REMOTE FIREWORKS LAUNCHER

By

Daniel Middendorf

Michael Hlinka

Trent Sanford

Final Report for ECE 445, Senior Design, Fall 2018

TA: Kexin Hui

December 2018

Project No. 5

Abstract

Our senior design project is a remote fireworks launcher. The project consists of four physical components: a controller, receiver, and two sensor pods. The controller is a custom-made user interface to send launch commands to the receiver at a safe distance of 150 feet and display important system information. The receiver is connected to a set of 32 electrical fuses to ignite fireworks, and along with our two sensor pods, contains motion sensors to detect activity near the launch deck. Our project was successfully able to ignite fuses, detect motion near the launch deck, and had reliable wireless communication between all components.

Contents

1. Introduction

1.1 Background

There are many annual occasions where fireworks are a staple in celebrating. These occasions, most notably the Fourth of July, often have large shows where fireworks are launched by professionals. Unfortunately, there are also plenty of amateurs during these events that launch fireworks privately. In 2016, it was estimated that 11,100 people were injured and 4 deaths were reported as firework related [1],[2]. These numbers have varied a bit over the past decade, with the lowest estimate being 7,000 in 2008 and the highest estimate being 11,900 in 2015 [1]. Most minor injuries from fireworks are from sparklers, while most of the firework related deaths are due to users being in close proximity while lighting off mortar firework rounds.

"On July 4, 2016, a 42-year-old male from Florida suffered fatal injuries when the fireworks device he was lighting malfunctioned. According to the county deputies, the victim was trying to set off large mortar-type fireworks in a PVC pipe that was anchored to the ground." [1]

There is not much to be done about the individuals' safety while using sparklers, but more can be done to help prevent the more serious accidents. Our solution for the more serious types of injuries was to design and implement a wireless launcher that can ignite mortar rounds for amateur consumers. The user would be able to observe and control the launching of the fireworks a safe distance from the launch deck, approximately 150 ft away. Our design would be equipped with motion sensors to detect anyone within proximity of the fireworks launching platform. We want our project to emphasize safety above all else. While backyard fireworks are always dangerous, our goal is to help minimize injuries at an affordable cost.

Right now, there are some products similar to the ones we just described. Cobra Firing Systems provides professional equipment for large shows. This equipment ranges from $500, at its cheapest, to $1350 [3], [4]. This is obviously not reasonable for the average consumer. Firefly Firing System offers a cheaper alternative for firework enthusiasts, at $200 [5]. While this is cheaper than the professional firing systems, there are no safety measures in place to indicate when someone may be in danger around the fireworks.

Our completed solution is cheaper and, more importantly, safer than what is currently available on the market. Our project has reliable communication between the controller and the receiver from 150 ft away, can set off our igniters at a 95% success rate and reliably detect anyone within an 8 ft radius of the firework launch deck with our motion sensors. Additionally, our project is rather affordable and is less than our initial price goal of $120.

1.2 Components

Our design consists of four separate components: a controller, receiver, and two identical sensors pods. The individual block diagrams for these components can be found below in **Figure 1.2.1**, **Figure 1.2.2** and **Figure 1.2.3**.

