THE SMART BACKPACK

By

Troy Chmieleski

Kevin Noh

Rajat Sharma

Final Report for ECE 445, Senior Design, Spring 2015

TA: Drew Handler

06 May 2015

Project No. 46

Abstract

Our senior design project was aimed towards designing and implementing a “Smart Backpack” that would be capable of measuring its own its weight. Its primary function was to measure and monitor its own weight using a smartphone applicate via Bluetooth LE. It also included secondary features such as an integrated mobile charger, rechargeable battery, and a proximity sensor. This paper provides an overview of the entire design process and it will hopefully aid future collaborators in improving the design of our Smart Backpack.

Contents

1. Introduction

1.1 Motivation

Our senior design project proposed to solve an on-going health problem that has been affecting many people across the globe. Numerous studies have concluded that the effect of carrying a heavy backpack can lead to various health issues, especially with younger children. A research conducted in University of California in Riverside published a study stating that 90% of students said they had experienced “bad” or “very bad” back pain due to heavy backpacks at some point in their school lives. References [1] – [3] all point out different health issues caused due to excessive backpack weights. These included permanent curvature of the spine (scoliosis), involuntary muscle contractions, and chronic lower back pain that may last up to years. Moreover, studies in many different countries, including Spain, Germany, and Japan have published similar research results concluding that heavy backpack has been leading to various health issues. Through these studies, we were able to verify that the heavy backpacking issue was indeed a global health issue.

Experts advise everyone should keep their backpack loads to no more than 10% of his or her body weight. Reference [4] shows us that loads of 20% of the body weight can start causing serious back issues. To propose a solution to these issues, our group implemented the “Smart Backpack.”

1.2 Objectives

Our main objective of the project was to provide a solution to overcome the health issues students were having with carrying heavy backpacks. We proposed a design of a smart backpack which allowed the user to monitor the weight of their backpack wirelessly through an iPhone application. This would allow users to reduce backpack related injuries by continuously monitoring of the backpack weight. Moreover, the goal was to design the phone application and the backpack to help motivate users to adopt healthier habits.

Moreover, we implemented additional features in our backpack to provide a better user experience. While the main features were built in digital scale and the iPhone application, our secondary features of the design included functions such as an integrated mobile charger, rechargeable battery, and proximity sensing features. The proximity sensing feature notified the users whenever the user walked away from the backpack, to prevent misplacement of the backpack. With the iPhone application at the center of all these features, it truly had the features to provide a “smart” backpack experience.

1.3 Block Diagram

This subsection provides a brief introduction of the basic block diagram, and the functionality and roles of these components. The block diagram of the Smart Backpack is presented in Figure 1.

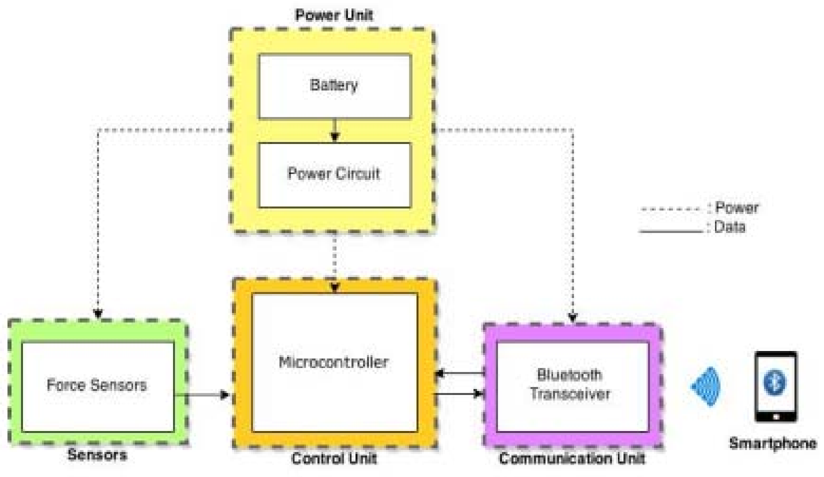


Figure 1. Block Diagram of the Smart Backpack

The five components in our design allowed us to efficiently implement our design. The power unit is the central power unit that supplies the correct amount of power to each of the individual components. This unit also included the battery charging circuit to charge the battery, and a USB charging circuit to provide on-the-go USB charging. The sensor unit was the unit that included the digital weighing scale. The control unit and the communication unit was the core of the design which allowed the backpack to take the sensor unit’s voltage reading, convert it to a weight, and wirelessly transmit it to an iPhone via Bluetooth LE, where it provided the backpack weight in a user-friendly application setting. The detailed design process and overall integration can be found in Section 2. Integration of all the blocks into a PCB is presented in Figure 2.

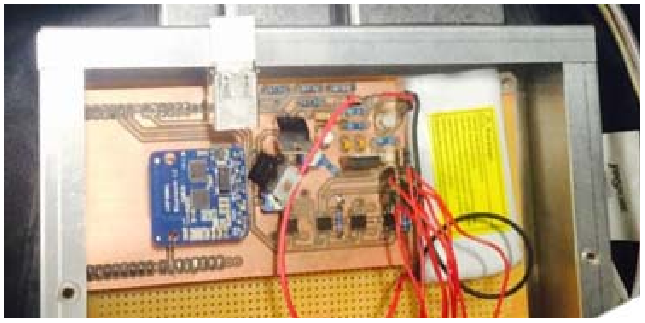


Figure 2. PCB of the Smart Backpack

2 Design

2.1 Power Unit

A 7.4 V, 2200 mAH battery was used to power the entire circuit. The strain gauge weight sensors were provided an excitation voltage of 2.3 V by passing the 7.4 V battery through a 5 V regulator that was in turn connected to a potentiometer that dropped the voltage down to 2.3 V. The 5 V regulator was used to provide a steady voltage, since the voltage of the battery could potentially drop down to 6.5 V. The inclusion of the regulator ensured a constant excitation voltage to the weight sensors. The microcontroller was provided 7.4 V, which was used to provide 5 V to the Bluetooth module.

2.1.1 Battery Charging Circuit

This circuit was designed to recharge the 7.4 V li-ion battery. The schematic of the circuit used to do so is shown in Figure 3 below.

