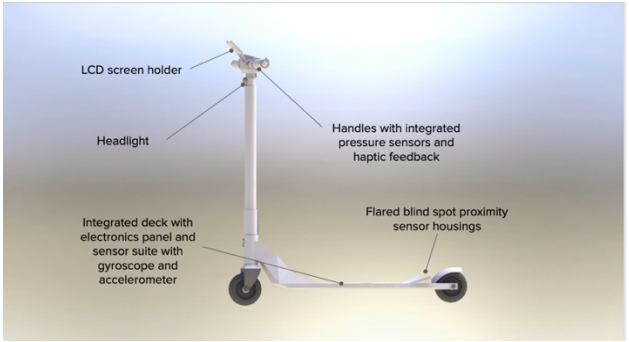


Figure 1: Google Go Safety Features

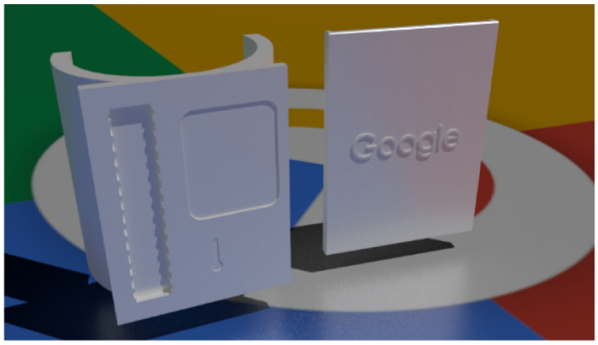


Figure 2: Microcontroller/Module Housing

1.4. Implementation

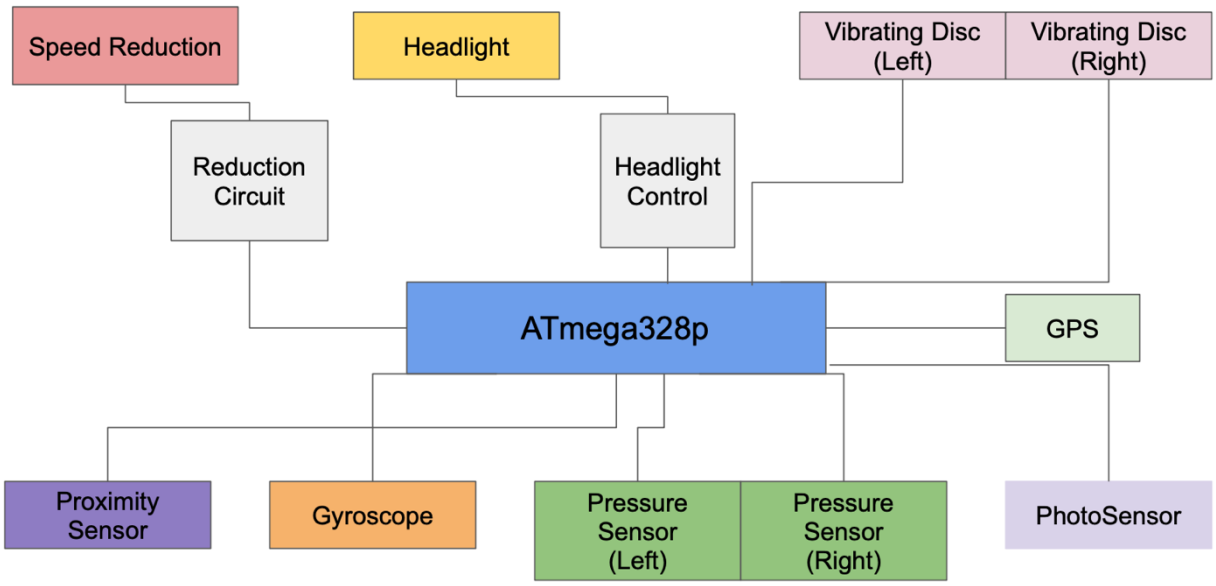


Figure 3: Input and Output System Schematic

The intended design of the scooter, as previously mentioned, was to have all the external module inputs located throughout the scooter. However, due to time constraints, we were only able to place the pressure sensors and the vibrating disks in the intended locations. The rest of the inputs, such as the proximity sensors, gyroscope, and photosensor were placed on the development board as shown in Appendix B. The inputs gathered information and depending on what was measured, would send a signal to the microcontroller which would enable/disable the corresponding output/safety feature.

For example, our speed regulator reduced the speed of the scooter on two different use cases. The first was through the gyroscope input. The gyroscope measures the orientation or "tilt" of the scooter and if the "tilt" is great enough, it would send a signal to the microcontroller which would enable the speed reduction circuit. The second case is equivalent to the first except that the input measured is the pressure felt by the pressure sensors located on the handlebar grips. This is so that the scooter will have its speed reduced if the rider does not have a grip on both handlebars.

