5-2014

Mobile Marksman IV

Nicolas Moreau

Mobile Marksman IV

by

Nicolas Moreau

Undergraduate Honors Thesis under the direction of

Dr. Zhou

Department of Electrical Engineering

Submitted to the LSU Honors College in partial fulfillment of the Upper Division Honors Program

May, 2014

Louisiana State University
& Agricultural and Mechanical College
Baton Rouge, LA

Table of Contents

Problem Context

As humans, one of our most natural tendencies is to find and become adept at something we love doing. However, some people are challenged with great difficulty in the pursuit of this goal. The sport of hunting is a way of life for many people around the world, and for a quadriplegic—someone completely paralyzed from the neck down—hunting is all but impossible. Although a handful of charitable organizations take handicapped hunters on guided expeditions and use what limited technology is available, such outings are extremely expensive—for both the hunter and the organization. The Mobile Marksman Project is designed to be an affordable means of returning quadriplegics to their hunting stands.

Inspiration for this project comes courtesy of Mr. Doug Ramussen—a quadriplegic mechanical engineer and outdoor enthusiast. His personal handicap and work with organizations allow him to understand the challenges handicapped hunters must confront. Three previous prototypes precede the Marksman yet still leave much improvement to be desired. The current iteration aims to design a gun platform for semi-autonomous control by a near to fully quadriplegic user as the finalized model for the Mobile Marksman. This project should also become a blueprint for others to build their own with little outside knowledge and at a total cost of less than $1,600.

Description of Document

This document will contain all design information on the Mobile Marksman IV project. This includes the main objectives of the project as well as a detailed breakdown and analysis of every subsystem that has been implemented.

Full Problem Statement

First and foremost, the Mobile Marksman must accommodate high-level quadriplegics. It is assumed that a chaperone will accompany the user and be able to perform the preliminary setup and assembly. But other than these initial procedures, the entire system must be strictly controlled by the head and mouth due to the quadriplegic's lack of mobility. This includes both aiming and firing the weapon.

Secondly, the entire platform must be reasonably simple to reproduce for the average person. Ideally this project should be an instruction guide for others to construct their very own Mobile Marksman. It must also be reasonably quick and easy to assemble once it has been built. The platform should be made of separate, easy-to-transport components that can be assembled in a desired location such as a hunting stand or firing range.

And finally, the Mobile Marksman must be safe for both the operator and fellow shooters. The control system must be fail-safe, and the firing mechanism must have some additional way of being disarmed. Safety should be the number one priority when operating a firearm, and this project is no exception.

Design Requirements

As stated before, there have been three previous attempts at the Mobile Marksman Project. Because of this, all concepts from the past efforts were considered, and the ones that worked were kept in this version. All previous concepts that were not satisfactory were redesigned, and that is where the bulk of this project lies. Figure 1 provides the main inputs and outputs of the entire Mobile Marksman IV system.

Figure 1: Level 0 Functional Decomposition

The user inputs will be used to aim and fire the weapon. They are also required to be strictly limited to head and neck movements due to the handicaps of quadriplegics. Because quadriplegics would have extreme difficulty looking down a gun sight, the system must also display an image of the current target. Lastly, the system must output a bullet to hit the target.

The Mobile Marksman IV will be operated in five different modes: Safe Mode, High Speed Mode, Low Speed Mode, Target Lock Mode, and Firing Mode. Each mode will only allow for certain control actions to take place by enabling and disabling different subsystems throughout the rest of the system. These different modes will provide safety measures and better precision for the user. Figure 2 provides a breakdown of the Mobile Marksman into its main subsystems. Each subsystem has its own restraints and requirements which will play a role in their respective concept generations and evaluations.

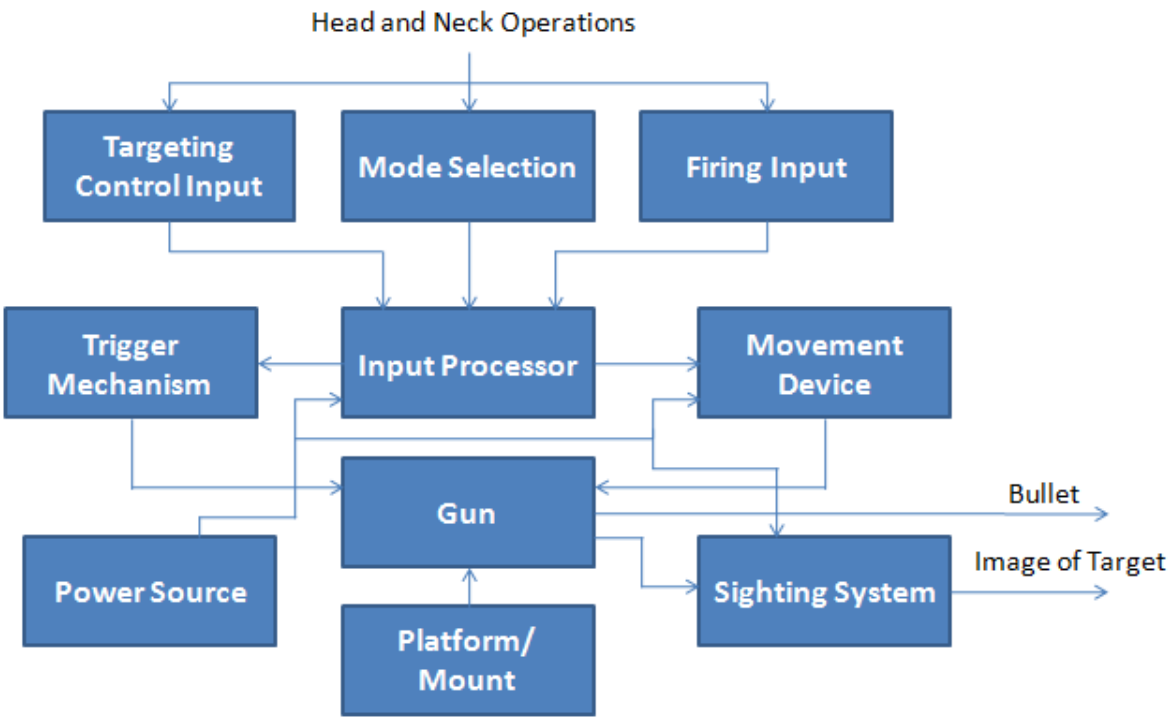

Figure 2: Mobile Marksman IV Subsystems

Targeting Control Input

Due to the limited physical capabilities of quadriplegics, a method that would allow the user to provide aiming inputs with movement exclusively from the neck or above was devised. In the previous version of the Mobile Marksman and other commercial devices, a joystick is the main method of directional input. The joystick is positioned so that it can be controlled by the user's chin. A joystick is inexpensive, and implementation is relatively simple; however, operation of the device is intrusive and difficult when used by quadriplegics. Because of this, the Mobile Marksman IV will implement a head-tracking system. This will be accomplished by securing an Inertial Measurement Unit (IMU) to the head of the user—either on the brim of a hat or inside of hearing protection. The IMU will then send measurements of the users head orientation to the Input Processor which will be used for aiming the weapon.

Firing Input

The user will also need to be able to send a command to the system to fire the weapon. This device must also be operated strictly by head or neck actions. A sip/puff switch will be used to accomplish this. This device consists of two separate switches which are operated by a tube placed in the user's mouth. Sipping on the tube closes one switch while puffing on the tube closes the other. The sip/puff switch is suitable for a quadriplegic because it is operated by mouth. The benefit of the sip/puff switch is that it allows the user to operate two different switches with only one device.