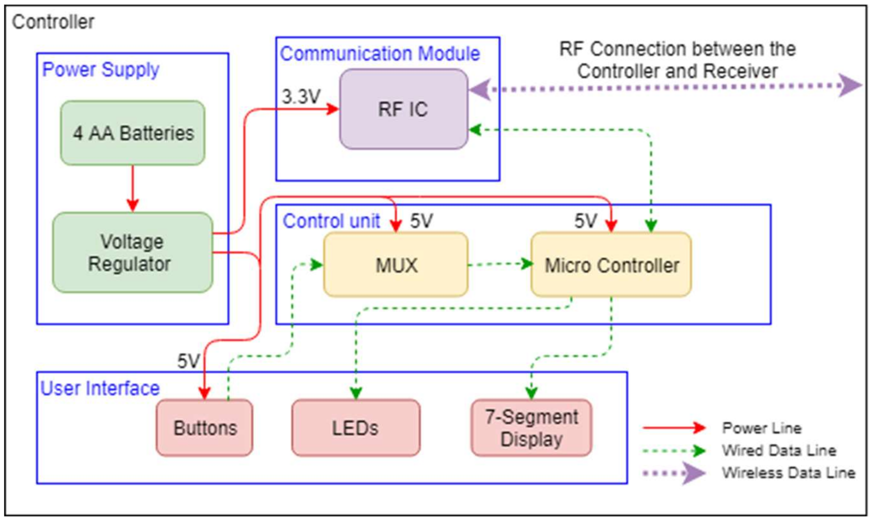

Figure 1.2.1 Controller Block Diagram

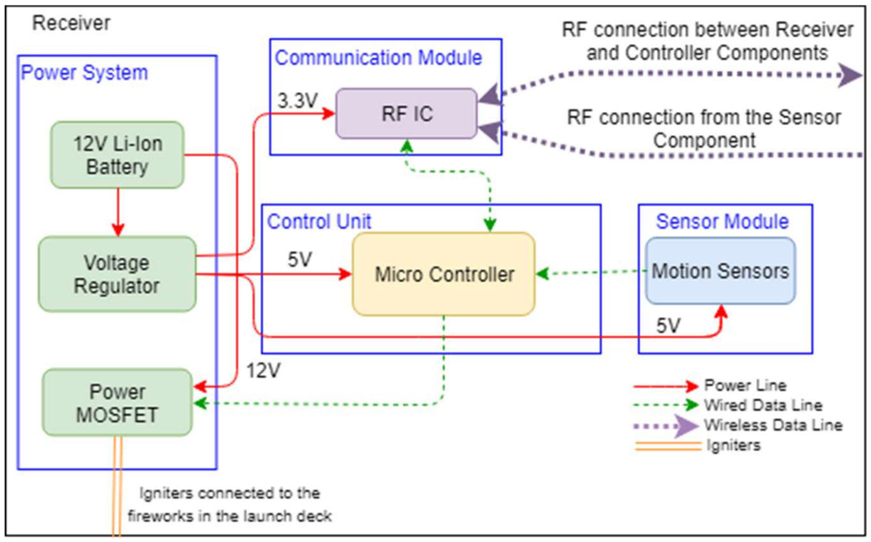

Figure 1.2.2 Receiver Block Diagram

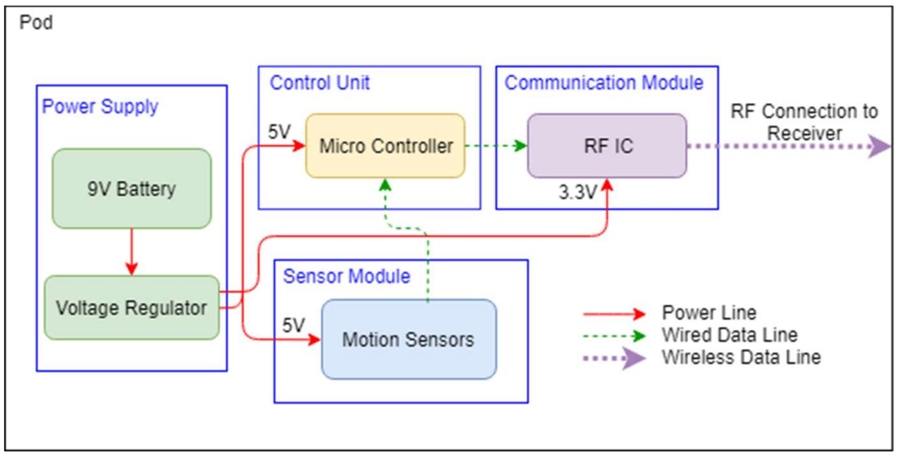

Figure 1.2.3 Sensor Pod Block Diagram

Even with separate components, a few of our blocks are shared among all three. Each component has its own power supply with 3.3V and 5V voltage regulators to power all the different IC components. The three components also have their own microcontroller and RF chip to communicate with each other. The controller uses one acknowledge button, a four digit seven-segment display, and a set of thirty-two buttons and LEDs as a medium between the user and our project. A serial multiplexer (MUX) is used to send button input to the controller's microcontroller for processing. Both the receiver and the sensor pods use PIR motion sensors to detect motion around the launch deck. Additionally, the receiver uses a set of power MOSFETs controlled by the receiver's microcontroller to set off the electronic igniters.

Our group has made some small changes from what was originally laid out in our design document. The type of battery used for the sensor pods was changed to a single 9V battery because we already had the correct connectors and they fit more easily into the sensor pod's 3D printed casing. The DEMUX originally planned to be used in the receiver was cut due the large number of excess digital pins that were available on the receiver's microcontroller. Keeping the DEMUX added an unnecessary element of complexity and its exclusion would reduce costs.

2 Design

2.1 Power Supply Design

When considering power supply, we needed to consider the physical size, ideal output voltage, and in the case of the receiver, the max current the battery could supply. Since we used IC chips, we knew that we needed at least 5V at the source to meet the power supply requirement for those chips and for our wireless communication chips. With the minimum size of the controller limited by the length needs of our user interface, we were free to use a larger battery pack of 4 AA batteries in series.

The size of the battery became a problem when dealing with the sensor pods. Originally, we were planning on using a similar type of battery pack as the controller with AAA batteries instead. We realized that even these smaller batteries still would have a hard time fitting into our cases. After discussing the issue, we decided to use a single 9V battery, which can fit inside our pod casing without any problems.

When figuring out the battery needed for the receiver, we needed a more powerful supply than the previous two. The igniters we chose required at least 12V and 1A to be set off consistently. After doing research online, we chose an ABENIC 12V Li-ion battery since it was cheaper than other rechargeable batteries and met the power needs for our igniters. A visual representation of what the power sub-circuit looks like can be seen in **Figure 2.1.1** below.

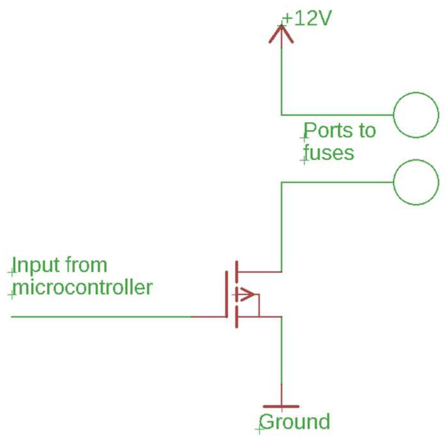

Figure 2.1.1 Power Circuit for Igniters

Along with the different batteries, each device also needed two voltage regulators to step down the voltage to 5V for the IC components and 3.3V for the wireless communication chips. The receiver also required n-type MOSFETs to act as a barrier switch that can be triggered by an IC chip while still being able to handle the current and voltage needed for the fuses.

2.2 Control Unit Design

When choosing the parts for our microcontrollers we considered how many programable pins are available and how much supporting software and documentation the chips have online for programing them. We settled on using two different chips for our project: Atmega2560, Atmega328p. The Atmega2560 offers eighty-six general programing pins while the Atmega328p offers 23. We specifically chose these two chips specifically because they are both chips that are used in a widely used breakout microcontroller company called Arduino.

Our controllers' functionality was managed by an Atmega2560 microcontroller paired with a serial MUX to process user input. An Atmega2560 was specifically chosen to ensure that our design had enough digital I/O pins for both our user interface and wireless communications. This microcontroller was programmed to follow the state system shown below in **Figure 2.2.1**. Following the states shown in the figure, the user can press one of the 32 launch buttons on the controller to enter the 'Queue' state. The user can change the currently queued igniter number by pressing a different launch button. After confirming the launch number by pressing the acknowledge button, the 'Wait' state is reached and the controller sends a wireless launch message to the receiver. If an acknowledge message from the receiver is read, we move to the 'Complete' state, signifying a successful launch. Afterwards, there will be a short 10 second cooldown in the 'Cooldown' state before the user can start the process over to set off another igniter. While in the 'Wait' state, if there is no response from the receiver after 3 seconds, the controller will transition to the 'Fail' state and must be acknowledged by the user. All of these states can be interrupted if the controller gets a motion error message from the receiver, moving the controller into the 'Error' state. This state will return to the 'Start' state with no buttons queued after 15 seconds but, the timeout will restart if another motion error is received while still inside the 'Error' state.