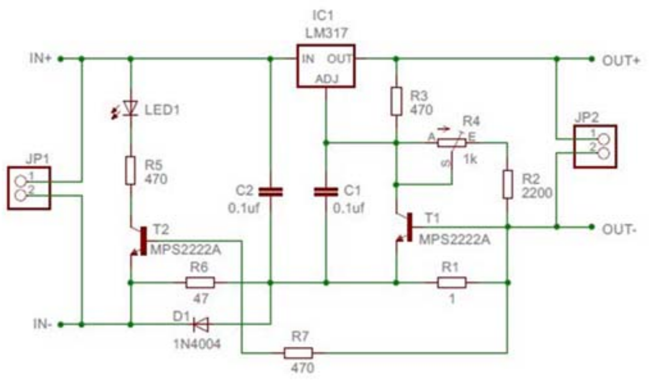


Figure 3 Battery Charging Circuit Schematic

The LM317 is a voltage regulator that determined the amount of voltage that would be delivered to the output, JP2. The input, JP1, was connected to a 12 V DC voltage supply to power the charging circuit. The voltage regulator and resistors R2, R3, and R4 were used to create a voltage regulator through voltage division. The voltage divider created across R2 and R4 multiply this voltage to get the desired value of 7.4 V at the output, JP2. Transistors, T1 and R1, were used to set the current limit. The current drawn by the battery passes through R1. As the voltage across R1 approaches 0.65 V, T1 begins to turn on. T1 then draws current from the voltage divider. Consequently, U1 reduces the voltage at the output

The circuit was designed such that a constant current of 0.7 C was provided until 3.2 V/cell is reached. Then constant voltage was used until the current dropped to 0.1 C. At that time, the charging would stop. However, it does not turn off the charge entirely. Testing has shown that the current drops to almost zero anyway.

The circuit simplifies this by limiting the charge voltage to 7.4 V. When the battery reaches 7.4 V, it will no longer draw current. The charger is also current-limited. Below 75% of charge, the limit current is reached. After about 80% charge, the current starts to decay toward 0 A. At about 95% charge the current drops to only a few milliamps. In theory, the battery will never finish the charge, since the closer it gets to the threshold, the less current it draws. If the charger is left on for 2 hours or so, it will reach near 100% charge. But, 95% can be reached in less than an hour in most cases. The battery's charging curves using this method are shown in Figure 4.

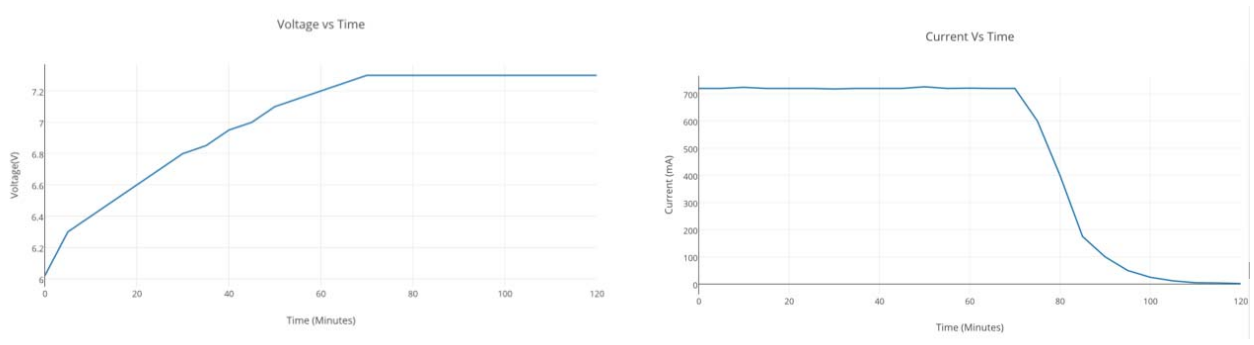


Figure 4 Battery Charging Curve

2.1.2 USB Charging Circuit

The purpose of this circuit was to charge an IPhone 5s or IPhone 6 via USB. In order to do so, we used a female USB port. An image of the port is shown in Figure 6. Our initial plan was to provide 5V to V_{cc} and 0 V to ground. However, we found there was a problem with this plan. Apple has a pre – defined set of requirements for charging Apple products over USB. These requirements allow Apple devices to trust a charging source. Consequently, we had to alter our design. The schematic for our final USB charging circuit is shown in Figure 6.

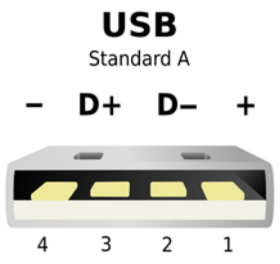


Figure 5 USB Pin Layout

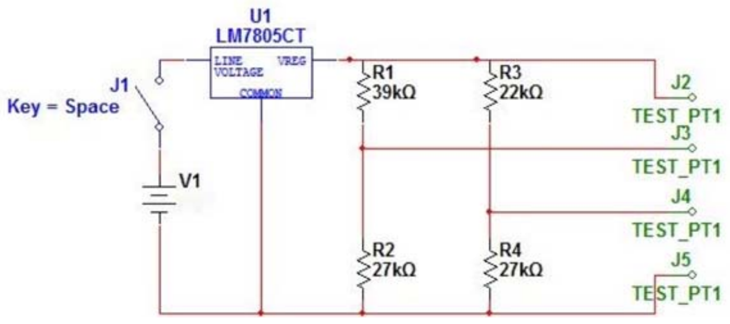


Figure 6 USB Charging Circuit Schematic

The output pins in Figure 6 and Figure 7 were connected such that J2 was connected to Pin 1, J3 to Pin 2, J4 to Pin 3, and J5 to Pin 4. The LM7805 5 V voltage converter was supplied with 7.4 V using the battery and it provided 5 V to J2. The two voltage dividers across the resistors provided 2.7 V to Pin 2 (D-), and 2.3 V to pin3 (D+). Pin4 was connected to ground.