Mode Selection

Because the Mobile Marksman IV will be operated in the five different modes described earlier, there must me some sort of input device that allows the user to toggle between these modes. The sip/puff switch—which is also being used as the firing input—will be used as the mode selection input. Through different series of sipping and puffing, the user will be able to navigate through the different modes of the system. A master safety switch will also be used to send the system into Safe Mode at any time. This will be a hand switch that can be used by the chaperone to effectively shut down the entire system for any safety reason. Between the sip/puff and the safety switch, the user will be able to navigate through each mode of the system while allowing the chaperone to safely disengage the system in the case of any hazards.

Input Processor

All user inputs will need to be converted into the appropriate output, and therefore some sort of processor will need to be used. This processor will need to be capable of receiving the signals from the IMU, sip/puff switch, and safety switch and determine into which mode to move the system. The current mode will then be used to determine which other subsystems the Input

Processor will enable. This processor will effectively use the inputs to output the correct signal to the rest of the subsystems.

Trigger Mechanism

A device to pull the trigger will also need to be designed for the Mobile Marksman IV. This will be required to receive an input signal from the Input Processor and then pull the trigger. To accomplish this, a cam-lobe will be mounted to the trigger using a custom designed casing (See Figures 3 and 4). The cam-lobe will be rotated by a servo. The servo will receive the signal from the Input Processor to determine when to rotate.

Figure 3: Cam-Lobe Design

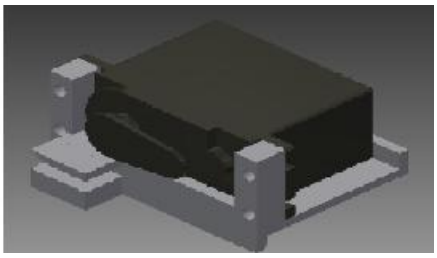

Figure 4: Trigger-Lock Device

Movement Device

Some device will need to be used to move the gun when the user sends targeting input. The Mobile Marksman III used two linear actuators—one for vertical movements and one for horizontal movements—to accomplish this. This concept will also be implemented in the Mobile Marksman IV as well.

Platform/Mount

The Mobile Marksman III introduced the idea of a freestanding platform separate from a wheelchair to support the entire system. This concept will be repurposed for the Mobile Marksman IV. A standard gun mount used for sighting rifles will be connected to the top of the platform, and the weapon will be secured to this mount. Minor modifications will be made to the mount to allow connections to the linear actuators. This mount will be required to rotate as the

actuators move to allow for aiming of the gun. The platform and mount will also be required to absorb any recoil after firing the gun.

Sighting System

Because a quadriplegic will not be capable of looking down the sight of the gun, an image of the current target must be displayed. The Mobile Marksman III mounted a camera to the scope of the rifle and fed the image from the camera to an LCD screen. This same concept will be used for this iteration of the project with minor adjustments.

Power Source

All of the system's electronics will need to be powered by a 12V DC power source. This power supply should provide enough current to power the electronics for the duration of at least 3 hours and must be stored on the platform. A portable 12V battery will be used as the power source.

System Specification at Subsystem Level

Targeting Control Input

The targeting system is directly manipulated by the user through the implementation of an Inertial Measurement Unit (IMU). This will be attached to the user's head and will be required to output two values: pitch and yaw for vertical and horizontal position. The IMU will effectively measure the orientation of the user's head. These values will be used as inputs to the Input Processor and ultimately be used in aiming the gun.

A GY-80 module will be implemented as the IMU for the Mobile Marksman IV. This device includes a gyroscope, accelerometer, and magnetic compass to ensure an accurate orientation reading. The measurements from the GY-80 are then processed by an ATmega microcontroller via the SDA and SCL serial data terminals. This ATmega can control up to 13

digital and 5 analog inputs/outputs. In this instance, the only terminals used will be the Serial data transmission lines along with the I^2C data interface pins (SDA and SCL). The ATmega is programmed to convert these readings from the GY-80 into roll, pitch, and yaw values. For wireless data transmission, two HC-05 Bluetooth modules are used—one for transmitting and one for receiving. The transmitting HC-05 module is connected to the TX and RX pins of the ATmega and set to a transmission rate of 112500 BAUD. The transmitting HC-05 module transmits the roll, pitch, and yaw values to the receiving HC-05 module connected to the Input Processor. A 9V battery is connected to a 5V regulator which will provide power to the ATmega, GY-80, and HC-05.

Firing Input and Mode Selection

The sip/puff switch that will be used for the Mobile Marksman IV is an Enabling Devices Sip and Puff Switch (Double Closure) with Gooseneck #970 (See Figure 5). This device comes with a mounting clamp that allows for mounting onto a wheelchair and a 19" gooseneck which allows the user to position the tube at almost any comfortable position. It also contains two switches: one activated by a sip and one activated by a puff. The corresponding switch closes when a sip or puff is applied to the tube, and remains open otherwise. The switch requires a moderate strength sip or puff to be activated. Because the device is not sensitive to low strength sips or puffs, it will prevent unintentional inputs from occurring.

The safety switch that will be used for the Mobile Marksman IV is an All Electronics 12V, 1" Red Lens momentary pushbutton switch (see Figure 6). This contains a 1" diameter button that can be pressed to close the switch. The switch will remain open while the button is not pressed and closed while the button is pressed.

Figure 5: Enabling Devices Sip/Puff

Figure 6: All Electronics Pushbutton

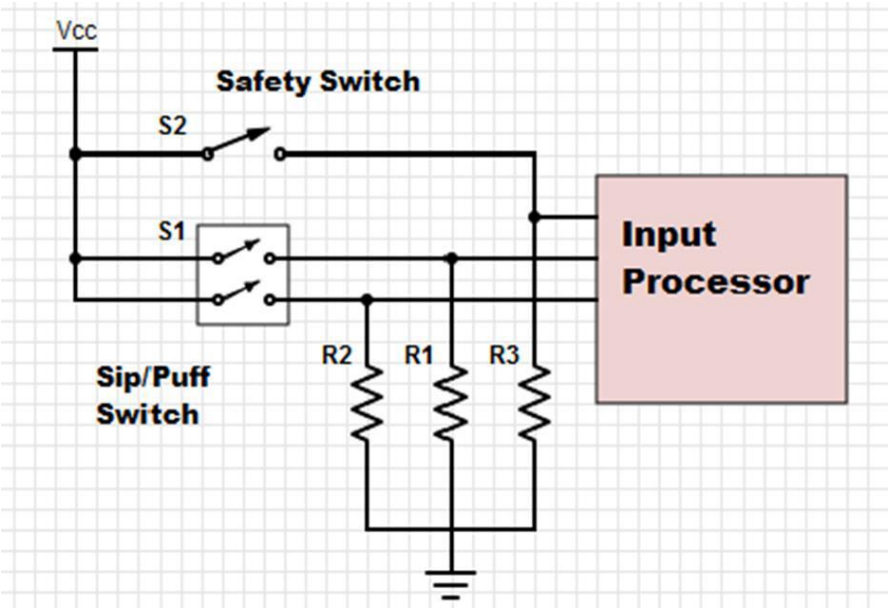

Figure 7: Sip/Puff and Safety Switch connections to Input Processor

Figure 7 shows how the sip/puff and safety switches are connected to the Input Processor. When the switches are open, a logic low is sent. When the switches are closed, a logic high is sent. This setup allows the input processor to recognize a sip or puff from the sip/puff device and a press of the safety switch.