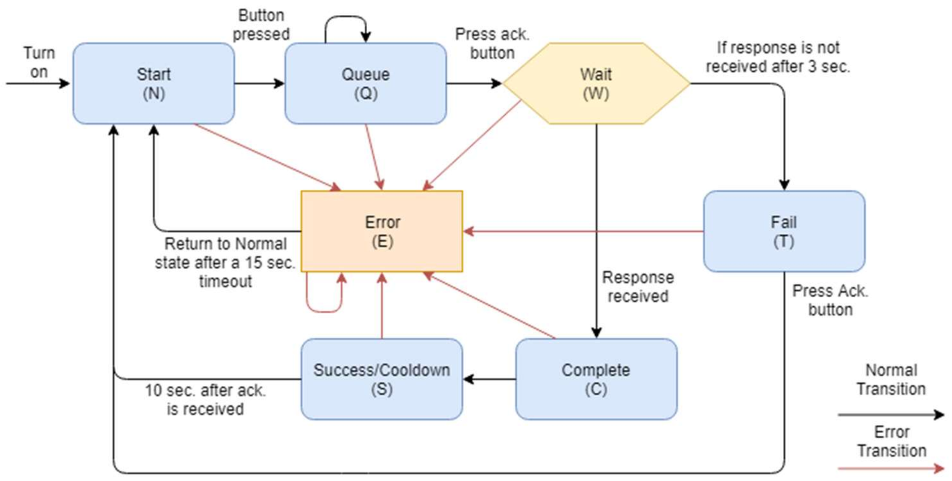

Figure 2.2.1 Controller State Flowchart

The receivers' control unit is somewhat simpler than that of the controller. The receiver also used an Atmega2560. With all the pins offered on the 2560, we were able to simplify our design and omit the DEMUX by tying the gates of our MOSFETs directly to digital pins on the microcontroller. The receivers' control unit is responsible for forwarding wireless messages and processing launch commands from our controller as illustrated in **Figure 2.2.2**. When a launch command is received, the microcontroller will set the appropriate MOSFET gate to high, allowing current to flow through the MOSFET and igniter, setting it off. Afterwards, the microcontroller uses our communication module to send an acknowledge message back to the controller and return the MOSFET gate to low. If instead, a motion error is received from one of our pod units, or generated from the receiver itself, a motion error message will be sent to the controller and the receiver will begin a 15 second timeout where no launch operations can be performed. This timeout will be restarted if another motion error occurs while still in the timeout.

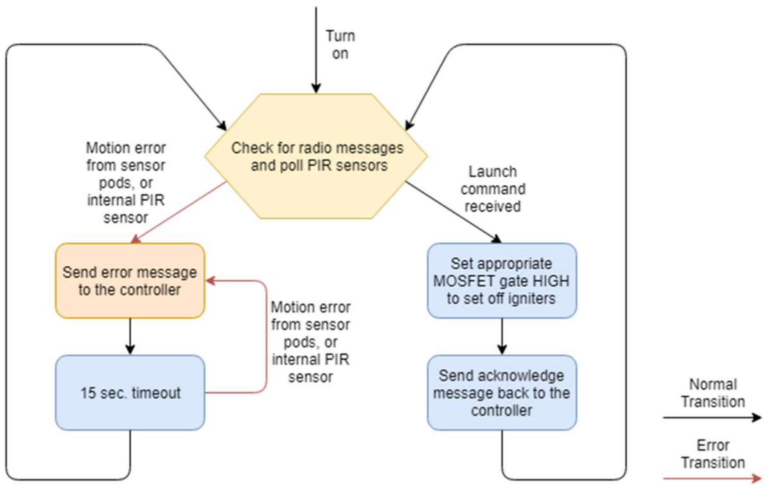
Turn on
Check for radio messages and poll PIR sensors
Motion error from sensor pods, or internal PIR sensor
Launch command received
Send error message to the controller
Motion error from sensor pods, or internal PIR sensor
15 sec. timeout
Set appropriate MOSFET gate HIGH to set off igniters
Send acknowledge message back to the controller
Normal Transition
Error Transition

Figure 2.2.2 Receiver Software Flowchart

The sensor pods' control unit is a simpler version of that in the receiver. The sensor pods' microcontroller constantly monitors the readings from their internal PIR sensors and sends a motion error signal to the receiver if a spike is detected (more on this in section 2.5). Because the sensor pods only need to do this one operation, we used the smaller surface mount Atmega328p microcontroller in each of the sensor pods.

On each of our components PCB's we added breakouts for the 5V, GND, RX, TX, MISO, MOSI, SCK, and RESET pins. These breakouts allowed us to upload the Arduino bootloader and Arduino sketches onto our microcontrollers without having to programing beforehand. This made code debugging in all of our components much easier.

2.3 User Interface Design

When first considering how we wanted the user interface to look and what it consisted of, we first needed to decide how many unique igniters the user would be able to set off before needing to reset the system. We set our starting number of unique launches to be twenty-five since any number less than that would require too much setup from the user without sufficient reward. After declaring this minimum, we turned our attention to different ways we could minimize the number of inputs to our microcontroller. We found a 32:1 serial mux that reduced out total number of microcontroller pins needed from thirty-two to three. After finding out the number of buttons we are going to use, we concentrated on how we would arrange the buttons on the face of the controller. We decided to go with a 2D matrix of eight columns and four rows as this was the most visually appealing arrangement to us.

After taking care of the input side of the user interface, we then discussed how we wanted to represent that the user has already set off one of the igniters. Wanting to keep the design simple to understand, we decided to use a similar dimensioned 2D matrix of LEDs as shown in **Figure 2.3.1** below. A lit LED indicates that the igniter has not been set off yet; conversely, a darkened LED indicates to the user that they have already pushed that button and the receiver has set off that igniter. Looking at this set up, there is no way to potentially power all thirty-two LEDs at the same time. To make it appear that all the LEDs are on at the same time, we trick the human eye with a phenomenon called "Persistence of Vision" or POV for short. What this trick entails is to cycle through the different rows fast enough that the human eye cannot discern which row of LEDs are powered on and which are not. This tricks the human eye into believing that all the rows are always powered.

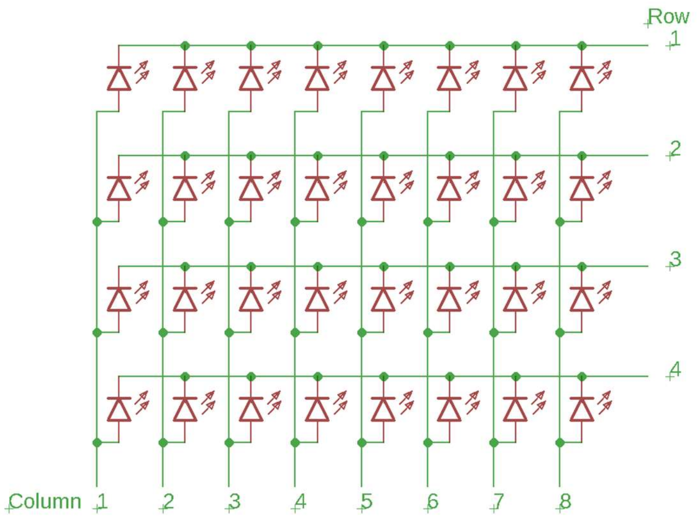

Figure 2.3.1 LED Matrix Representation

Since our project does more than mindlessly set of igniters, we have multiple states our system can be in. Each of these states requires different actions to be taken by the user. This meant that we needed to include some type of display to aid the user and inform them of what state the system is in and the actions they can take at that moment. We chose to use a four digit seven-segment display to convey this information. Our display will tell the user if motion has been detected by the launch platform, which button they have queued up, if the system is ready to take another button input, and whether the launch packet has been receiver by the receiver. We found that the most logical position on our controller interface for our display is centered at the top, right next to the acknowledge button since one of the main functions of the display is to parrot the launch number back to user and wait for the signal from the acknowledge button to send the launch packet.