The other safety features act the same way as the speed regulator but instead the input and output change. The haptic feedback feature uses sonar sensors as inputs and vibrating discs located on the handlebar as outputs to alert the rider if a vehicle is nearby. The automatic headlight feature instead uses a photosensor as its input and a headlight as the output to be able to automatically turn the headlight on if the photoresistor detects a low-light environment.

2. TECHNICAL DOCUMENTATION

2.1. Microcontroller

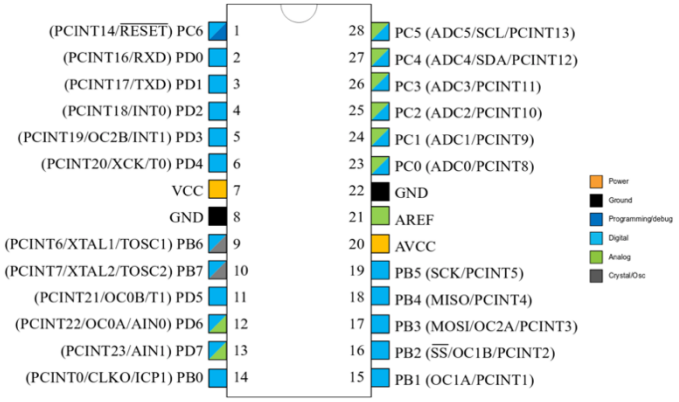


Figure 4: ATmega328p Pinout

PB0	Left Sonar Output	PC0	ALS Input	PD0	GPS Serial Input
PB1	Right Vibrator Output	PC1	Left Pressure Sensor Output	PD1	Serial Output for debug
PB2	Left Vibrator Output	PC2	Right Pressure Sensor Output	PD2	Brake Control Output
PB3	uC Programmer	PC3	Throttle Input	PD3	Downhill Output
PB4	uC Programmer	PC4	Accelerometer SDA	PD4	ALS Output
PB5	uC Programmer	PC5	Accelerometer SCL	PD5	Throttle Comparator Output
PB6	Oscillator Input			PD6	Left Sonar Input
PB7	Right Sonar Output			PD7	Right Sonar Input
AREF	NC	AVCC	NC	VCC	5V Scooter Power

Table 1: ATmega328p Pinout

2.2. Modules

2.2.1. Analog Light Sensor (Adafruit ALS-PT19)

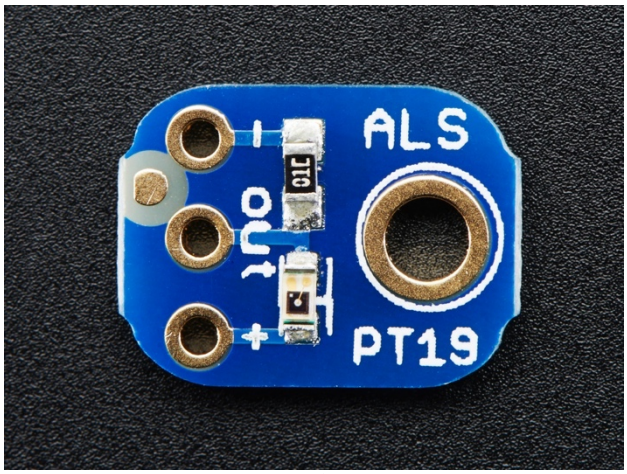


Figure 5: Photosensor

2.2.1.1. Hardware

The ALS module is responsible for controlling the automatic headlights. The module consists of three pins, Vdd, Gnd, and Out. The Vdd pin is connected directly to the board power supply and operates off 5V DC. The Gnd pin shares the board ground. The Out pin provides an analog output to the microcontroller. As more light is exposed to the ALS, the output ranges from 0V to a saturation output voltage of 4.6V. The microcontroller processes this input and provides output signal to the tri-state buffer responsible for toggling the automatic headlights. The output of the tri-state buffer is connected directly to the red wire of the push button responsible for toggling the headlights. The red wire signal is a constant 12V and it is pulled down for a minimum 250ms until it returns to 12V. In order to simulate this signal, the input of the tri-state is ground to pull the signal down while the microcontroller output controls the enable of the tri-state.

2.2.1.2. Software

In order to interface with the analog light sensor and retrieve its sensor values, ADC pin is needed to read the varying voltage level depending on the intensity of the light source and convert it to a digital value. To do so, pin 0 of port C on the microcontroller is used to receive the analog data from the sensor. As there are many other sensors that use the ADC pins to convert analog values to digital, an ADC pin needs to be specified for the ADC sampling process to retrieve values from a specific sensor.

The main program retains information about the current state of the light sensor in a variable called *is_dark*, specifying whether the sensor is detecting light above or below the brightness threshold. After ADC registers are correctly initialized, the light sensor values are used every iteration of the main loop, changing the state based on the values read. At every state transition, the program simulates a button press of the scooter by pulling PD4 low for about 250ms. Logical zero from PD4 activates tri-state buffer connected between the input voltage from the scooter's power button and GND and pulls the power button signal down for a time period, effectively generating the same signal as a short power button press needed to turn the scooter light on or off.