Input Processor

All inputs of the system must flow through a centralized processor. This design implements an Arduino Mega microcontroller to accomplish this. The Arduino is connected to the receiving HC-05 Bluetooth module over the TX and RX transmission lines. The receiving HC-05 receives the roll, pitch, and yaw values from the transmitting HC-05 connected to the head tracker and relays these values to the Arduino Mega. Also, the sip/puff device and safety switch connect to the digital input terminals on the Arduino. An H-bridge motor driver is also attached to the Arduino Mega to supply power to the actuators. The H-Bridge driver contains two pairs of terminals for outputs. The trigger servo is connected to a digital PWM pin on the Arduino which is used to orient the trigger mechanism in a "pulled" or "not pulled" position.

A level zero functional decomposition of the Input Processor is shown in Figure 8. The Input Processor can be further broken down into four different modules: Mode Controller, Trigger Servo Controller, Calibrator, and Actuator Speed Controller (See Figure 9). Each of the four modules plays a key role in providing the appropriate output to the mechanical parts of the system.

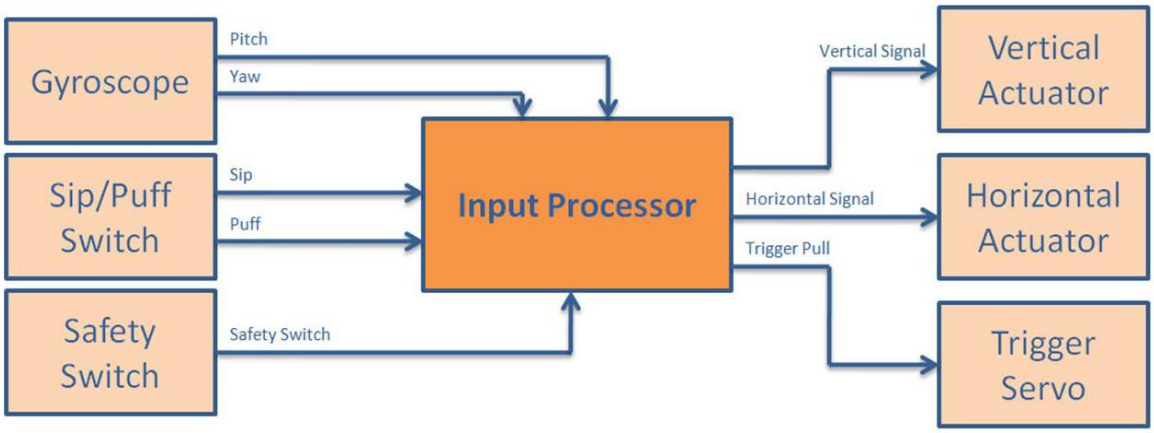

Figure 8: Level 0 Functional Decomposition of Input Processor

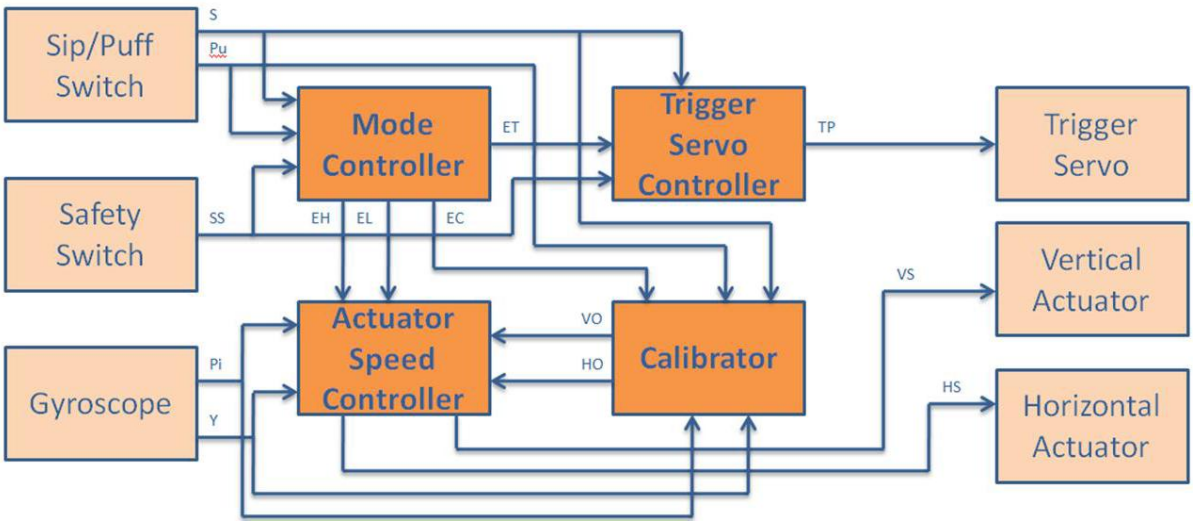

Figure 9: Level 1 Functional Decomposition of Input Processor

The first module of the Input Processor is the Mode Controller. This module receives inputs from the sip/puff and safety switch and outputs enables to the other three modules: Trigger Servo Controller, Calibrator, and Actuator Speed Controller (See Figure 10). The Mode Controller is essentially a state machine with outputs that enable the rest of the Input Processor modules. The Mode Controller effectively allows the user to operate the system to operate in five different modes: Safe Mode, High Speed Mode, Low Speed Mode, Target Lock Mode, and Firing Mode.

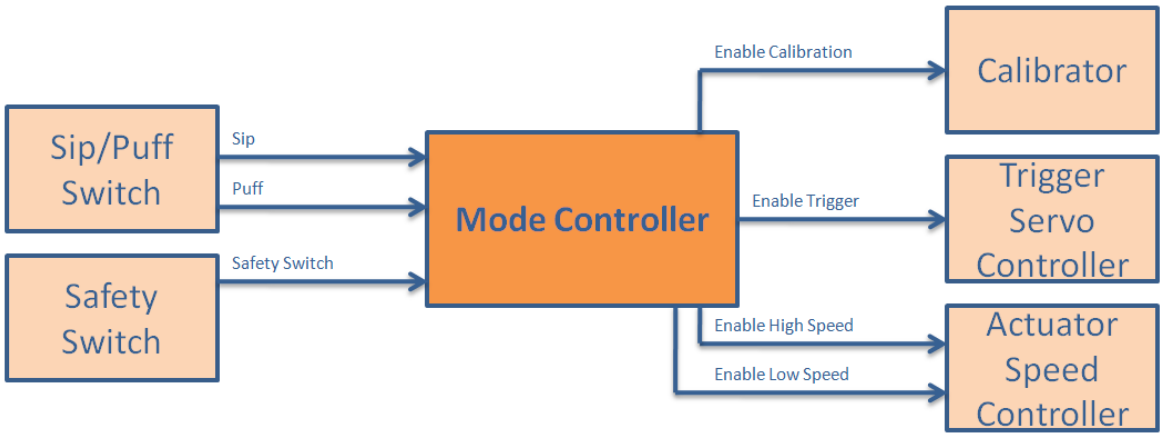

Figure 10: Mode Controller Functional Decomposition

In Safe Mode they only module that is enabled is the Calibrator. Everything else is disabled. This means that the gun cannot be aimed or fired. The only action that can take place is the calibration of the IMU.

In the High Speed and Low Speed Modes, only the Actuator Speed Controller is enabled. The trigger mechanism is still disabled. The only action that can take place is aiming the gun via user input from the IMU. The actuators will move the gun at a high speed for quick target acquisition or a low speed for precision aiming depending on the current mode.