2.4 Communication Module Design

The communications module is one of the most important blocks of our design and enables the primary safety feature of this project. To facilitate this block as much as possible, we used multiple NRF24L01 transceiver modules in our design. Research suggested that these modules were easy to use with

Arduino combatable microcontrollers and should meet our 150 ft minimal range requirement. Originally, we were considering using the ESP8266 Wi-Fi modules instead, but after researching how to use the modules we concluded that we would need to have two per component for reliable two-way communication. Communication over Wi-Fi also added other networking complexities to the design where simple RF communications would suffice.

Our project communicates with its fellow components using 3-byte messages sent using the NRF24L01 (with help from the RF24 Arduino library). An example data packet can be seen in **Figure 2.4.1**.

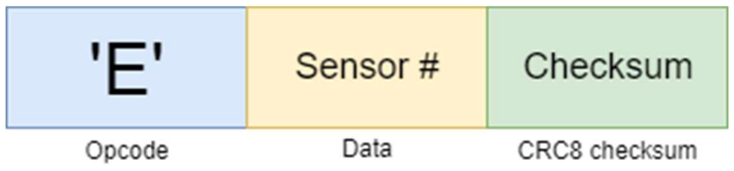

Figure 2.4.1 Sample Data Packet

Each message begins with a byte long opcode signifying the message type (a full list of message types can be found in **Table 2.4.1**). The second byte contains data relevant to the packet and the final byte is an appended CRC8 checksum for error checking. The 8-bit version of the CRC checksum was used due to our messages' small size. The CRC8 checksum also helps in preventing random signals being potentially interpreted as firework launch commands.

Type	Opcodes	Data
Launch	L	Button/Fuse #
Acknowledge	A	Button/Fuse #
Motion Error	E	Sensor #

Table 2.4.1 Wireless Message Types

2.5 Sensor Module Design

The objective of these sensors is to detect motion within an 8 ft radius around the launch area. The sensor module of our system is designed to utilize the IRA-S210ST01 passive infrared sensors, paired with the IML-0866 lens, and gather readable information for our microcontroller to interpret as described in Section 2.2. Our project uses three of these sensor modules in an equilateral triangular setup to maximize our field of view when attempting to detect motion around the launch area. The sensors and lens we used had a field of view of 6 meters across and 3 meters deep as seen in **Figure 2.5.1.** The physical layout of our motion sensing area can be seen in **Figure 2.5.2**.

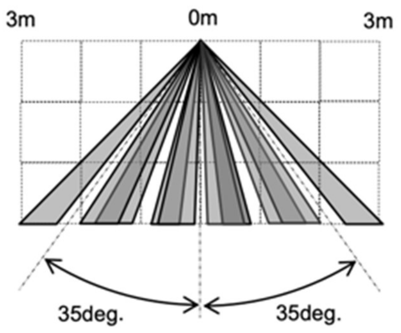

Figure 2.5.1 IRA-S210ST01 Field of View [6]

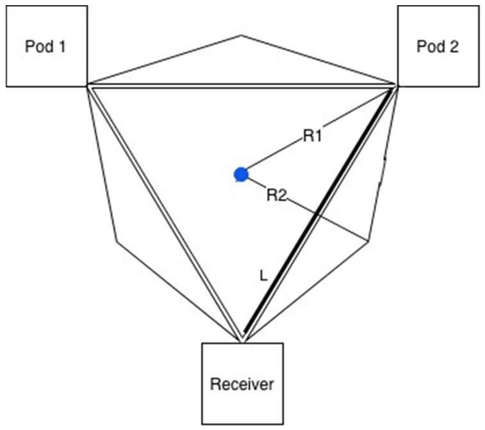

Figure 2.5.2 Physical Layout of Detection Area

The specific sensor circuit went through a few design iterations before our module could reliably detect motion within our detection area. These motion sensors are capable of outputting a minimum of 0.2V to a maximum of 1.5V via the source pin [6]. We started our design by using the test circuit for the sensors source voltage seen in **Figure 2.5.3**.

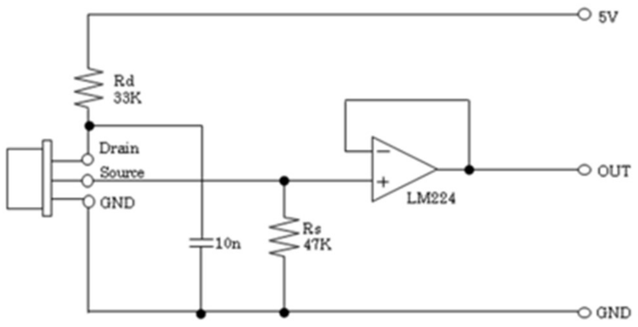

Figure 2.5.3 Test Circuit for Source Voltage [6]

After testing this circuit for source voltage, we moved to make the circuit more barebones to see if we could receive the same results as the test circuit. The change in output voltage is what we were concerned most with because measuring this change from the sensor was how we would detect motion within the launch deck. We found that even though the operational amplifier did amplify the source voltage, it had no effect on the range of voltages given by the sensor. We found that when testing with an Arduino Uno, the 10nf capacitor and 33kΩ resistor were seemingly unnecessary for our measurements also. In hindsight, these two components created a low-pass filter to help reduce noise from our input voltage. When we moved to testing on our PCBs, it was apparent there was much more noise present than the test on the Arduino Uno. Our final sensor circuit iteration re-added the filtering capacitor to stabilize the output we were receiving on the source voltage pin as seen in **Figure 2.5.4**.