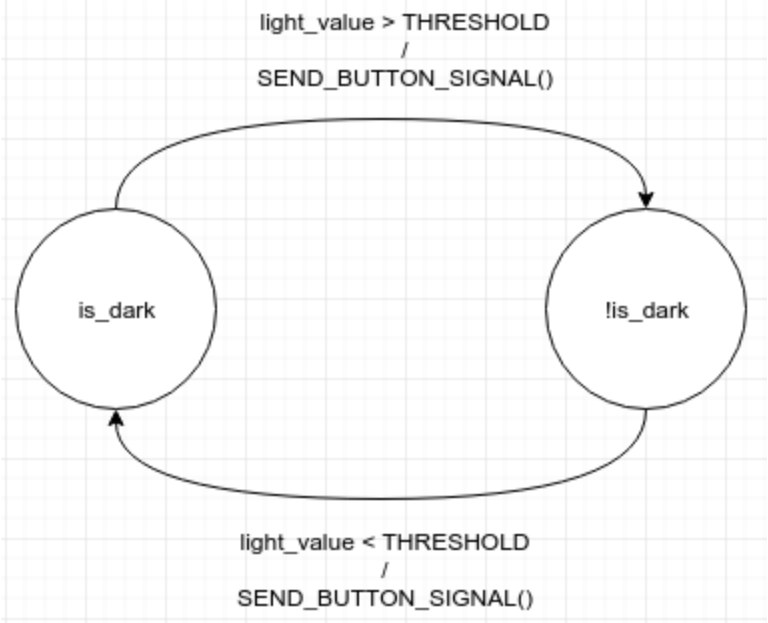


Figure 6: State Diagram for Automatic Headlights

2.2.2. Force Sensitive Resistors (Adafruit Interlink 402)

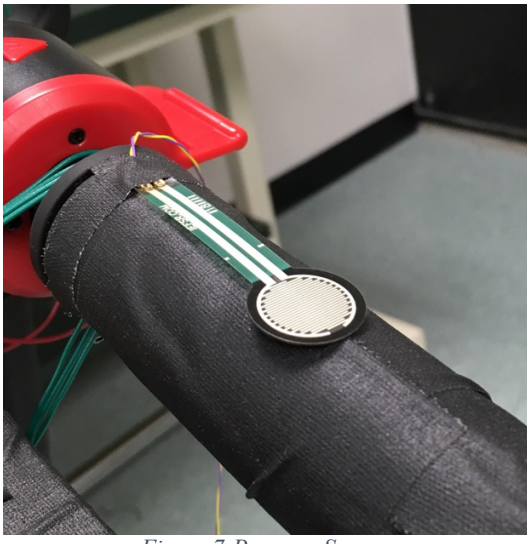


Figure 7:Pressure Sensor

2.2.2.1. Hardware

These force sensitive resistors are simply variable resistors that are controller by pressure. They are placed on each handle to determine if the rider is placing both hands on the handles. The FSRs are easily configured. One lead is connected to Vdd while the other is connected to the microcontroller. These FSRs are not particularly accurate and can vary by ~10% from sensor to sensor. However, the output to the microcontroller is not concerned with the specific analog reading but, instead, uses the reading to determine a high or low signal. If either of the pressure sensors are not being pressed, an output signal from the microcontroller enables a tri-state buffer to activate the brake and slow down the scooter. The scooter brake lever functions as a

potentiometer and outputs ~870mV when the brake is disengaged and outputs ~4.5V when the brake is fully engaged. A 10k potentiometer is placed at the output of the tri-state to accurately tune the brake output voltage to the motor controller. The Vout_Brake is tuned to ~1.75V which allows the throttle to be disengaged while the mechanical brake is applied slightly enough to slow down the scooter but not enough to abruptly stop it.

2.2.2.2. Software

Similar to the analog light sensor, the force sensitive resistors also output varying levels of voltage depending on the pressure a rider gives when holding onto the handle bars. Because of this, two ADC pins are used at PC1 and PC2 for left and right handle bar, respectively. For every iteration of the main loop, the program constantly checks the values of the pressure sensors at each handle bar, synchronously waiting for both values to return from the ADC conversions. Because handle bar status is one of the more time-sensitive safety features of the scooter, whenever the program encounters a situation where pressure sensor at either handle bars (or both) outputs a pressure value that is less than the threshold to detect proper hand placement, it immediately sends a low signal to activate the tri-state buffer. By doing so, the tri-state buffer lets the properly tuned brake signal to the scooter, making the scooter apply a soft brake to slow itself down to a halt whenever the rider does not have both of his or her hands on the handle bars.