In Target Lock Mode all modules are disabled. The gun cannot be repositioned and the trigger cannot be pulled. This mode locks the gun into the current position and waits for a confirmation from the user that it is on target. From Target Lock Mode, the user can either confirm that the gun is on target and move into Firing Mode or return to the High and Low Speed Modes to acquire another target.

Firing Mode is the only mode in which the trigger mechanism is enabled. From Firing Mode, the user can either send a final confirmation to pull the trigger or return to the aiming modes if the gun is not on target.

The Mode Controller behaves as seen in the state diagram in Figure 11. The user will be allowed to navigate through the different modes with use of the sip/puff switch. The chaperone will also be able to use the master safety switch to send the system into Safe Mode.

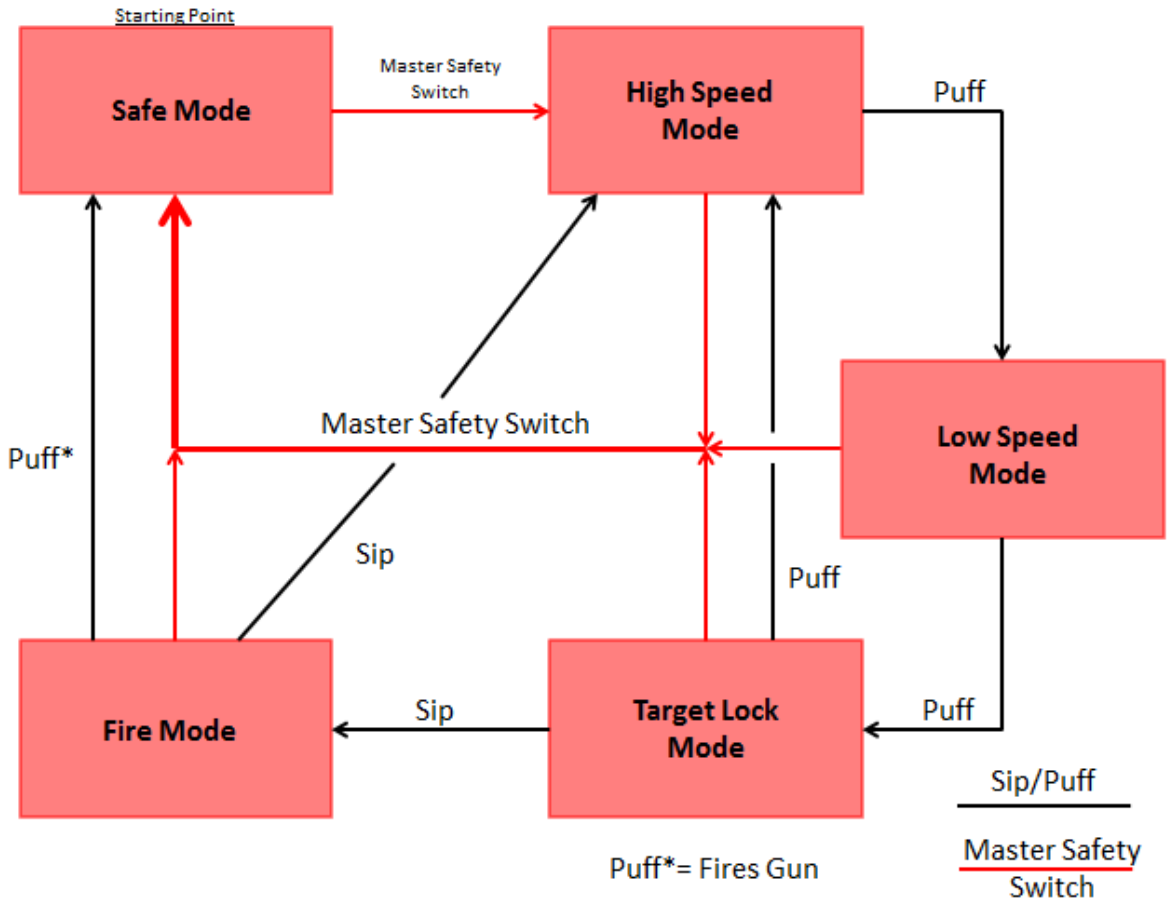

Figure 11: Mode Controller State Diagram

The second of the four modules in the Input Processor is the Trigger Servo Controller. This module receives inputs from the sip/puff device, safety switch, and Mode Controller and outputs to the servo of the trigger mechanism (See Figure 12). When this module is enabled by the mode controller, a sip from the sip/puff device will output a PWM signal to the trigger servo to move the servo into a "pulled position". The controller will then wait for 2 seconds and then output a PWM signal to move the servo back into the "not pulled" position. The module will then wait for the safety switch to reset the trigger mechanism. The Trigger Servo Controller will not allow for multiple firings without being reset by the safety switch.

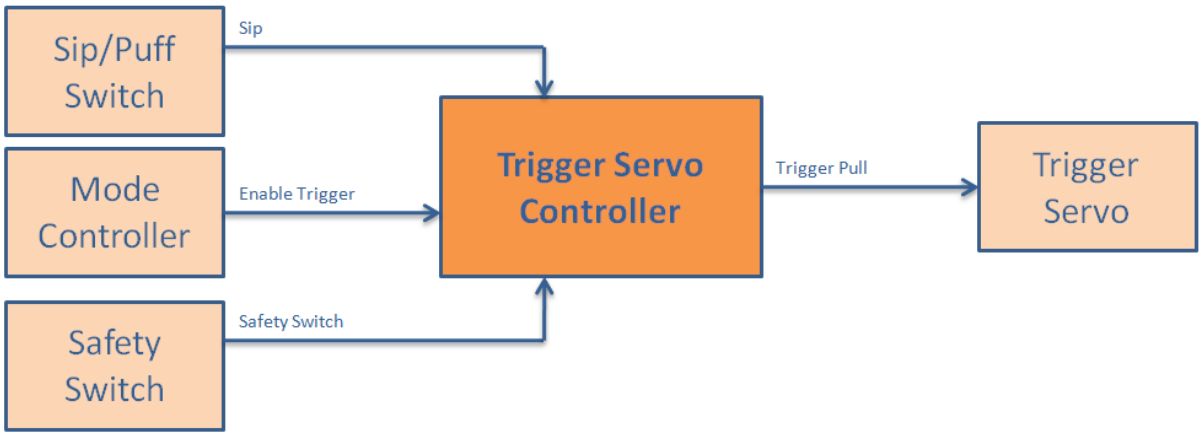

Figure 12: Trigger Servo Controller Functional Decomposition

The third of the four modules in the Input Processor is the Calibrator. This module receives inputs from the IMU, sip/puff switch, and Mode Controller and outputs an offset to the Actuator Speed Controller (See Figure 13). When this module is enabled by the Mode Controller, the current pitch and yaw values from the IMU will be sent to the Actuator Speed Controller as offsets when a sip or puff is inputted. These offset values will then be used by the Actuator Speed Controller to effectively set the position of the IMU at the moment of Calibration as the new "zero" point.