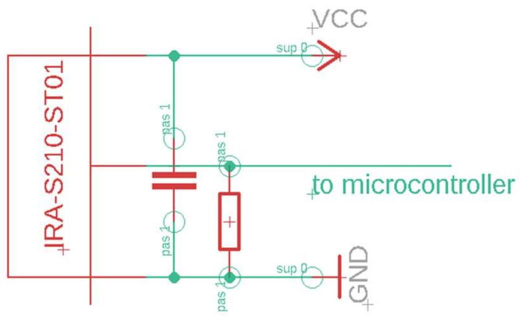

Figure 2.5.4 Final Sensor Circuit Schematic

3. Design Verification

3.1 Power Supply Verification

Part of our requirements for our batteries was that they could fit, along with other electrical components, inside the physical devices we created. This was easily tested and verified by simply wiring up all the components and seeing if they could all fit inside our devices. All of our power supply units did end up fitting inside our devices along with the other components of each system.

Each power system also included a 5V and 3.3V voltage regulator to step down the supply voltage for the IC chips and IR communication chip respectively. The verification of these regulators was done by providing a source voltage of 6V to the input of the regulator and observing the output voltage using a multimeter. For both the 5V and 3.3V regulators we observed the ideal output voltage plus or minus the allowed tolerance of 5%. We continuously supplied the regulators with power for over an hour to make sure that they would not overheat with constant use and after over an hour of use we observed no great increase in temperature in the regulators.

Along with the battery supply and voltage regulators, the receiver also required n-type MOSFETs to protect the IC chips from the power needs when setting off the fuses. To test that our chosen MOSFETs worked, we applied a 12 V source to a resistor connected to the drain of our MOSFET. Refer to **Figure 2.1.1** for clarification. With the source pin connected to ground we set the gate pin to 5V, representing what the signal from the microcontroller would send, and measured the voltage different across the resistor before and after turning on the MOSFET. Measuring the voltage drop across the resistor before applying 5V to the gate, we observed a voltage drop of 0V. After setting the voltage to our gate to be 5V, we observed that the voltage drop jumped up to being within our tolerance of acceptable voltages to set off our igniters.

3.2 Control Unit Verification

The separate control units for out controller, receiver, and sensor pods have their own set of requirements and verifications as seen in **Table A**, but many of these requirements are overlapping. To test the full functionality of the logic from the controller and receiver's microcontroller shown in **Figure 2.2.1** and **Figure 2.2.2** respectively, we developed the code by unit testing and debugged with help from the Arduinos' serial monitor. For the controller specifically, we printed our current state on the serial monitor to ensure that the state transitions happened when expected and produced the intended behavior. We also used the serial monitor to test if the controller can identify if the user has pressed a button and the specific button pressed. This was done by printing the button value that the microcontroller thinks were pressed to the serial monitor and checking the value. For the receiver we connected our MOSFET drain to a LED in series with a resistor. We then wrote a simple program that would switch the MOSFETs' gate voltage from 5V to 0V repeatedly. If the MOSFET has been solder correctly, then the LED would blink on and off.

We were able to show that our components had the intended software functionality and could send data over the communication module in our project's demonstration. Our controller had unique messages display for each state from **Figure 2.2.1** and our receiver could set of a specified igniter when a launch signal was sent from the controller.

3.3 User Interface Verification

The verification for the user interface happened in two stages. After getting all the parts we tested the functionality of each part individually. For the buttons we made a simple pullup resistor circuit connected to one terminal with the other one tied to ground. Measuring the voltage of the terminal with the pullup resistor, we pressed the button and observed if the voltage would go from 5V to 0V. The LEDs were a bit simpler. For those we just touched the two ends of the LED to a CR2032 Maxell 3V battery and observed if the LED would light up. The seven-segment display screen was the most complicated unit test of the interface components. For this we wired up each terminal to an Arduino UNO and, using an existing library, tested some basic output features that we would be using in our final implementation discussed in section 2.3.

The second stage for verifying our user interface implementation was at the end of our project by connecting all the components together on the inside of the controller device and observing that everything worked as expected. For the buttons we wired one terminal of each button together and connected that to a ground breakout pin. The other ends were individually connected to pullup resistor breakouts. For wire management purposes, we designed our PCB board such that each vertical column of buttons' wires could be broken out as a ribbon cable and attached to the board that way. For the LEDs we attached each rows cathode (-) together and connected those rows to a ribbon cable to attach to the breakouts implemented. We then attached the anodes (+) of each column together to a breakout ribbon cable. Unlike the original testing and verification, the seven-segment display was the easiest to install in the controller face. All we needed to do attach ribbon cables to all the pins such that they could be plugged into the breakout pins of the PCB. After installing all the components, we then connected all the ribbon cable breakouts, turned on our device and observed the functionality of our controllers' user interface.

3.4 Communication Module Verification

The key function of our communications module is to ensure that the control units of each component can properly communicate with each other. To test this, we used an Arduino Uno connected to one of our spare RF transceiver modules to test if each was able to send and read wireless messages. The Arduino's were uploaded with test programs that we wrote beforehand with the help of Dejan Nedelkovski's tutorial for the NRF24L01 transceiver modules on the *How To Mechatronics* website [7]. We used the serial monitor on both the Arduino and our components microcontrollers to check that the communication worked successfully and use that information to debug our system.

Additionally, we tested that the transceiver modules used in our components could reliably communicate between each other from a minimum range of 150 ft as specified in our high-level requirements and the RF module section of **Table A**. This was tested using two Arduino Unos connected to RF transceivers. One Arduino will send a specific message to the other and the other Arduino responds by sending a unique message back. We took these two Arduinos to opposite ends to the hall outside of the ECE Senior Design lab (with distance of roughly 200 ft) and confirmed that the sender could display the receiving Arduino' s unique message back on its serial monitor. This confirmed that our wireless communications met our requirements.

To ensure that our checksum also met our requirements, we created a short test program using the FastCRC library used in the code for our components. This test program took a sample set of (1,000, 10,000 and 100,000) randomly generated 3-byte messages and tested if any passed our CRC8 checksum. We found that for each sample size we tried, 99.5% of the random values failed the checksum, meeting our checksum requirement.

3.5 Sensor Module Verification

The verifications for our sensor module required detection of motion within 8 ft of the sensor. To correctly identify motion within our detection area we first needed to understand the range of output voltages we were dealing with. The way we tested this was by using the Arduino IDE serial monitor and a digital multimeter. The sensors' output was tied directly to an analog input pin on the Atmega328p microcontroller and converted to an integer value of 0-1024 with a reference to 1.1V. To start, we conducted a control test where everything within the detection area was motionless. We then had a person walk across the field of view 8 ft away from the sensor and recorded the voltage range the sensor outputted. The range we observed can be seen in **Table 3.5.1**.