2.2 Sensor Unit

2.2.1 Initial Approach

The sensor unit is the unit responsible for taking a measurement of the weight of the backpack and sending the information to the control unit. Thus, we needed a reliable sensing unit that was able to detect the weight of the backpack accurately. Our initial approach in tackling this problem was through utilizing piezo-resistive force sensor in between the shoulder strap and the shoulders. We tried come up with a mathematical model that was able to predict the weight of the backpack with a measurement of the force applied on the backpack. Unfortunately, extensive calculations and measurements showed that this model was not a promising model of measuring the weight of a backpack accurately. Results of the force measurements and the weight did not have a clear relationship. The picture of the initial design and an example measurement of the initial design is shown in Figure 7 and Figure 8, respectively. Also, the initial circuit schematic is shown in Appendix B.

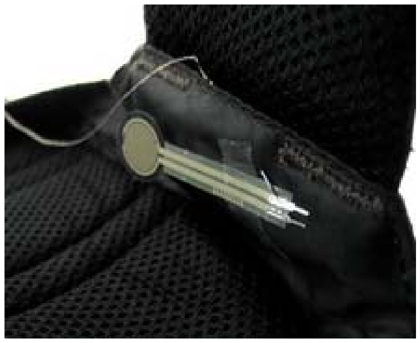


Figure 7 Initial Sensor Design

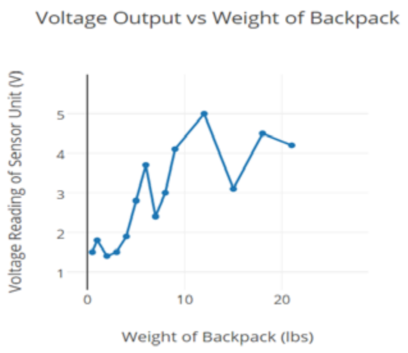


Figure 8 Measurement Data of Initial Design

2.2.2 Strain Gauges

Due to the inaccurate nature of piezo-resistive force sensors, we changed our design of the sensor unit by utilizing strain gauges to measure the weight of the backpack. Strain gauges are devices which have resistances proportional to the amount of strain applied in the device. It is modeled in a form of Wheatstone-Bridge which is a very effective way to measure minute changes in resistances. An electrical model of the strain gauge is shown in Figure 9.

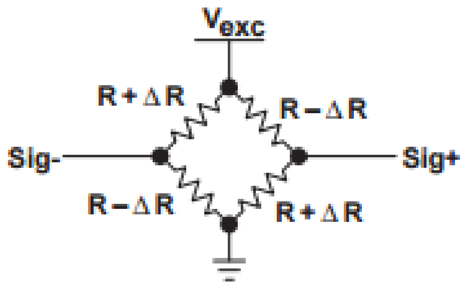


Figure 9 Instrumental Op-Amp Configuration

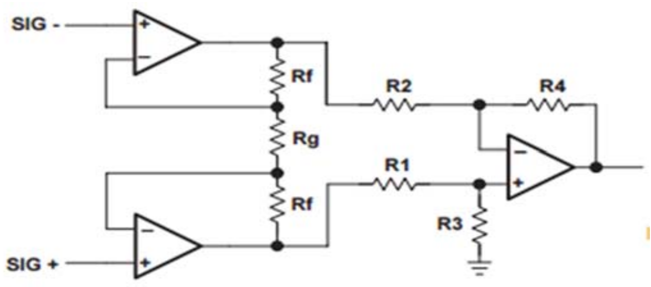


Figure 10 Strain Gauge Electrical Model

As we see in Figure 9, when an excitation voltage is applied to the V_{exc} terminal, the strain gauge outputs a voltage proportional to the strain applied between the two ends of the strain gauge. However, the output of these strain gauges are in the range of millivolts. Thus, in order to have the microcontroller read and differentiate the voltages, we needed to amplify the signal by a factor of 5,000 for an ideal mapping of 0 – 30 lbs to 0 – 5 V.

In order to amplify the signal with the most accuracy, we implemented a three op amp instrumental amplifier configuration to overcome challenges we faced with impedance mismatches between the terminals. The instrumental op-amp configuration is shown in Figure 10. This configuration allowed us to control the gain of the amplifier using an external resistor that was needed for the individual strain gauges. Finally, we had a linear model between the voltage reading and the weight of the backpack as shown in Figure 12.

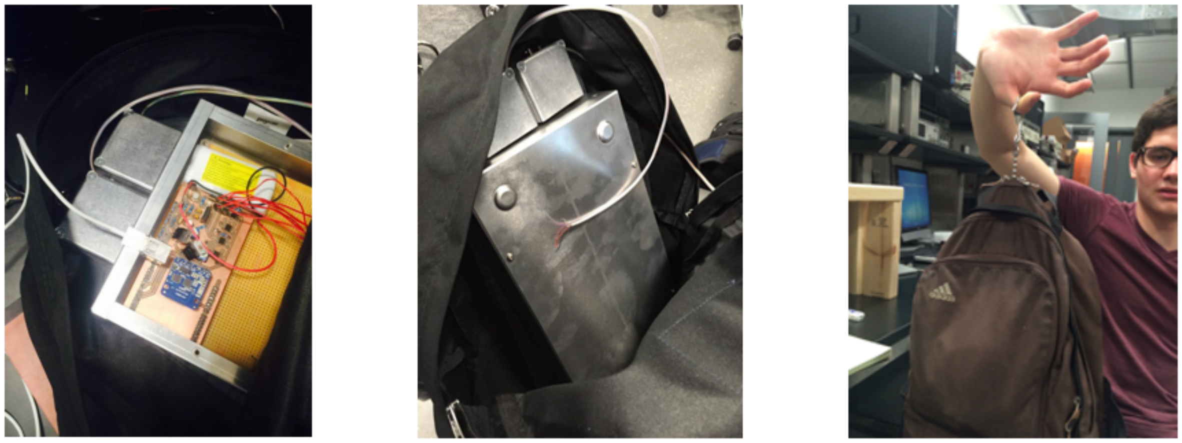


Figure 11 Integration of Strain Gauge with Backpack

The final integration of strain gauge into the backpack is shown in Figure 11. First, enclosures were create to protect the fragile strain gauges and the PCB. Then, these enclosures were secured to the backpack with screws, with the strain gauges coming out through the top strap of the backpack. We show a demonstration of how the backpack is weighed on the left of Figure 11.