2.2.3. Triple-Axis Accelerometer (Adafruit LIS3DH)

Figure 8: Accelerometer

2.2.3.1. Hardware

The LIS3DH is a triple-axis accelerometer with 10-bit precision. The LIS3DH can be configured using I2C or SPI. This implementation uses I2C. Therefore, most of the pins can be ignored since only Vin, Gnd, SCL, and SDA are being utilized. The LIS3DH IC has a max supply voltage of 3.6V. However, the Adafruit breakout board includes a 3.3V regulator on the PCB, which allows the dev board Vdd to be connected directly to Vin.

The LIS3DH is a major component for the downhill throttle speed limit control circuit. This circuit limits the speed of the scooter when it surpasses a decline threshold determined by the software, and it is implemented in hardware via two cascaded HK4100F relays. The Vin_Throttle signal measures how hard the user is pressing the throttle lever, and this signal is read by the microcontroller in order to determine if Vin_Throttle is greater than Vref, the threshold speed voltage. The microcontroller voltage comparator output is used to switch from the default Vin_Throttle to Vref when Vin_Throttle > Vref. The LIS3DH is used to determine the magnitude of the downhill slope. The microcontroller downhill output is used as the switching control of the second HK4100F relay. The default output of the second relay is

Vin_Throttle, and it is switched to the output of the first relay when the microcontroller downhill output is high. Each microcontroller output signal is connected to base of their respective 2N2222 NPN transistors to facilitate the switching of the relays. The output of the second relay, Vout_Throttle, is output to the motor controller.

2.2.3.2. Software

The accelerometer is used for retrieving the angle at which the scooter is oriented at. More specifically, it is used to determine whether the scooter is going down on a slope or not. The sensor communicates with the microcontroller using I2C protocol, synchronizing over SCL and SDA pins of the processor. Similar to the light sensor, slope detection uses a state machine to keep a record of its current state and produce an appropriate signal depending on its state. Slope detection operates in two states, stored in the variable *is_downhill*, and it outputs a signal at PD3 depending on its state that is used as the select bit to choose between the original throttle voltage and the limited one.

The state transition for the state machine depends on the acceleration value from the accelerometer. After registers used for I2C are all properly initialized, the program reads acceleration information from the sensor in three axes at every iteration of the main loop. The three values are then compared to the threshold set to differentiate between tilted-down orientation, signifying downhill slope, versus others. However, due to high noise level of the sensor values, a counter is also used to make sure the reading stabilizes before making any necessary state transitions.

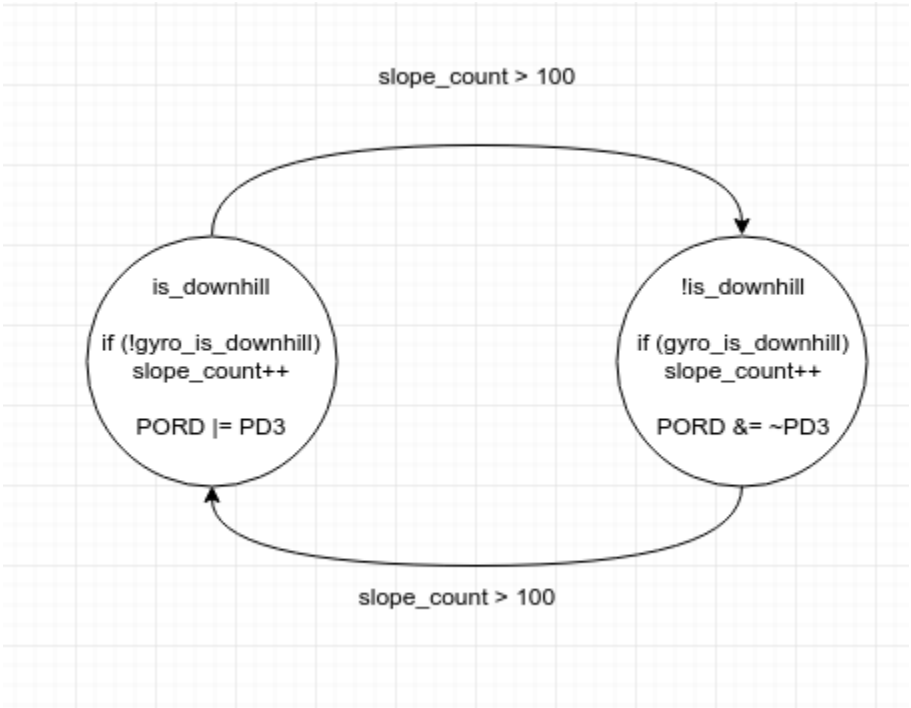


Figure 9: Scooter Downhill State Diagram

Another functionality in the program involves a comparator. In order to apply a correct limiting function to the input throttle voltage, input voltage is read through an ADC pin at PC3 and is compared to the reference voltage that the throttle is limited at when the scooter is going downhill. At every iteration of the main loop, the throttle voltage is read and compared. Depending on the comparison result, PD5 outputs a logical low or high to the relay, selecting

between the original input throttle voltage if it is lower than the voltage reference and the voltage reference if the input is higher.