Range	Motionless	Moving 8 ft Away
Low (mV)(Integer)	502mV (467 integer)	502mV (467 integer)
High (mV)(Integer)	506mV (471 integer)	517mV (481 integer)

Table 3.5.1 Sensor Voltage Test Range

Our main concern was determining what constitutes movement within this voltage range. As you can see from **Table 3.5.1**, the voltage from the sensor changes very little when standing still. From our testing, it seemed that the sensor would find a resting voltage when everything was stationary in its field of view and would jump considerably, relative to the range of voltages we were getting, whenever there was movement at 8 ft away, or closer. Due to this observation we designed our sensor code to poll the analog output every fifth of a second and do a simple reference check to the previous sensor value. If the difference between the current value and the previous value was greater than one, we marked that as a motion spike. The delay in polling allowed us to safely disregard any minimal voltage change a still detection area may give our sensor while allowing a great enough voltage change to occur if there was motion in our detection area.

4. Costs

4.1 Parts

This project was designed and built with an average consumer in mind. This is the reason we made this project as low cost as possible and set our upper limit to be $120. Many of the parts we used for the prototype were purchased at the single unit cost. These same parts also had the option for bulk purchases at a cheaper cost per unit. Ideally, if we took this project to market, we would order and assemble our system in mass quantities so the limit of $120 applies to our mass production costs. The breakdown of our parts cost for both our prototype and mass production units can be found below in **Table 4.1.1**.

Along with the electrical parts, we required bodies for each of our components. We were lucky enough to have access to a 3D printer for the development of our prototype as this did not increase our prototype cost. This would not be possible for mass production. The alternative we investigated was injection molding and came to an estimate of $5 per unit [8]. This brings our total estimate for mass production cost per unit to $84.68 and keep us $35 under our cost limit.

Part	Manufacturer	Quantity	Retail Cost/Unit ($)	Prototype Cost ($)	Bulk Purchase Cost ($)
ATmega2560	Microchip	2	11.85	23.70	17.22
ATmega328p	Microchip	2	2.07	4.14	3.44
IRA-S210ST01	Murata Technologies	3	3.12	9.36	3.72
IML-0688	Murata Technologies	3	3.30	9.90	4.08
LD1117S33TR	STMicroelectronics	4	0.44	1.76	0.72
LD1117S50TR	STMicroelectronics	4	0.44	1.76	0.72
NRF24L01	Nordic Semiconductor	4	1.20	4.80	4.80
ADG731BSUZ	Analog Device	1	10.67	10.67	6.23
SI1442DH-T1-GE3CT-ND	Vishay Siliconix	32	0.57	18.24	2.88
TDCG1050M-ND	Vishay Semiconductor	1	2.91	2.91	1.02
A4YBS Uxcell	UXCELL	33	0.39	12.87	12.87
A13122500ux0991	UXCELL	8	1.45	11.63	11.63
LEDS	Areyourshop	36	0.01	0.36	0.36
Rechargeable 12V Li-ion Battery	ABENIC	1	20.99	20.99	7.99
PCB	PCBway	4	0.50	2.00	2.00
Body of Components	N/A	1	N/A	FREE	5.00
Parts Total				**135.09**	**84.68**

Table 4.1.1 Part Costs

4.2 Labor

Our labor cost comes from the average salary of students who graduated with a BS in Computer Engineering in the '2014-15' academic year. These students earned an average of $84,250 per year [9]. This salary divided by 2,080 because this is the amount of work hours in a year (40 hours per week * 52 weeks per year). This division yields and hourly rate of $40.50 which we rounded down to $40 per hour.

Name	Rate ($/h)	Hours (h)	Total*2.5 ($)
Daniel	40.00	225	22,500.00
Michael	40.00	225	22,500.00
Trent	40.00	225	22,500.00
Total Labor			67,500.00

Table 4.2.1 Labor Costs

4.3 Total Prototype Cost

Total Parts ($)	Total Labor ($)	Grand Total Prototype Cost ($)
135.09	67,500.00	67,635.09

Table 4.3.1 Total Costs

5. Conclusion

5.1 Accomplishments

Our project was to design a remote fireworks launcher with an emphasis on safety. We accomplished this by developing three different devices: a controller that will be the main medium between the user and our product, sensor pods that will be placed around the launch area to help detect motion, and a receiver that will set of the igniters and act as a communications hub between the controller and the sensor pods. Our project worked just as purposed. The user can select between thirty-two buttons, each corresponding to a unique igniter. Once the user selects a button, the seven-segment display will show the requested number back to the user. If no motion errors have been detected by the receiver or the sensor pods, then the user can hit the acknowledge button which sends a launch code packet to the receiver where it will be interoperated and the corresponding MOSFET will be activated. This allows current to flow from the Lithium-ion battery, past the igniter, and through the MOSFET to ground which then sets off our igniters, lighting the fireworks fuse.

5.2 Uncertainties

One of the aspects of our project that we believe was unsatisfactory was the detection ability of our sensor modules. While these modules worked great at detecting motion, they were unable to detect stationary people, or foreign objects, within the launch area. Even in our array of sensors, once you give all the sensors time to stabilize to a stationary environment, nothing will be detected unless you move. Since much of our testing for our sensor modules was done with a single sensor, we had initially believed that having all sensors present would help detect stationary targets within the detection area but that was not the case with our current implementation. As mentioned in Section 3.5, the sensors seemed to have resting voltage. There may have been potential to calibrate the sensor array to know exactly what the resting voltage of each sensor should be when set up in the proper configuration, see **Figure 2.5.2**. Once these resting voltages have been identified, we then might have been able to compare that value with the most recently pulled value instead of using our existing method. This would allow us to detect any anomaly within the launch area and potentially reveal any stationary person in danger. Unfortunately, we were unable to test this theory.

Aesthetically the most important part of our project is the user interface. Even though functionality wise the interface works perfectly, one thing we would have wanted to improve is the refresh rate of the LEDs and display. Currently both parts appear to flicker slightly and can become a distraction to the user. We would have liked to decrease this effect so that the LEDs and display appear to be more solid instead.

5.3 Ethical Considerations

Due to the nature of our project there are some ethical, and legal concerns to consider. The primary use of this project is to safely ignite pyrotechnic products, specifically mortar fireworks. When using these products, federal and local laws/policies should be strictly followed. The use of fireworks is regulated federally and are illegal in the state of Illinois, including the city of Urbana, without a proper permit. However, smaller pyrotechnic devices such as sparklers, smoke devices, or trick noise makers can be legally used in Illinois without any kind of permit. More details about these laws can be found in Illinois' Pyrotechnic Use Act [10] and the city of Urbana's firework policies [11].