As a final note for the sensor unit, there were extensive measurements and physical calculations to determine an accurate model with the strain gauges mounted on the shoulder strap so that the user can keep track of the weight at all time he or she is wearing the backpack. However, there were too much variation due to the strain in the lower part of the shoulder strap and the force applied towards the back. Thus, to focus on achieving an accurate measurement model, we designed with the strain gauges on the top strap as shown in Figure 11.

2.3 Control Unit

The control unit consists of an Arduino Uno that serves as the brains behind the smart backpack. The code written for the microcontroller serves as the glue to make all of the components work together correctly. Since the control unit is a key component of our project, we designed it to be robust and closely follow the necessary design requirements.

To function correctly, the control unit must be designed to accept an input voltage of 7.4 V +/- 0.3 V. The microcontroller receives its input voltage from our power supply which has an output voltage within the required range. Too much input voltage can do damage to the microcontroller. The microcontroller can handle an input voltage in the range of 7 - 12 V. Our requirements for the input voltage keep the microcontroller functioning within the recommended input voltage range.

The smart backpack weight detecting mechanism relies on the data being sent from the strain gauges that are mounted inside the backpack. In order to ensure that the strain gauge data is sent properly to the microcontroller, it is imperative that the analog input to the microcontroller be in the range of 0 - 5 V. The output voltage from the strain gauges when under strain is in the mV range. Using our amplification circuit, we are able to amplify the incoming analog voltage output between the recommended range of 0 - 5 V.

One thing that we noticed when designing the amplification circuit is that our amplifiers were more accurate when powered with a supply voltage of 7.4 V coming from our power supply. While a supply voltage of 5 V to the instrumentation amplifiers ensured that the amplified analog voltage output would never go beyond the 5 V upper bound, we noticed that the output swing from the instrumentation amplifier caused the max voltage to be lower than we expected, limiting us from accessing the full range of the 0 - 5 V output.

The datasheet for our microcontroller shows that while the recommended analog input voltage range is 0 - 5 V, the microcontroller can actually accept up to 7.5 V without issue. Taking into account the output voltage swing from the instrumentation amplifiers, our max voltage is ~7.0 V +/- .3 V under a load of 50 lbs. With our recommended max load of 30 lbs for the smart backpack, the max analog voltage is ~5 V +/- .3 V. This is within the recommended range of the input voltage for the analog pins on the microcontroller and ensures that the control unit continues to function under normal operation without too much additional stress on the analog pins.

The microcontroller is programmed to parse the analog data coming from the strain gauges. The analog data coming from the strain gauges is sent to the microcontroller using the A0 and A1 analog pins on the Arduino Uno. We accomplish collecting data from the strain gauges by continuously monitoring the output from the strain gauges. When designing an early prototype, we noticed that constantly monitoring the strain gauge data was wasting a lot of cycles on the microcontroller's CPU. We realized that monitoring the strain gauge output was only really necessary when the user was actively viewing the weight data from the companion iPhone app. By adding a weight button to the app, we can tell the microcontroller when exactly to monitor the incoming data from the strain gauges. Since continuously monitoring the output from the strain gauges is expensive, reducing the amount of time spent parsing force data enables the microcontroller to perform its tasks without being put under excessive load.

When working with the microcontroller we noticed that it operates on a single run loop. This is the run loop that we used to monitor both the incoming and outgoing data from the strain gauges and the Bluetooth module. Our initial prototype monitored incoming force sensor data every 200 ms by setting delay on the main run loop. We noticed quickly that this worked poorly since we would not be able to effectively monitor both the data coming from the strain gauges and the Bluetooth module continuously if we set a delay in either or both of the processes. Simply put, any added delay to get a process to run at a set interval caused the program to effectively come to a halt. This made it impossible to continue to receive updates from the strain gauges and the Bluetooth module in a timely manner.

To address this issue, we looked into setting up a multithreaded environment on our microcontroller. While multithreaded environments work well when developing the iPhone app in Swift, the microcontroller is not well suited for a multithreaded environment. The overhead required to implement continuous monitoring for the strain gauges and the Bluetooth module using a multithreaded environment was simply not worth it in this case.

We found that a much better solution to get multiple monitoring components to work together in an effectively continuous manner was to perform checks to see whether or not a monitoring component should be run every time the run loop was executed. These checks keep track of the monitoring intervals, the current time and the previous time all in ms in order to determine when monitoring should be performed. If the current time - previous time >= interval then the desired interval has been passed and the monitoring update should be performed. This approach is much more effective than adding delay since an integer comparison is fast and cheap.

When handling incoming events from the Bluetooth module with the microcontroller we initially set up monitoring using a polling technique in conjunction with the C++ and Arduino API provided by Adafruit for the Bluefruit LE module to interact with the nRF8001 IC. This polling technique checked whether or not any data was available from the Bluetooth module. We noticed that a more effective approach was to set up a callback structure instead. While events were still monitored via polling, the incoming data was harder to parse and package when trying to send the data over Bluetooth. With callbacks, we are able to hook into the ACI events and the data available over the RX channel of the nRF8001 Bluetooth IC.

The event callbacks are received whenever Bluetooth advertising, connection or disconnection events are broadcasted. We can then respond to those events with the microcontroller as needed. The data sent over the RX channel is parsed in the RX callback by checking for triggers sent over Bluetooth. Communication triggers are explained further in the communication unit design description below. When the microcontroller receives either the start or stop measuring weight data trigger via the callback, it is then able to begin parsing, processing and packaging up the weight data to be sent over Bluetooth to the iPhone app.

A core responsibility of the microcontroller is to map the analog input from the strain gauges to a signal that can be processed digitally. Once the output voltages from the strain gauges have been amplified to a range of 0 - 5 V using our amplification circuit, this data can be utilized and processed by our microcontroller. We collected experimental data from our strain gauges to map the relationship between output voltage and weight. From the plot below, we can see that the output voltage from one of the strain gauges post amplification is in the require 0 - 5 V range and that the relationship between output voltage and weight post-amplification is approximately linear.