2.2.4. Ultrasonic Distance Sensor (Sparkfun HC-SR04)

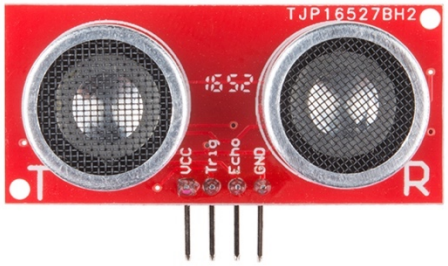


Figure 10: Sonar Sensor

2.2.4.1. Hardware

The ultrasonic distance sensor provides a sensing range from 2cm to 4m. Two HC-SR04s are placed on the left and right sides of the dev board to facilitate proximity sensing ability. This sensor operates with four pins, Vcc, Trig, Echo, and Gnd. Vcc operates at 5V which is connected directed to the dev board Vdd. The Trigger pin requires an input 10uS pulse to queue the sensor to emit an 8-cycle sonic burst at 40 kHz. The Echo pin then outputs an output pulse to the microcontroller. The microcontroller outputs a high signal to the left, right, or both vibrating disks when an object comes in proximity with the respective ultrasonic distance sensor. The vibrating disks are mounted under each handle grip in order to notify the user which side of the scooter an object is approaching from.

2.2.4.2. Software

The idea with the ultrasonic distance sensors is that they are placed at left and right tail ends of the scooter to detect upcoming vehicles when a rider is on the road. They are implemented using PB0 and PB7 for sending trigger signals and PD6 and PD7 for receiving echo signals from left and right sonar sensors, respectively. In order to properly receive and process the echo signal to get the distance data, a 16-bit timer and pin change interrupts at PD6 and PD7 are used. At each iteration of the main loop, distance values from each ultrasonic sensor are evaluated, and vibrating disks are activated if the distance from either the left or right sensor is too close through PB2 or PB1, respectively.

Due to the speed of fast-moving cars, the ultrasonic sensors are programmed to send trigger signals as frequently as possible. There is a couple of complications that need to be dealt with, however. First issue is that two echo signals sent in a proximity (both in terms of time and distance) may interfere one another, skewing the result data. The second issue is that the program should not be synchronously waiting for the entire distance measurement process, preventing other sensor measurements in the loop. In order to address these issues, a method similar to the producer-consumer model is implemented. The ultrasonic sensor code stores the most recent measurement values for left and right sonar sensors. Using the 16-bit counter for 60ms (the recommended cycle period for the sensor), the program runs a timer interrupt routine every 60ms to start the trigger signal to left / right sensor, alternating between the two sides every cycle. When an echo signal is sent out and received, another interrupt routine is run to store the timer value to the appropriate variable, left or right. This method not only eliminates the possibility of the two signals interfering one another but also guards against accidental or

non-returning echo signals. Furthermore, by asynchronously sending and retrieving ultrasonic signals through pin change interrupts, the distance checking code in the main loop simply reads the most recent distance measurement to evaluate the closeness, allowing other code to run without waiting for the completion of the entire measurement process for ultrasonic sensors.

2.2.5. GPS Module (Adafruit Ultimate GPS Breakout)

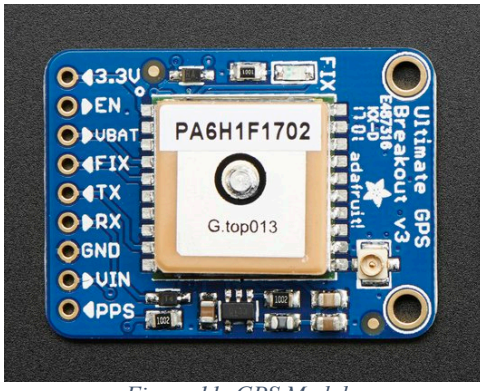


Figure 11: GPS Module

The goal of the GPS module is to eventually send its information to an LCD module to display the current location as well as to a remote server as an input to a data analytics system. The GPS module is interfaced with the microcontroller through serial communication. Using the serial receive interrupt, the microcontroller receives and stores each character that the GPS module sends. While the interrupt code and data parsing function correctly, they do cause a complication when implemented in the main loop of the program, causing the ultrasonic sensors to output incorrect distance values. This issue may be caused due to the fact that the GPS module sends a lot of data to constantly, prompting receive interrupt numerous times. Because interrupt routines are processed sequentially, it is possible that echo signal interrupt is not getting processed soon enough, causing the timer to keep ticking too long even after the echo signal has been received. Unfortunately, due to this complication, GPS module functionality is taken out of the main program. A simple solution to this problem would be to have a dedicated processor for the GPS module.