Legal concerns aside, our project does encourage the use of fireworks, which may be in violation of the 1st and 9th clause of the IEEE code of ethics [12]. Fireworks can prove to be very dangerous, but the purpose of our project is to make using fireworks safer for everyone by being able to set off these devices remotely with significantly less risk of harm. If this project wasn't used as intended, it can still cause violations with the IEEE code of ethics. Unfortunately, there is no way for us to govern how our project is used, but we believe that the increase of public safety due to the project's safety features being used as intended will outweigh the possible harm of its misuse.

5.4 Future Work

Although our project was able to meet all our requirements successfully, there is still room for improvement. To start, our controller could use a standard 5:32 MUX instead of the 1:32 serial MUX that we used in our design. We would be able to switch inputs on a 5:32 MUX much faster than with the serial MUX, which would have lowered the amount of processing needed for checking the buttons on the controller. The lowered processing requirements of polling a 5:32 MUX and further optimizations to our controller code would also increase the refresh rate of the controller's LEDs and hex display. This would reduce the amount of noticeable flickering in the controller's user interface. Another solution that has been suggested to us, is to replace the physical controller entirely and replace it with a mobile application. A mobile app replacement would significantly reduce our production costs, but more software security measures would need to be added to our communication module.

One point of improvement for the receiver would be to replace the 12V li-ion battery with a less volatile type of battery. These batteries can pose a fire hazard with improper usage due to high energy densities coupled with the flammable organic electrolyte [13]. Having a li-ion battery approximately tree meters away from active fireworks is a safety concern, even if it is protected by casing, and it's something that should be addressed.

Additionally, a revision needs to be made to both our receiver and sensor pods' PCB designs. In the PCBs we used in our current build we didn't add a capacitor to work as a stabilizing low-pass filter to support our PIR motion sensors. Luckily, we were able to add these capacitors using our programming breakouts, but these should be included normally in the PCB design. We would also want to experiment with other detection methods besides the PIR motion sensors we used. Our PIR motion sensors can successfully detect motion but they are not able to detect if someone is standing still in the launch deck area. If we move forward with this project this is another safety concern that we would address.

References

[1] U.S. Consumer Product Safety Commission, ‘2016 Fireworks Annual Report’, 2017. [Online]. Available: https://www.cpsc.gov/s3fs-public/Fireworks_Report_2016.pdf?t.YHKjE9bFiabmirA.4NJJST.5SUWIQJ [Accessed: 19- Sept- 2018]

[2] Andrew G. Simpson, ‘Facts About Fireworks: 11,000 Injuries, 4 Deaths in 2016’, *Insurance Journal*, 2017. [Online]. Available: https://www.insurancejournal.com/news/national/2017/06/30/456213.htm [Accessed: 19- Sept- 2018]

[3] Cobra Firing Systems, ‘Handheld Remotes’, [Online]. Available: http://www.cobrafiringsystems.com/remote.html [Accessed: 19- Sept- 2018]

[4] Cobra Firing Systems, ‘Modules’, [Online]. Available: http://www.cobrafiringsystems.com/modules.html [Accessed: 19- Sept- 2018]

[5] Firefly, ‘Product Details’, 2018. [Online]. Available: https://shootfirefly.com/pages/product-details [Accessed: 19- Sept- 2018]

[6] Murata Electronics, ‘Data Sheet Pyro Electric Infrared Sensor Fresnel Lens’. [Online] Available: https://www.murata.com/~/media/webrenewal/products/sensor/infrared/datasheet_pir.ashx?la=en [Accessed: 4-Nov-2018]

[7] Dejan Nedelkovski, ‘Arduino Wireless Communication – NRF24L01 Tutorial’, *How To Mechatronics*, 2017. [Online] Available: https://howtomechatronics.com/tutorials/arduino/arduino-wireless-communication-nrf24l01-tutorial/ [Accessed: 3-Nov-2018]

[8] Rex Plastics, ‘How much to Plastic Injection Molds Cost?’, 2013. [Online] Available: https://rexplastics.com/plastic-injection-molds/how-much-do-plastic-injection-molds-cost [Accessed: 8-Dec-2018]

[9] Department of Electrical and Computer Engineering, ‘Salary Averages’, 2018, [Online]. Available: https://ece.illinois.edu/admissions/why-ece/salary-averages.asp [Accessed: 3-Oct 2018]

[10] Illinois General Assembly, ‘(425 ILCS 35/) Pyrotechnic Use Act’, 2009. [Online]. Available: http://www.ilga.gov/legislation/ilcs/ilcs3.asp?ActID=1635&ChapterID=38 [Accessed: 15- Sept- 2018]

[11] City of Urbana, ‘Fireworks’, 2009. [Online]. Available: https://www.urbanaillinois.us/residents/fire-safety/fireworks [Accessed: 15- Sept- 2018]

[12] IEEE.org, ‘IEEE Code of Ethics’, 2018. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html [Accessed: 15- Sept- 2018]

[13] University of Washington, ‘Lithium Battery Safety’, 2018, [Online]. Available: https://www.ehs.washington.edu/system/files/resources/lithium-battery-safety.pdf [Accessed: 15-Oct-2018]

Appendix A Requirements and Verifications Table

Block	Requirements	Verification	Verification status (Y or N)
Voltage Regulators	1. Reduce any voltage > 5V to 5V +/-5% 2. Reduce any voltage > 3.3V to 3.3V +/-5%	1. Verification for Requirement 1 & 2 a. With a power supply, apply 6V to the input pin of the regulator and connect GND to GND b. Probe the output pin and GND pin with a digital multimeter and read output c. Confirm that output on multimeter is within 5(3.3)V +/-5%.	Y
	3. Voltage regulators operate continuously for over 1 hour under 125 degrees Celsius	2. Verification for Requirement 3 a. Apply 6V to each regulator. b. Run circuit for 1 hour c. Measure temperature of each regulator and confirm it is less than 125 degrees Celsius	Y
Power MOSFETS	1. The MOSFETs can handle the required 12V +/- 1.2V & 1A +/- 0.1A needed to set off the igniters	1. Verification for Requirement 1 a. Tie the drain of the MOSFET's to a 12V source b. Attach a resistor from the MOSFET's source pin to GND c. Toggle the voltage at the gate of the MOSFET between VCC (5V) and GND and probe the voltage drop across the resistor d. Confirm the voltage drop and current	Y
MUX	1. Take in all 32 different button signals and output the signal of the numbered button corresponding to the binary representation sent from the microcontroller.	1. Verification for Requirement 1 a. Tie all inputs to VCC (5V) b. Connect the selection data lines and output to the microcontroller c. Switch one input line to GND d. Observe if the corresponding binary number is represented with the LEDs e. Check for all 32 possible inputs	Y
Microcontroller (Controller)	1. The microcontroller recognizes input from the MUX	1. Verification for Requirement 1 a. Tie output and selection lines of the MUX module to GPIO pins on the microcontroller	Y