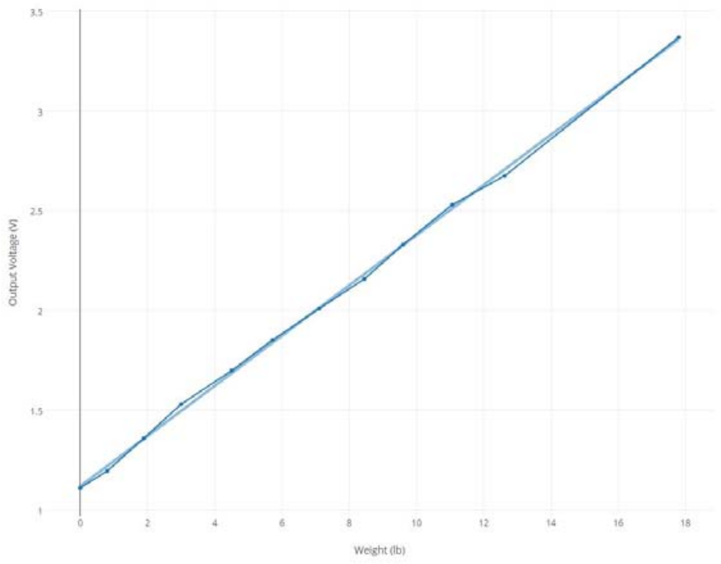


Figure 12 Voltage vs Weight Post Amplification

We used this linear relationship to create a mapping algorithm on the microcontroller side that utilized the experimental strain gauge data that we collected. To linearize the dataset, we found the line of best fit for the output voltage to weight relationship for each of the strain gauges.

Using the output of the microcontroller's A/D converter we can take the digital output of the strain gauges which is an integer value in the range of 0 - 1023 and use that in combination with our lines of best fit to generate a mapping to the weight of the smart backpack in pounds.

To convert the digital signals in the range of 0 - 1023 back to volts, we can take our upper bound of 5 volts and multiply that by the digital signal over the maximum digital signal of 1023. We can then rearrange our lines of best fit to solve for the weight coming from each strain gauge.

For detailed analysis of the weight data on the iPhone we also send over the voltage readings from each of the strain gauges. To map the digital signal to a voltage value, we use the map function to map the force sensor digital reading which is in the range of 0 - 1023 to the output voltage range from 0 - 5000 mV. We divide by 1000 mV in order to show the output voltage in V on the iPhone app.

We can see the final equations used to calculate the weight of the backpack in pounds as well as the equations used to compute the voltages from each of the strain gauges below. For the equations below, let *fsr1Lbs*, *fsr2Lbs* = weight of strain gauges (LBS), let *fsr1Reading*, *fsr2Reading* = digital output of strain gauges (0 – 1023) and Let *fsr1Voltage*, *fsr2Voltage* = analog output voltage of strain gauges (V).

$$fsr1Lbs = \frac{\frac{5.0 \cdot fsr1\mathrm{Re}ading}{1023.0} - 1.082}{.04646}$$

$$fsr2Lbs = \frac{\frac{5.0 \cdot fsr2Reading}{1023.0} - 1.086}{.1083}$$

$$fsr1Voltage = map(fsr1\mathrm{Re}ading, 0, 1023, 5000) / 1000.0$$
$$fsr2Voltage = map(fsr2\mathrm{Re}ading, 0, 1023, 0, 5000) / 1000.0$$

One thing to note about the above equations is that the equations for the lines of best fit for each of the strain gauges are quite different. This reveals that while the strain gauges we used for the smart backpack were the same, the output voltages from the Wheatstone-Bridges on the strain gauges were inherently different.

In order to address the issue of strain gauges with inherently different characteristics, we had to calibrate each of the force sensors individually. By calibrating each strain gauge individually, we could get an accurate weight reading on each of the force sensors. The total weight of the backpack is computed as the sum of the weights calculated from the individual strain gauges.

2.4 Communication Unit

The communication unit consists of the Adafruit Bluefruit LE Bluetooth module. The Bluetooth module serves as the communication unit by transmitting the weight data collected by the strain gauges to the iPhone app and receiving incoming events from the app and relaying that data back to the communication unit.

The Bluetooth module utilizes the UART protocol to accomplish transferring and receiving data up to a distance of 1 m to and from the iPhone. The UART protocol works on the nRF8001 by means of sending out packets of data 20 bytes at a time. When sending data larger than 20 bytes, the data must be packetized into chunks of 20 bytes. Attempting to send a buffer larger than 20 bytes results in undetermined behavior oftentimes noticeable by means of buffer overflow or the data simply being cut off.

To avoid overflow, the control unit carefully adheres to the UART protocol by means of the Adafruit Bluetooth C++ and Arduino API in conjunction with custom API for handling the parsing and packaging of data being transmitted to and from the Bluetooth module.

When receiving incoming data, the 20 byte packetized chunks coming from the RX channel on the nRF8001 are placed into a buffer and parsed for recognizable triggers. Parsable triggers are used by the control unit to figure out what action should be performed in response to data received and transmitted via the Bluetooth module. The triggers used by the control unit can be seen in the trigger table below.

Table 1 Triggers

Trigger	Meaning
!STRTWD	Start weight data
!STOPWD	Stop weight data
!WDT	Weight data total
!WD1	Weight data first strain gauge
!WD2	Weight data second strain gauge
!VD1	Voltage data first strain gauge
!VD2	Voltage data second strain gauge

When interfacing the Bluetooth module with the microcontroller, we noticed that the Bluetooth module requires continuous monitoring in order to be informed about incoming Bluetooth data. We also noticed that packaging up data into 20 byte chunks can be quite tricky since we must watch out for buffer overflows. The packaging process itself can also be time consuming. In order to effectively pack up triggers and the respective data we want to transfer when sending strain gauge data, we set up a 600 ms timer. This interval is long enough to allow for the packaging process to send over buffers with the trigger name and the associated trigger data without being overwritten by the next execution of the run loop. This allows us to send over the triggers and corresponding values for starting and stopping weight data collection and individual strain gauge and total weight and voltage data without running into timing issues.

The limited range of the Bluetooth chip also affected our design decisions. The upper range of the nRF8001 is about10 m. We ended up using this upper bound when implementing our loss prevention feature on the iPhone app side for when the backpack is out of range of the iPhone.

We also noticed that constantly polling for Bluetooth events uses a lot of power. In order to reduce the amount of power we realized that we should only open the communication channel when needed. We achieved this by implementing a means for the Bluetooth module to start and stop transmitting weight data. This enabled our design to be more power efficient and consume less CPU cycles on the microcontroller when processing data to be sent over Bluetooth.