3. ANALYSIS

3.1. Challenges

The initial challenge encountered was reverse engineering and becoming familiar with the Ion-Voyager electric scooter as there was no accessible documentation online regarding technical modules. This process included a complete tear down of the electric scooter to locate the motor-controller, motor, and all necessary control signals. Although the desired safety features were selected prior completing this step, further development of these features were gated by the completion of the reverse engineering phase. It was determined to be most feasible to tap into control signals upstream of the microcontroller in the handlebar region. The development process continued after creating a wiring documentation including each connector on the scooter.

The hardware design encountered a very late bug in the downhill throttle speed limit control circuit during the system integration phase. The original design utilized an op-amp / BJT circuit, shown in figure 12, to select the smaller voltage of the two inputs, the original throttle voltage or Vref. This circuit was verified on the bench when Vthrottle was simulated with the

bench power supply. However, the circuit was providing incorrect outputs when it was integrated to the system.

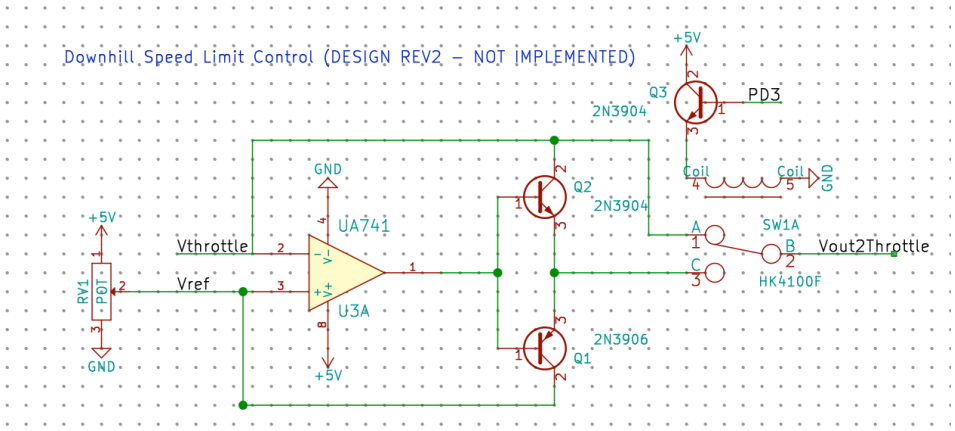


Figure 12: Rev2 Downhill Speed Limit Control Circuit

A software workaround was introduced to compare the input throttle voltage and Vref. The op-amp / BJT circuit was replaced by an additional HK4100F relay, figure 13. Relay K1 is used to switch from the default Vin_Throttle to V_Ref when Vin_Throttle > V_Ref in order to limit the speed. The downstream relay K2 is used to switch from the default Vin_Throttle to the output of K1 when the scooter is going down a hill.

This new design once again was verified on the bench with the power supply and oscilloscope. However, when supplied by the scooter battery, the relays were not switching properly. It was discovered that there was not sufficient current in the circuit to switch both relays at the same time. As a result, the 2N3904 was replaced by the 2N2222 NPN transistor which has a higher Ic current, and the Downhill Speed Limit Control Circuit bug was resolved.

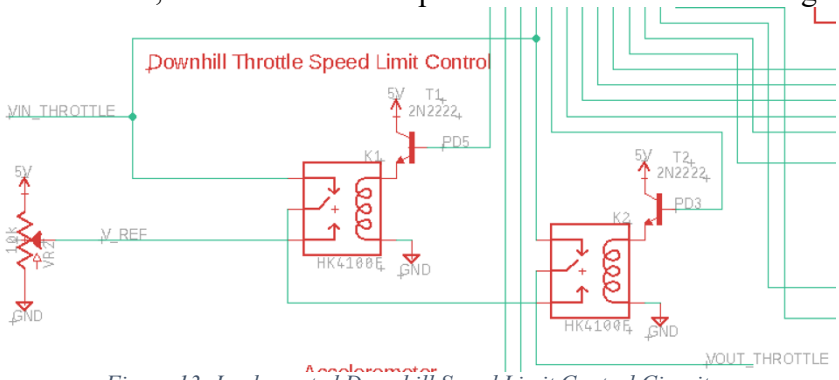


Figure 13: Implemented Downhill Speed Limit Control Circuit

3.2. Cost

The final development cost of the Google Go totals $377.40. Since the Google Go is not positioned to be a consumer electric scooter there is no explicit retail price point. Instead, it will operate in a fleet of sharable last mile electric scooters. Competitors utilize a $1.00 unlock fee followed by $0.15/min. The Google Go will be waiving the unlock fee and maintain the $0.15/min fee as it obtains market share. As a result, each device will require ~41.9 hours of ride time in order to reach its breakeven point. The Google Go will also be monetizing location data via Google Maps. However, the revenue generated from monetizing location data is not reflected in the calculated breakeven point.