	2. Microcontroller can be programed to follow the logic in **Figure 2.2.1** 3. Microcontroller can send data to and from the communication module	b. Program microcontroller to cycle through all 32 possible input lines c. Tie one input line from the MUX to GND d. Program microcontroller to output the binary number of the data line that was tied to GND to a bank of LEDs 2. Verification for Requirement 2 a. Write code following the logic in **Figure 2.2.1** b. Test software behavior given hard coded inputs 3. Verification for Requirement 3 a. Setup 2 communication modules with one using this microcontroller b. Attach a button to one of the microcontroller's GPIO pins for both microcontrollers and have it send a signal through the communication module when pressed c. Have the receiving communication module's microcontroller light an LED if it receives a message	Y Y
Microcontroller (Receiver)	1. Microcontroller can be programed to follow the logic in **Figure 2.2.2** 2. Microcontroller can read inputs detect spikes in motion sensor data (check if current reading has a difference greater than a specified threshold from the previous reading) 3. Microcontroller can send data to and from the communication module	1. Verification for Requirement 1 a. Write code following the logic in **Figure 2.2.2** b. Test software behavior with hard coded inputs 2. Verification for Requirement 2 a. Power motion sensor and connect its output pin to a GPIO pin for our microcontroller b. Program microcontroller to detect a significant change between readings and power a small LED circuit when one is detected 3. Verification for Requirement 3 a. Setup 2 communication modules with one using this microcontroller b. Attach a button to one of each microcontroller's GPIO pins and have it send a signal through the	Y Y Y

		communication module when pressed c. Have the receiving communication module’s microcontroller light an LED if it receives a message from the sending RF IC	
Microcontroller (Pods)	1. Microcontroller can read inputs detect spikes in motion sensor data (check if current reading has a difference greater than a specified threshold from the previous reading) 2. Microcontroller can properly send data to the Communication Module	1. Verification for Requirement 1 a. Power motion sensor and connect its output pin to a GPIO pin for our microcontroller b. Program microcontroller to detect a significant change between readings and power a small LED circuit when one is detected 2. Verification for Requirement 2 a. Setup 2 communication modules with one using this microcontroller b. Attach a button to one of the microcontroller’s GPIO pins and have it send a signal through the communication module when pressed c. Have the receiving communication module’s microcontroller light an LED if it receives a message from the sending RF IC	Y Y
7-Segment Display	1. The 7-segment display must be able to display the value given to it by the microcontroller, using the same frequency as the LEDs	1. Verification for Requirement 1 a. Program a microcontroller to send a test message to the hex screen using 15 GPIOs. Use 4 of those GPIO pins to cycle through the common anode pins and the others to specify which segments to light b. Verify that the intended message is displayed on the hex display	Y
Buttons	1. Must be easily pressable	1. Verification for Requirement 1 a. Press button and ensure that it can be done without strain	Y

LEDs	1. The microcontroller must remember what buttons have received success signals and have the corresponding LED light up. 2. The microcontroller must make 100 cycles every second, since there are four rows for every cycle, we need the switching rate to be at least 400 per second.	1. Verification for Requirement 1 a. Program microcontroller to decode binary representation of the button number [0-4] b. Wire output pins to LEDs in series with resistors tied to GND c. Send binary signals through microcontroller and check that it lights the corresponding LED 2. Verification for Requirement 2 a. Connect microcontroller to LED circuit in Figure XXXX b. Program the microcontroller to have at least one LED on per row c. Observe LED array for any sign of flickering and that each LED can be lit up independently	Y Y
RF Module	1. Communicate reliably (95% success rate) at a range of at least 150 ft with another communications module. 2. Use control unit microcontroller to perform error checking on received data packets.	1. Verification for Requirement 1 a. Set up 2 communication modules 150 ft away from each other b. Send sample data from one module to the other 20 times c. Verify sample data isn't changed 95% of the time 2. Verification for Requirement 2 a. Setup communication with another RF IC b. Use CRC checksum library c. Send random n-bit signals to the other RF IC and check that they are flagged as errors 95% of the time	Y Y
Motion Sensors	1. IR motion sensors must be able to detect movement from 8 ft +/- 1 ft away.	1. Verification for Requirement 1 a. Power sensor with 5V Vcc b. Use an oscilloscope to check the analog signal and see if there are fluctuations when we create motion 8 ft in front of the sensor	Y

Table A Requirements and Verifications

Appendix B Work Schedule

Week Of	Michael	Trent	Daniel	Checkpoints
10/8	Solidify power solution for lighting fuses	Research how to program microcontrollers	Start wireless communication tests	Order first round of parts
10/15	Start Eagle PCB design for controller and receiver	Test PIR motion sensors for responsiveness	Continue wireless communication tests. Start Eagle PCB design for sensor pods	
10/22	Finish first iteration of controller PCB design	Determine the sensitivity of the sensors and design method for determining motion	Finish first iteration of sensor pod PCB design	First round of PCB orders (10/25)
10/29	Finish first iteration of receiver PCB design	Start programming microcontrollers (load bootloaders/test programs)	Test serial MUX	
11/5	Convert Controller and receiver PCBs to breakout version	Write code for sensor pods and 7-segment display	Make final PCB adjustments. Solidify microcontroller programming procedure	Final round of PCB orders due (11/8)
11/12	Begin final designs of physical devices and printing those designs	Tune sensitivity of our sensor pod code	Implement communications error checking	
11/19	Continue printing physical designs. Assemble the controller PCB and user interface	Start programming the sensor pods	Start programming the controller and receiver	Fall break
11/26	Finish soldering parts for the controller and receiver	Complete sensor pods and verify communications with receiver	Finish and debug controller and receiver code	Mock Demo (11/27)
12/3	Final touches and debugging	Final touches and debugging	Final touches and debugging	Final Demo (12/4) Mock Presentation (12/6)
12/10	Prepare for presentation and write final report	Prepare for presentation and write final report	Prepare for presentation and write final report	Presentation (12/11) Final Report (12/12)

Table B Work Schedule