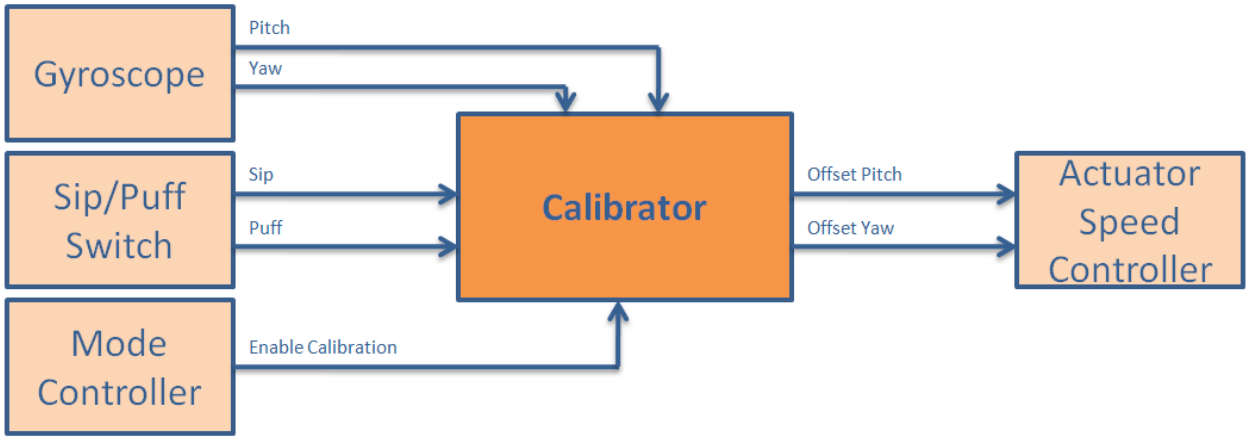

Figure 13: Calibrator Functional Decomposition

The final module of the Input Processor is the Actuator Speed Controller. This module receives inputs from the IMU, Calibrator, and Mode Controller and outputs to the linear actuators (See Figure 14).

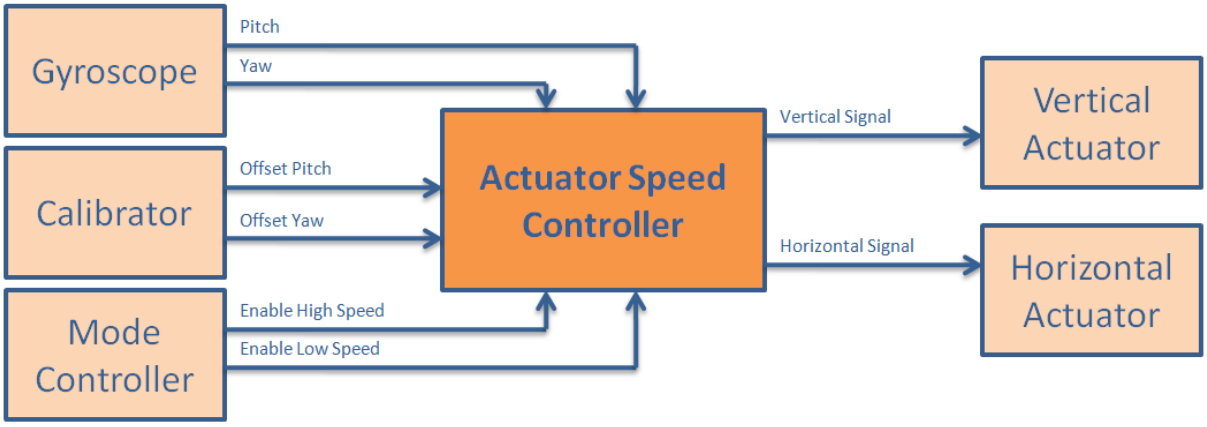

Figure 14: Actuator Speed Controller Functional Decomposition

The Actuator Speed Controller is programmed to operate as described in Figure 15. A "gray zone" concept will be used to determine how sensitive the actuators will be to head movements. When the IMU values go beyond a certain threshold of the calibrated "zero" position, the actuators will begin to move the mount in that direction. The greater the "gray zone" value, the less sensitive the actuators will be to head movements. The value for the "gray zone" can be selected depending on the preference of the user.

Local Variables	Init Values	Info
GZ	0-90 deg	Gray zone
HIGH SPEED	50-100ms	high speed delay
LOW SPEED	10-50ms	low speed delay
Calibrated pitch	0 deg	-
Calibrated yaw	0 deg	-
Appropriate signal	0	-

Inputs:
Pitch
Yaw
Offset Pitch
Offset Yaw
Enable High
Enable Low
Vert. direction
Horiz. Direction

Outputs:
Vertical Signal
Horizontal signal

Step 1:

- Calibrated pitch = pitch – offset pitch
- Calibrated yaw = yaw – offset yaw
- If (Enable High = 1) {
  Appropriate signal = high speed signal;
  }
  Else If (Enable Low = 1){
  Appropriate signal = low speed signal;
  }
  Else{
  Appropriate signal = 0;
  }

Step 2:

- Remap the calibrated pitch to range (-90 to 90)
- Remap the calibrated yaw value to range (-180,180)

Step 3:

- If calibrated pitch < -GZ
  Vert_direction = 0 (0=down, 1=up)
  Vert_output = 1 (0 = off, 1 = on)
  Else if calibrated pitch > GZ
  Vert_direction = 1 (0=down, 1=up)
  Vert_output = 1 (0 = off, 1 = on)
  Else
  Vert_output = 0;

- If calibrated yaw < -GZ
  Horiz_direction = 0 (0=Left, 1 = Right)
  Horiz_output = 1 (0 = off, 1 = on)
  Else if calibrated pitch > GZ
  Horiz_direction = 1 (0=Left, 1 = Right)
  Horiz_output = 1 (0 = off, 1 = on)
  Else
  Horiz_output = 0;

***This process exists in the main program loop, meaning it is repeated.**

Figure 15: Actuator Speed Controller Code Description

The signals shown in Figure 16 provide an example of the possible signals sent to the actuators. In High Speed Mode, the Actuator Speed Controller will send a signal with a higher duty cycle. In Low Speed Mode, the Actuator Speed Controller will send a signal with a lower duty cycle (the exact duty cycles have been determined through testing and experimentation with the actuators). In any other mode, the Actuator Speed Controller does not send a signal. Both high and low speed can never be enabled at the same time.

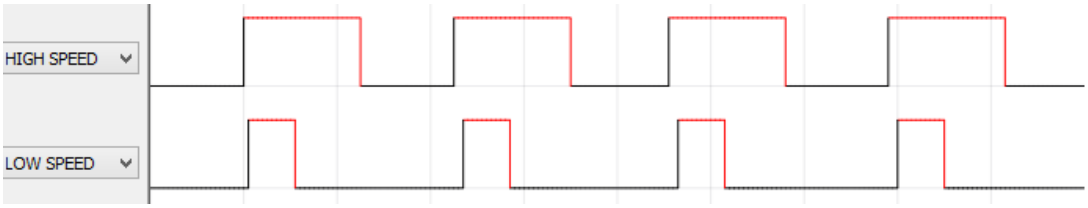

Figure 16: Actuator Speed Controller Duty Cycles

Trigger Mechanism

In order to mount the mechanism to the weapon, a custom designed servo casing was designed as seen earlier. Stainless steel nuts and bolts are used to secure the casing to both the trigger and the servo, and the cam-lobe is inserted into the casing with adequate room for movement. The HS-755MG servomotor (See Figure 17), with a shaft and cam lobe attached, is then inserted into the modified casing with the servo facing outward. The bolts are also connected to the motor in order to keep it locked into place. The servo is connected to a PWM digital output pin of the Input Processor which will be used to determine when to pull the trigger. This servo operates at 4.8-6V and draws 285mA of current when rotating.

Figure 17: HSS-755MG Servomotor

The HS-755MG servomotor outputs an 11 lb-in torque. This can be justified by noting that the trigger walk is no more than half an inch, therefore a 0.75 inch lobe is used. The torque needed to produce a 9 lb trigger pull is 6.75 lb-in. As the motor spins, it will rotate a 3/4” by 5/8” lobe, which pushes the trigger into its firing position (See Figure 18).