Part	Vendor	Device	Description	Cost
C1	Jameco	15229	0.01uF Ceramic Capacitor	$0.12
IC1		74LS125	Quad bus BUFFER, 3-state	$0.84
J1	Adafruit	FSR402	Force Sensative Resistor	$7.00
J2	Adafruit	FSR402	Force Sensative Resistor	$7.00
K1	Hui-Ke	HK4100F	SPDT Electromechanical Relay	$0.99
K2	Hui-Ke	HK4100F	SPDT Electromechanical Relay	$0.99
M1	Adafruit	ADA1201	Vibrating Motor Disk	$1.95
M2	Adafruit	ADA1201	Vibrating Motor Disk	$1.95
QC1	Digikey	SER1727	7.3728 MHz DIP Oscillator, 14-in DIP size	$2.21
T1		2N2222	NPN TRANSISTOR, TO-92	$2.23
T2		2N2222	NPN TRANSISTOR, TO-92	$2.23
U$1	Sparkfun	HC-SR04	Ultrasonic Ranging Module HC-SR04	$3.95
U$2	Sparkfun	HC-SR04	Ultrasonic Ranging Module HC-SR04	$3.95
U1		ATMEGA328P	8-bit AVR Microcontroller	$2.14
U3	Adafruit	ALS-PT19	Analog Light Sensor	$2.50
U4	Adafruit	LIS3DH	3-Axis SPI/I2C Accelerometer	$4.95
VR1	Adafruit	10k POTENTIOMETER-PTH-9MM-1/20W-20%-	Potentiometer (Pot)	$1.25
VR2	Adafruit	10k POTENTIOMETER-PTH-9MM-1/20W-20%-	Potentiometer (Pot)	$1.25
	Walmart	571538531	Ion Voyager Electric Scooter	$250.00
*	Adafruit	746	Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates - Version 3	$39.95
*	Adafruit	2090	2.8" TFT LCD with Cap Touch Breakout Board w/MicroSD Socket	$39.95
			Total Price	$377.40
	* Parts included in concept but not implemented			

Figure 14: Controls Circuit BOM and Price

3.3. Future Improvements

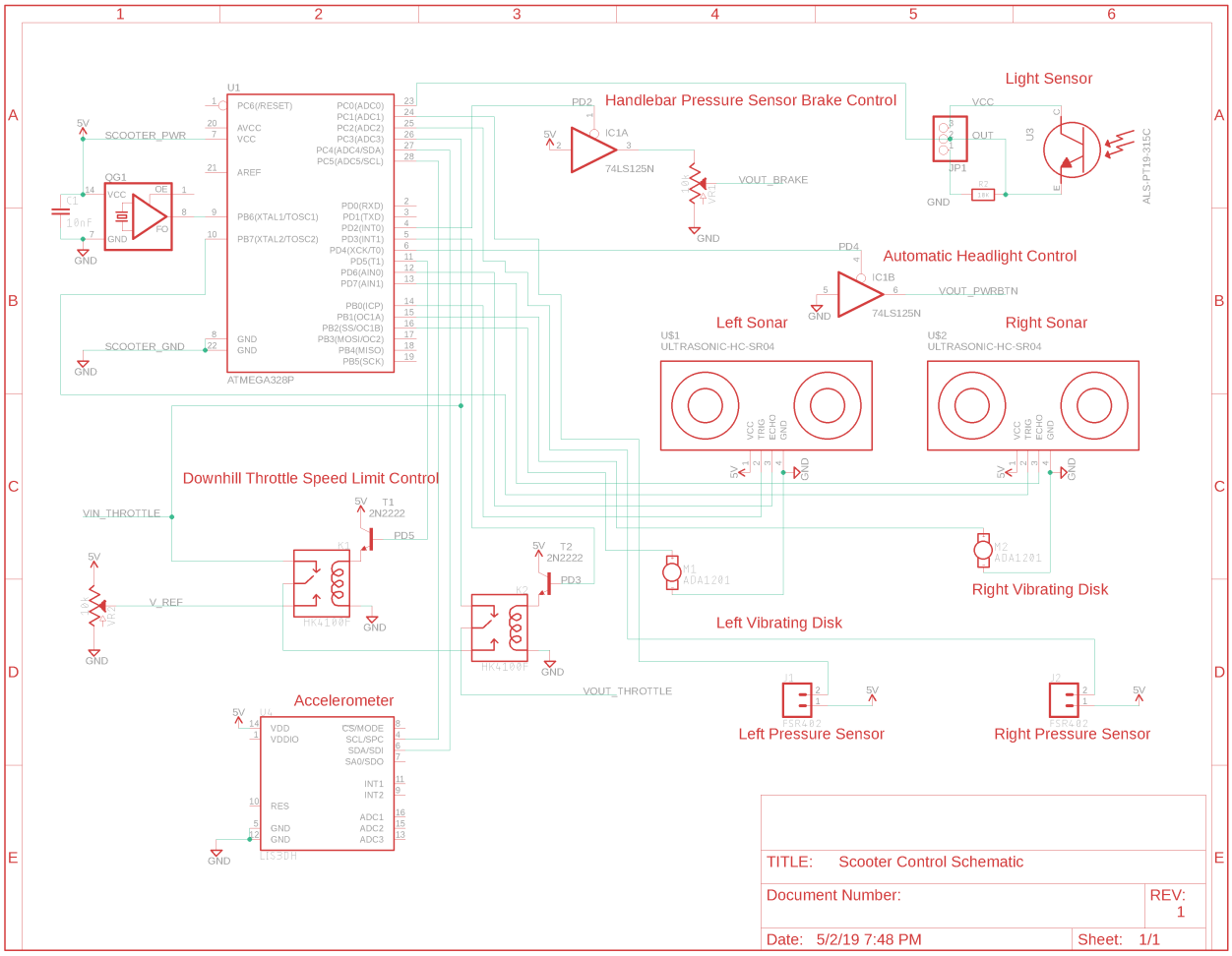
One possible future improvement to the scooter features is the integration of GPS and LCD to display rider's current location to the map on the screen. One of the safety hazards come from a rider looking into his or her phone to navigate where to go, not being able to have both hands on the scooter. In order to mitigate this issue, an installed LCD screen can utilize the GPS module (and Google Maps further down the line) to display where the rider needs to go, eliminating one reason for riders not to have both their hands on the handle bars. While GPS parser is functional, the team has decided that it would be better to have its own, dedicated processing unit so that it does not interfere with other sensing units while also being able to get more accurate data. Furthermore, the same processor can also interface with an LCD, providing the most up-to-date information retrieved from the GPS module to the screen.