2.5 iPhone App

The iPhone app serves as the front end portion of the smart backpack. The iPhone app is required to display the weight of the backpack as well as receive a proximity alert when the backpack is ~20 m (30 steps) away from the iPhone.

To achieve these requirements, we wrote an iOS app in Swift that would interpret incoming Bluetooth data and display the weight and voltage data on the iPhone in real time. To parse and process data on the iOS side, we utilized the Adafruit Bluetooth Swift API as well as Core Bluetooth to receive incoming data from the Bluetooth module.

When writing the iOS app in Swift we realized that since Swift is a relatively new language it can change rapidly. As a result the Swift compiler can be buggy. We ran into issues where compiling would result in segmentation faults but multiple times in a row would surprisingly fix the issue.

We also had to make sure to distribute Bluetooth events properly to individual view controllers so we had to set up a distribution center for Bluetooth events to make sure that that each portion of our app was receiving the correct data.

In order to make the app responsive, we took advantage of processing incoming Bluetooth data on a background thread and then updating the UI on the main thread. This way incoming updates to the iOS app are immediately available to the user.

3. Design Verification

3.1 Power Unit

3.1.1 Power Supply
The output of the battery was measured using a multimeter to ensure that the output of the voltage was 7.4 V ± 0.3 V. The 5 V voltage regulators were tested using an oscilloscope for any ripple voltages. If the voltage had increased due to some reason, the outputs from the sensors could vary greatly. Additionally, the battery was tested for reverse currents using a diode to ensure proper functionality of the current flow. The diode was placed in series with a resistor and current was supplied to the negative side of the diode. Then the current flowing through the resistor was measured on the oscilloscope. It was found that no current was flowing through it.

3.1.2 Battery Charger
The output voltage of the battery charging circuit was measured using a multimeter to ensure that the charging limit was set to 7.4 V, since a higher voltage could potentially damage the battery. Additionally, an LED is connected to the charging circuit such that the LED was ON, whe the battery was charging and it would turn OFF when the charging was complete. This allowed us to keep track of the charging cycle duration, and prevented us from overcharging the battery. The charging characteristics plots in Figure 4 allowed us to confirm the functionality of the circuit because the voltage and current curves were what we had expected them to be.

3.1.3 USB Charger
The inputs to the USB charger pins were measured using a multimeter. By doing so, we were able to confirm that the voltage at V_{cc}, D- , D+, and Ground were 5 V, 2.7 V, 2.3 V and 0 V. This process allowed us to confirm that our USB charging port met the requirements set by Apple for charging Apple products over USB.

3.2 Sensor Unit
The excitation voltage of the weight sensors were designed to operate at 2.3V ± 0.1V. A multimeter was used to measure this voltage and verify its values with this excitation voltage. In order to ensure that the 2.3 V input did not change over time, the battery was connected to a 5 V regulator that provided a steady voltage. This 5V input was dropped down to 2.3 V using a potentiometer. The addition of the 5 V convertor allowed us to ensure a constant 2.3 V to the sensors with ripples maximum of roughly 50 mV.

First, the outputs of the first and second sensors were verified to be 0.22 ± 0.02mV and 0.44 ± 0.02mV using a multimeter when measured in rest. Second, the first amplified output of the first sensor and the first amplified output of the second sensor were measured to be 0.22V ± 0.4V and 0.18 ± 0.4V (respectively) using a multimeter. This stage of amplification had a gain of 1,000. Finally, the second amplified output of the first sensor and the second amplified output of the second sensor were measured to be 1.1V ± 0.4V and 1.1V ± 0.4V (respectively) using a multimeter. This stage had a gain of 5.

In order to test the sensors were functioning correctly, we created a plot that measured the output voltage of the sensors versus weight. A linear plot shown in Figure 12 demonstrated the correct functionality of the sensors.

3.3 Control Unit

3.3.1 Interface with Sensors

Multiple control packets for initialization, registration, verification, and deletion were sent from the sensors to the microcontroller. The reception of those packets confirmed the connection to the sensors.

3.3.2 Serial Communication at 9600bps

Data packets were sent to and from the microcontroller and a smartphone via Bluetooth. Multiple packets of 1 kb were transmitted and checked for completion and transfer rates. Perfect package transmission at 9600 bps was achieved.

3.4 Communication Unit

3.4.1 Connection with Smartphone and Microcontroller

Multiple control packets for initialization, registration, verification, and deletion were sent from the sensors to the smartphone over Bluetooth using UART. The Bluetooth module was programmed to echo its received data back to the microcontroller. Comparing the data sent to the data received allowed us to confirm the connection between the Bluetooth module and the microcontroller.

A similar test was conducted between the Bluetooth module and the smartphone. Comparing the data sent by the microcontroller over Bluetooth to the smartphone with data received by the smartphone allowed us to verify the functionality of the Bluetooth module.

3.4.2 Distance Check

We wanted the smartphone to work within a radius of 10 m from the backpack. Taking the smartphone to a distance greater than 10 m allowed us to determine that the Bluetooth module was able to transmit data packets in a stable manner up to 20 m.

3.5 Smartphone

3.5.1 Weight Display

The smartphone displayed the weight of the backpack. We measured the weight of the backpack using a digital scale and compared that weight to the weight displayed on the smartphone. This test demonstrated that the weight was within ± 0.5 lbs of the actual weight.

3.5.2 Proximity Sensor

In order to test that the proximity sensor was displaying an alert on the smartphone at 20 m ± 4 m, the phone was taken more than 20 m away from the backpack. We found that the smartphone was correctly displaying the alert at a distance greater than 20 m ± 4m. This test allowed us to determine that the proximity sensor was working correctly.