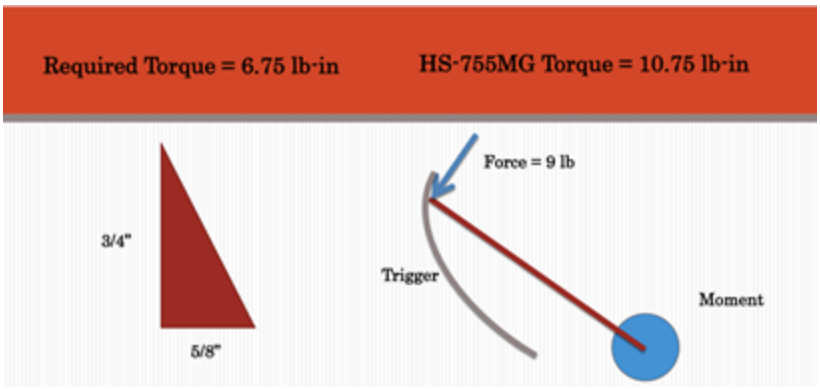

Figure 18: Trigger Servo Torque Calculations

Movement Device

Two linear actuators will be used to move the gun mount to a desired orientation. One will be used for vertical displacement and one will be used for horizontal displacement. Each actuator will receive an input signal from the Input Processor. The actuator will then move in the appropriate direction at the appropriate speed. These actuators will effectively aim the gun.

Servo City 115lb linear actuators were used in the Mobile Marksman III and are repurposed for the Mobile Marksman IV. For horizontal movement, a 4"stroke length is used. This actuator is mounted to the table using a 5/16" x 3" stainless steel bolt with a locking nut. The other end of the actuator is mounted to the gun mount using a 5/16" bolt. For vertical movement, a 2" stroke length is used. This actuator attaches to the gun mount using 5/16" bolts and locking nuts in the provided slots.

Each actuator has 115lbs thrust capacity along with a 500lb static load capacity. They operate between 6V-12V DC and run between a temperature range of -20 C˚ to +60 C˚. These Zinc Alloy actuators have an IP Grade of 65, giving total dust protection and slight water resistance. At maximum voltage, they can move 0.5" per second with no load and 0.39" per second at max load. When equipped to the mount, they project a total horizontal sweep of 36˚ (18˚ left and 18˚ right) and a total vertical sweep of 22˚ (9˚ up and 13˚ down). In testing, the

actuators achieved a minimum step size of 0.0008”, which translates to 0.96” trajectory difference at range of 200 yards. Figure 19 shows the linear actuators used.

Figure 19: Servo City Linear Actuator

Platform/Mount

The platform type selected for the Mobile Marksman IV is the freestanding platform designed in the Mobile Marksman III (See Figure 20). It is a simple design made to absorb the recoil force from the fired weapon while providing adequate support for all included units. The tabletop is made from a 2” x 6” x 14’ piece of treated lumber board that was cut into 5 pieces, each 2.5 feet long. The edges of these boards are rounded off for safety reasons. Two 2” thick pieces of treated lumber are screwed underneath the board perpendicularly for added rigidity and strength. Once the pieces were squared up, the pieces were glued and clamped together with carpenter’s clamps to ensure a tight fit. Once the tabletop was complete, five 3/8” diameter holes were drilled in the designated pattern so that the gun mount could be attached to it.

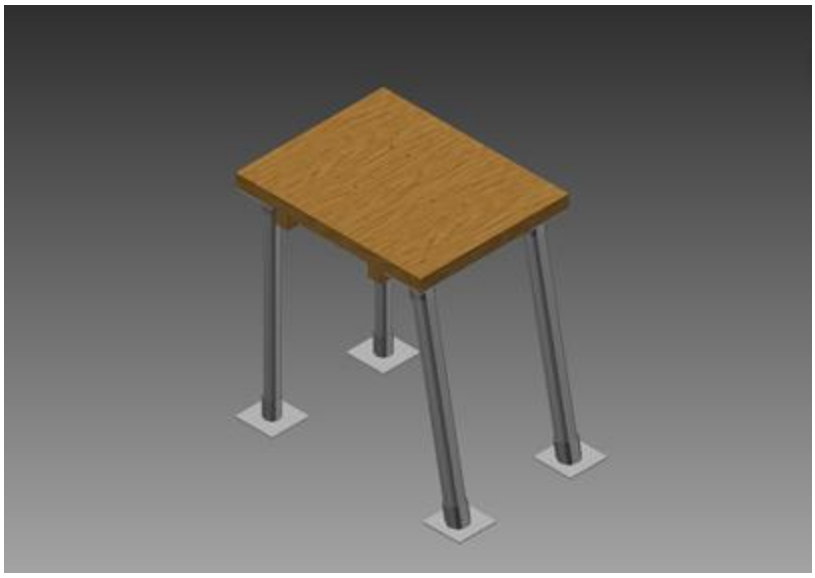

Figure 20: Platform Model

There are four 30” legs made of Aluminum 6063 and attached to the table via 2” threaded aluminum couplings. The couplings of the front legs are welded to a 6” x 6” x 1/4” flat plate, and the plates are bolted underneath the table with 0.5” coated lag bolts. The bases of each front leg are 2” coupling plates—welded to similar plates—and are screwed onto the legs. The ability to screw on and off the legs also allows for minor height adjustments with respect to ground’s surface. The rear legs have 2” couplings cut at an 11˚ angle and welded to identical plates as the front legs, which are also bolted to the tabletop. The base of the rear legs has a 2.5” schedule 40 piece of pipe cut at an 11˚ angle and welded onto a base plate of identical size as before. These back couplings are made so that they can slide on and off of the rear legs.

The weapon mount chosen for the Mobile Marksman IV is the Dead Eye Varmint Rest, manufactured by Hyskore. This mount was modified by the Mobile Marksman I group, and it was repurposed in the Mobile Marksman III. The gun cradle and dampening/spring assembly is moved forward with respect to the base plate of the mount. Actuator mounting tabs are attached to the rear portion of the gun rest for actuator implementation of pan and tilt motion. For vertical

actuator assembly, a tilt arm bracket is welded to the rear end of the dampening assembly portion of the mount. This arm is made of 1” square steel tubing and extends 4.5” horizontally towards the rear. This length provides clearance for the gun cradle, to allow the spring system to achieve full compression and maximize shock absorption. There are two rounded mounting tabs on the arm for the vertical actuator attachment with a bolt of the same inner diameter to secure it. Bushings are inserted into the actuator mounting points to increase rigidity. There are also two rounded mounting tabs welded onto the lower rotating frame of the mount used to secure the base end of the actuator. For the horizontal actuator assembly, a screw and standoff are mounted on top of the rotational portion of the frame, which secures the rod end of the actuator. The base end of the actuator is then mounted to the table platform using a 5/16” bolt and locking nut. Figure 21 shows all modifications made to the Hyskore mount; the actuator mounting tabs are highlighted in red.