4. CONCLUSION

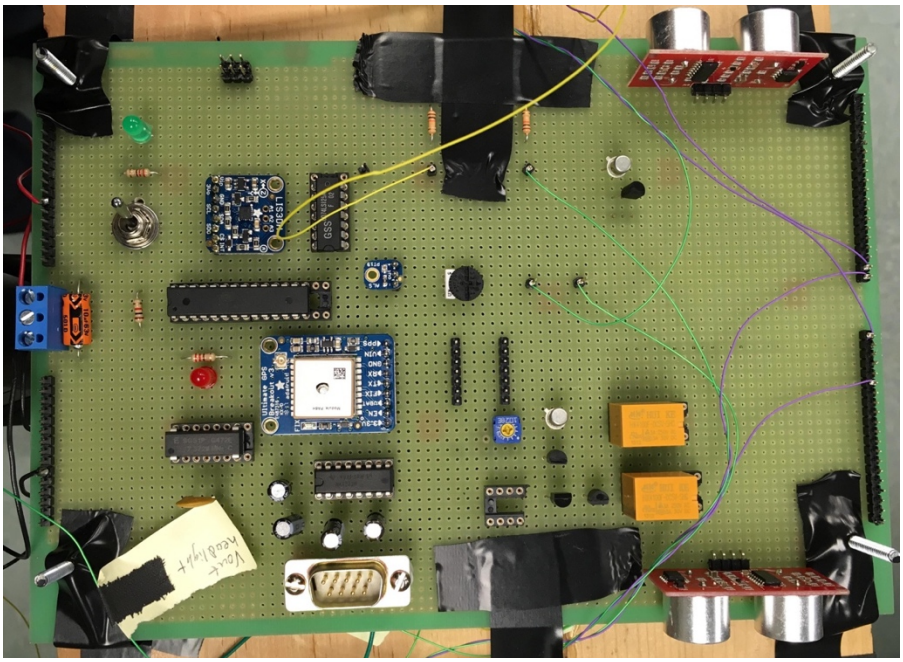
We were successful in creating a module that enhanced the safety of the electric scooter. With last minute changes to our designs, we were able to ride and test all the safety functions of the scooter. The safety features and competitive pricing of the Google Go is predicted to help the service overrun the competitors. Moreover, the cost of manufacturing the module and the scooter will most likely fall once production begins. In addition to charging for the use of the service, we will also be selling the GPS data of the user which will help bring the new product to profit margins quicker. Google Go will be the safest electric scooter renting service at the launch of the service.

5. APPENDICES

Appendix A: Scooter Controls Schematic



Appendix B: Dev Board



Appendix C: Proof of Concept Product

Appendix D: Signature Sheet

	Samuel Bruner	Dae Kim	Juan P. Mejia	Hyun-Bum Yang
System design	25	25	25	25
Hardware design	65	35	0	0
Hardware assembly	30	30	20	20
Hardware debugging	50	10	20	20
Microcontroller software design	0	0	0	100
Microcontroller software debugging	0	0	0	100
System integration	25	25	25	25
Documentation	100	0	0	0
Project report (oral)	30	10	30	30
Project report (written)	40	10	20	30

Samuel Bruner _____

Dae Kim _____

Juan P. Mejia _____

Hyun-Bum Yang _____