4. Costs

4.1 Parts

Table 2 Parts Costs

Part	Manufacturer	Retail cost ($)	Quantity	Total Cost ($)
7.4 Lithium-Ion battery	Tenergy	13.99	1	13.99
LM317 Voltage regulator	Fairchild Semiconductor	5.00	4	20.00
USB Female Port	Sparkfun Electronics	4.00	4	16.00
Strain gauge weight sensors	Omega	22.99	4	91.96
Digital weighing scale	Air Weigh	14.99	2	29.28
nRF8001 Bluetooth LE	Adafruit	19.95	1	19.95
Arduino Uno	Arduino	24.95	1	24.95
LM7805 5V converter	ECE store	1.50	4	6.00
Resistor Box	ECE Store	20.00	2	40.00
Transistors	ECE Store	0.50	5	2.50
Diodes	ECE Store	0.25	6	1.50
Capacitors	ECE Store	0.30	10	3.00
Potentiometers	ECE Store	0.50	5	2.50
LEDs	ECE Store	0.50	5	2.50
			Total	**274.13**

4.2 Labor

Table 3 Labor Costs

Group Member	$/Hour	Hours/Week	Number of Weeks	Multiplier	Total/Person($)
Kevin Noh	30.00	12	12	2.5	10,800
Rajat Sharma	30.00	12	12	2.5	10,800
Troy Chmieleski	30.00	12	12	2.5	10,800
				Total	**32,400**

4.3 Total Cost

Table 4 Parts Costs

Parts Cost	$ 403.63
Labor Cost	$32,000.00
Total Costs	**$32,803.63**

5. Conclusion

5.1 Accomplishments

The design worked as expected. The power unit was able to power all the other modules successfully. The charging circuit could charge the battery in 2 hours. Additionally, the USB charger could charge an IPhone 5s in 3.5 hours. The battery was able to power the rest of the modules for a period of 4 hours. The weight sensors were calibrated correctly such that the bag could measure a total weight of up to 60 pounds. This range was sufficient enough for the purpose of weighing backpacks that are used in schools. The microcontroller was successfully able to determine the weight of the backpack and transmit that data to the smartphone. The Bluetooth module was able to transmit data from the microcontroller to the smartphone up to a distance of 20m. The smartphone had an easy to use interface that displayed the correct weight of the backpack. Finally, the proximity sensor performed as expected because it was able to deliver an alert if the smartphone was moved more than 20m away from the backpack.

5.2 Uncertainties

Even though our project worked as expected during the demonstration, it is possible that the weight readings may become incorrect over time. The 5V voltage regulator has a tendency to overheat after 20 minutes. Since we do not have a heat sink in your circuit, the output of the 5V convertor may not be stable for long periods of time. This in turn could change the excitation voltage of the sensors, which may lead to incorrect weight readings. Also, the weight read on the smartphone could sometimes vary depending on how the backpack's strap was held. An uneven distribution of weight on the strap could lead to incorrect readings. This issue may be resolved by adjusting the current placement of the sensors. However, the current design may lead to this problem on occasion.

5.3 Ethical considerations

All group members adhered to the IEEE code of Ethics throughout this project.

Since the project was aimed to develop a product that may be used in a person's everyday life, it was important to adhere to code 1 of the IEEE code of ethics: *"to accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment"*[a1]

This project has many areas for growth and development. Therefore, it was important for us to adhere to code 7 of the IEEE code of ethics: *"to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others"*[a1]

Finally, since our project involves calculating the weight of the backpack based on our algorithm, there may have been some variations between the actual weight of the backpack and the weight determined by our algorithm. Consequently, it was important for us to adhere to code 3 of the IEEE code of ethics *"to be honest and realistic in stating claims or estimates based on available data"*[a1]

5.4 Future work

We could possibly integrate the weight sensors with the shoulder straps in the future. Doing so would allow the user to weigh the backpack by simply wearing it on his/her shoulders. Consequently, we would be able to track the user's weight in real time and determine the affect of the backpack on his/her health. We could also implement a weight consistency tracking feature. This feature would alert the user if he/she were to forget an item in his/her back at a location other than his/her home. For example, the smartphone could keep track of the user's backpack as he/she would be in a class. Now, if the weight of the backpack were to reduce as he/she were about to leave the class, then the smartphone could alert the user that he/she may have forgotten an item in the class. Finally, we could also develop a social platform for users so that they could compare their tracking habits with each other.

References

[1] Webmd.com, 'Heavy Backpacks Can Hurt Students' Backs', 2015. [Online]. Available: http://www.webmd.com/back-pain/news/20040813/heavy-backpacks-can-hurt-student-backs. [Accessed: 04- May- 2015].

[2] Z. Heuscher, D. Gilkey, J. Peel and C. Kennedy, 'The Association of Self-Reported Backpack Use and Backpack Weight With Low Back Pain Among College Students', *Journal of Manipulative and Physiological Therapeutics*, vol. 33, no. 6, pp. 432-437, 2010.

[3] Newsroom.ucr.edu, 'UCR Newsroom: Heavy Packs Bring Kids Pain', 2015. [Online]. Available: http://newsroom.ucr.edu/868. [Accessed: 04- May- 2015].

[4] D. Siambanes, J. Martinez, E. Butler and T. Haider, 'Influence of School Backpacks on Adolescent Back Pain', *Journal of Pediatric Orthopaedics*, vol. 24, no. 2, pp. 211-217, 2004.

[5] Ieee.org, 'IEEE IEEE Code of Ethics', 2015. [Online]. Available:
http://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 04- May- 2015].