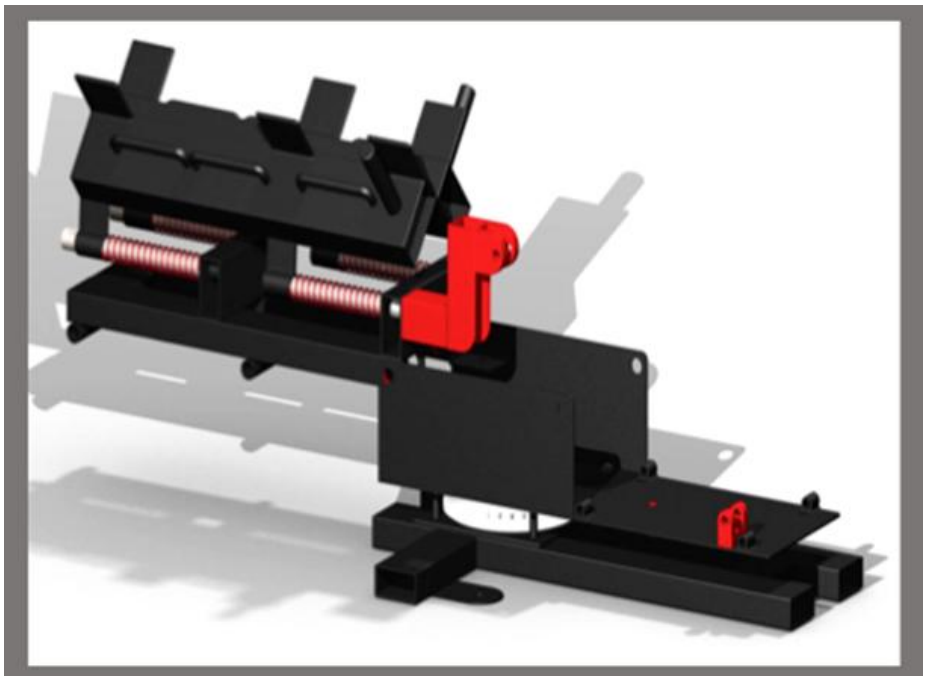

Figure 21: Hyskore Mount and Modifications

Further analysis was done on the platform and mount to ensure it could withstand the recoil forces from the gun. The first thing considered was the possibility of the table tipping over. In order to determine this, the force after dampening must be found. This is also the force that the table feels as it is pushed backwards. Without the proper tools to account for how much force the gun exhausts into the system, an alternative approach had to be taken. The Mobile Marksman III team recorded high-speed footage of the gun moving while flush against a scale (See Figure 22). As the gun is fired the scope moves in the right direction. Data points were taken at each frame and then plotted into line graphs—displaying the displacement versus time of the gun as it moves (See Figure 23). An average regression polynomial was found by averaging the input data and using the "trend line" function on excel (see Figure 24). A second order differential mass-spring-damper equation was then derived:

$$M\frac{d^2y(t)}{dt^2} + C\frac{dy(t)}{dt} + Ky(t) = F(t)$$

where M is mass, C is the damper coefficient, K is the spring coefficient, and F is the force. The mass-spring-damper equation yields a resultant force of 19.27lb (see Figure 25).