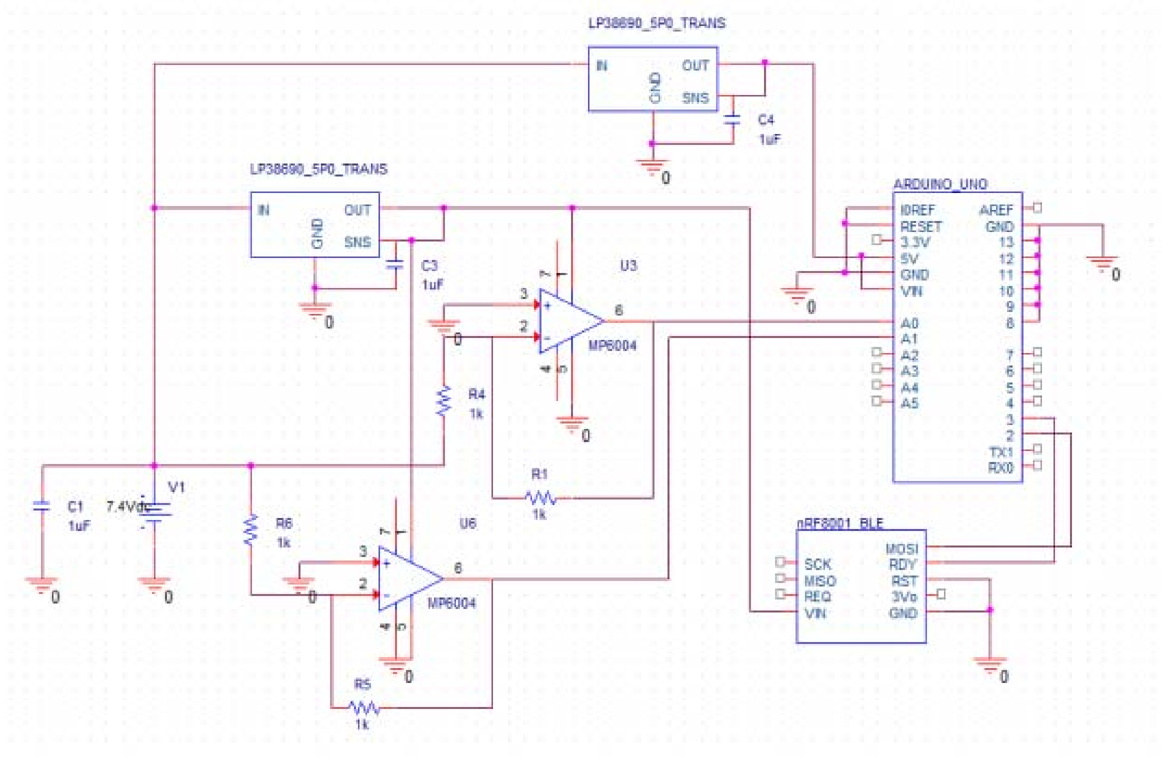
Appendix A Requirement and Verification Table

Table 5 Requirements and Verification

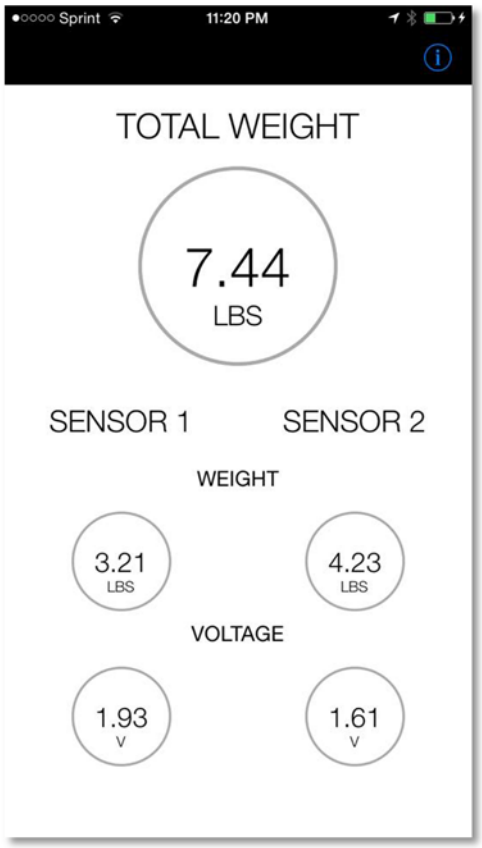
	Requirements	Verification	Verification Status
Power Unit	1. Battery must provide a voltage of 7.4V +/- 0.3V as output 2. Battery charger circuit charges the battery by providing 7.2 V (which is changeable) as output 3. The USB Charger circuit charges an IPhone by providing 5V, 2V, 2V, and 0V to Vin, D-, D+ and Ground to the USB female port	1. Measure output of battery using multimeter. 2. The LED lights up when the battery is charging and turns off when the charging is complete. Measure voltage across battery while it is being charged to verify charging of battery. 3. Connect an IPhone to USB port to verify occurrence of charging	Y
Weight Sensors	1. Excitation voltage is 2.3 V (+/- 0.2V) 2. Output of the first sensor is 0.22mV (Approx) Output of the second sensor is 0.6mV (Approx) 3. The first amplified output for the first sensor is 0.22V (Approx). The first amplified output for the second sensor is 0.18V (Approx) . 4. The Second amplified output of the first sensor is 1.1V (Approx). The Second amplified output of the second sensor is 1.1 V (approx) 5. Pulling the hook on the weight sensors changes the output voltage of the second amplifier to a maximum of 5 volts	1. Measure Excitation voltage using a multimeter 2. Measure output voltage of the first and second sensor using a multimeter 3. Measure the output voltage of the first amplifiers for both sensors 4. Measure output voltage for the second amplifiers for both the sensors 5. Attach weight to the sensors and measure output voltage of the second amplifiers of both sensors to view increase in voltage	Y

Microcontroller	1. Input voltage is 7.4V (+/- 0.3V) 2. Input is between 0V and 5V 3. Output from the Microcontroller is connected to the Bluetooth module	1. Measure voltage using multimeter. 2. Measure input voltage using a multimeter. 3. Test continuity of wires connecting the output of the microcontroller and the input of the Bluetooth using a multimeter.	Y
Bluetooth	1. Successfully transmits and receives signal via UART for the communication between the Microcontroller, Bluetooth module, and the Smartphone. 2. Bluetooth module must be able to transfer/receive data to/from an Iphone up to a distance of 1 meter.	1. Note the analog voltage reading recorded by the microcontroller. Display the reading on the smartphone. Compare the readings on the microcontroller and the smartphone to verify transmission of data. 2. Send a test message from the microcontroller to the smartphone over Bluetooth. Keep phone 1m away from the backpack. View message received on the smartphone to verify transmission of data 1m away	Y
Smartphone	1. Displays weight of the backpack 2. Receives a proximity alert when the backpack is 20m (30 steps) away from the smartphone	1. View smartphone app to view the weight of the backpack 2. Move 20m (30 steps) away from backpack to test proximity sensor	Y

Appendix B Initial Circuit Schematic of Sensor Unit



Appendix C Additional Pictures


●○○○○ Sprint
11:20 PM
TOTAL WEIGHT
7.44
LBS
SENSOR 1
SENSOR 2
WEIGHT
3.21
LBS
4.23
LBS
VOLTAGE
1.93
V
1.61
V