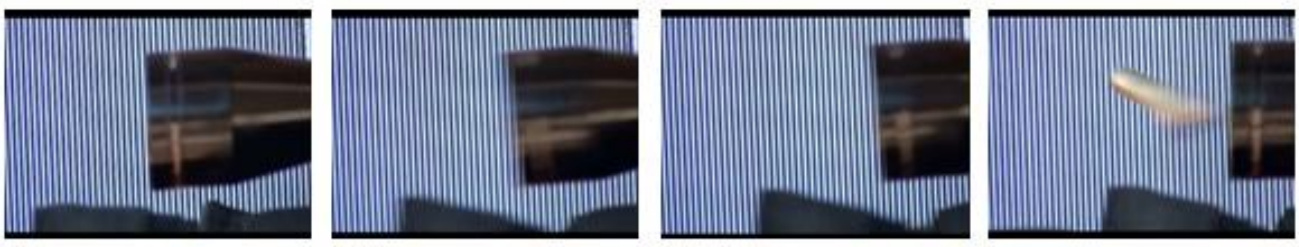

Figure 22: Rifle Recoil Test

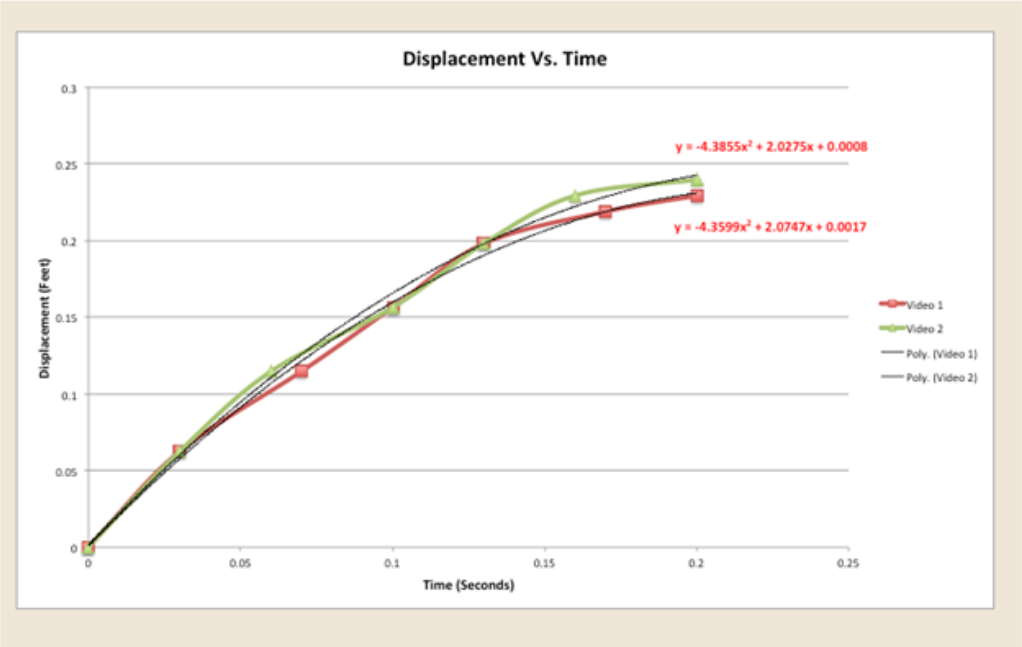

Figure 23: Data from Two Different Recoil Tests

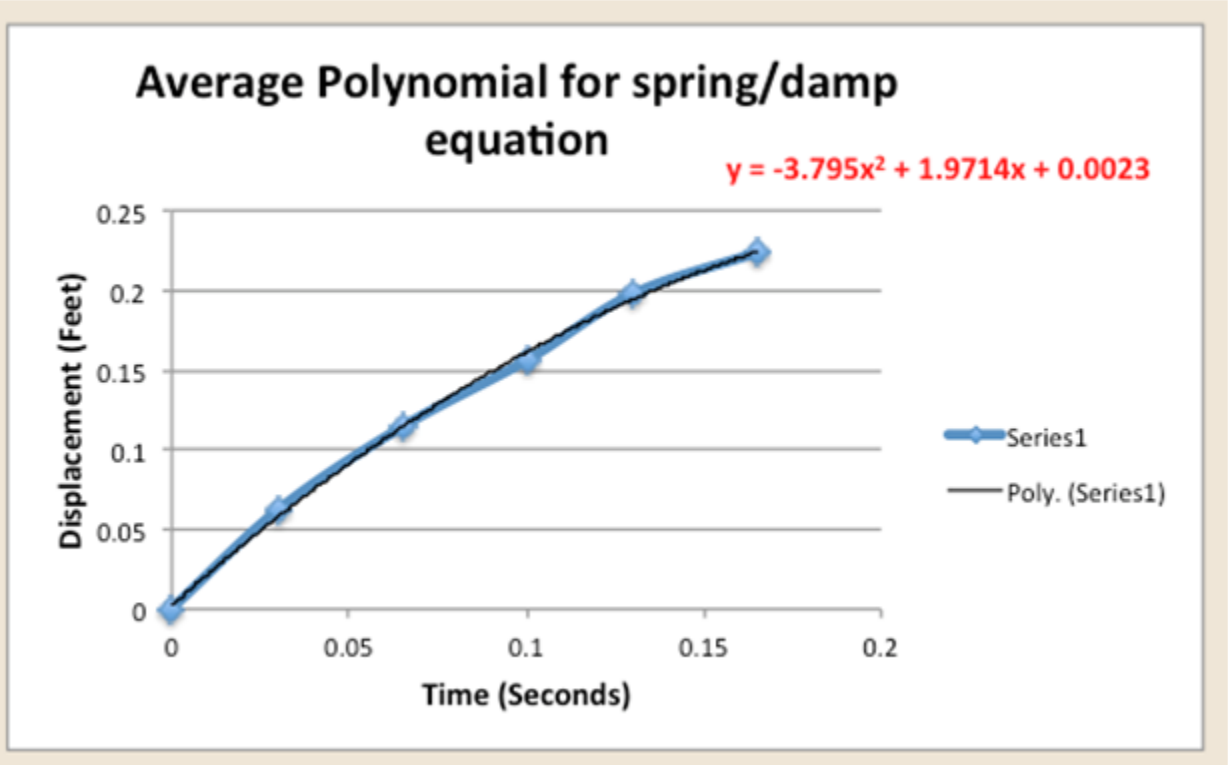

Figure 24: Average Polynomial from Two Recoil Tests

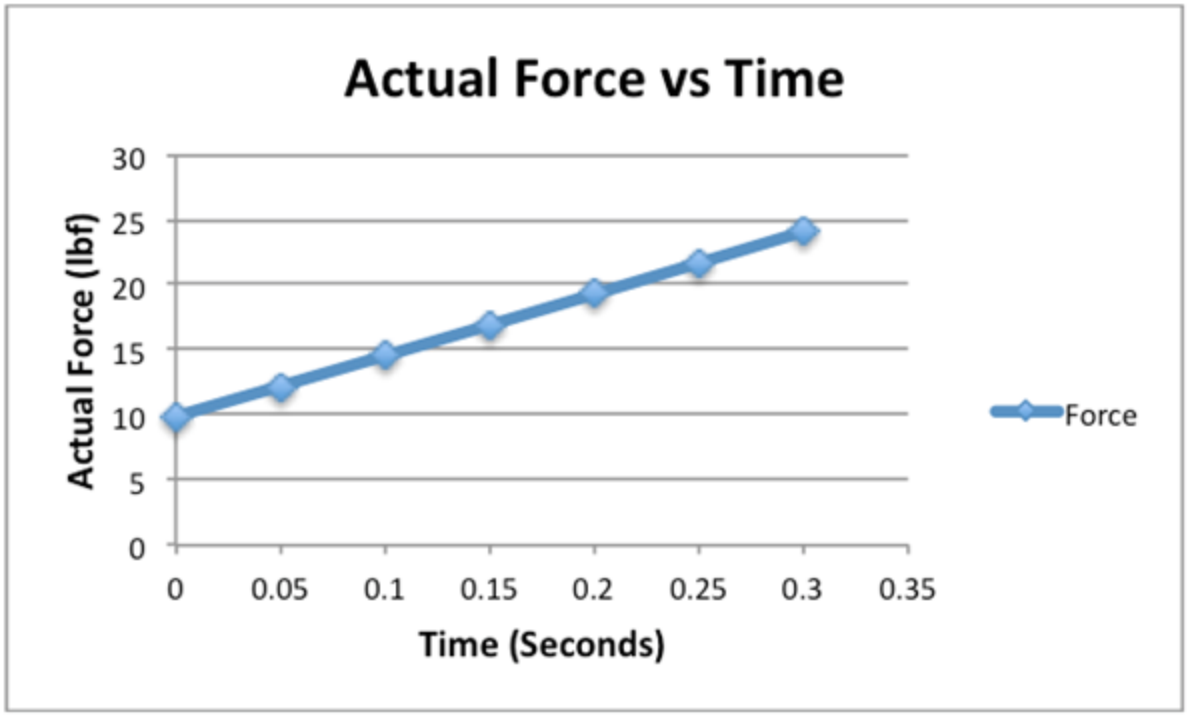

Figure 25: Calculated Force from Recoil Tests

With this force known, a static analysis is used to determine if the table will lift off of the ground during recoil. A free body diagram was used as the main reference for this static analysis (See Figure 26). A force of 23 lb was used as a conservative measurement (the actual force determined during testing was only 19.27 lb). The most important part of the static analysis is determining the reaction forces at the feet of the table. These forces are determined by summing the moment around Reaction 2 to find Reaction 1, and then summing the force in the y-direction to find Reaction 2. Figure 27 shows the results for both the steel and aluminum legs. Neither material allowed the table to lift of tip over.

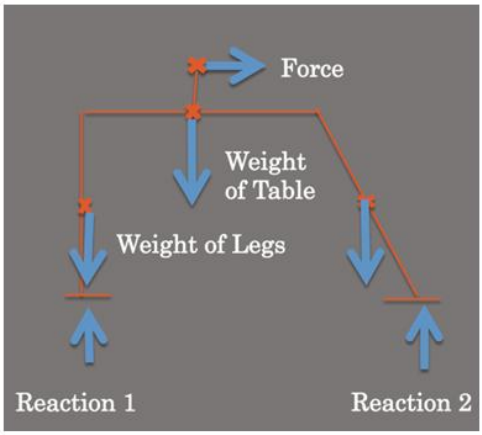

Figure 26: Free Body Diagram of Table

For Steel	For Aluminum
• Force = 23 lb	• Force = 23 lb
• Weight of legs = 20 lb	• Weight of legs = 6.67 lb
• Weight of Table = 68 lb	• Weight of Table = 68 lb
• Reaction 1 = 37.1 lb	• Reaction 1 = 22.625 lb
• Reaction 2 = 70.9 lb	• Reaction 2 = 58.735 lb

Figure 27: Force Analysis Results

It is confirmed that the table will not lift off of the ground, however it has still yet to be determined if the table will move in any direction once the shot has been fired. In order to determine if the table would move, a vibrations analysis was performed.

The system was modeled with moving masses for this analysis (see Figure 28). The HySkore mount includes four springs, one damper, and the mass of the mount. The springs were combined into one equivalent spring force. The HySkore owner's manual gives specific information about these components, including the damping coefficient and each spring's constant. The frictional component was proportional to the weight of the mass. The frictional coefficient for the system was determined to be 0.8, which is the closest resemblance of what the system would experience. Based on this model, the differential equations governing movement are:

$$m_{table}\frac{dv_{table}}{dt} = +\left[k(x_{gun}\text{-}x_{table}) + \mathrm{c}(v_{gun}\text{-}v_{table})\right] \pm \mu W$$

$$m_{gun}\frac{dv_{gun}}{dt} = F(t) - \left[k(x_{gun}\text{-}x_{table}) + \mathrm{c}(v_{gun}\text{-}v_{table})\right]$$

$$\frac{dx_{gun}}{dt} = v_{gun} \qquad \frac{dx_{table}}{dt} = v_{table}$$

These four differential equations were solved simultaneously using MATLAB to yield position information and velocity information for both masses and yielded two graphs (See Figure 29). The velocity versus time graph displays the settling time of the system. The displacement versus time graph shows that the system would only move 1" backwards due to the recoil.

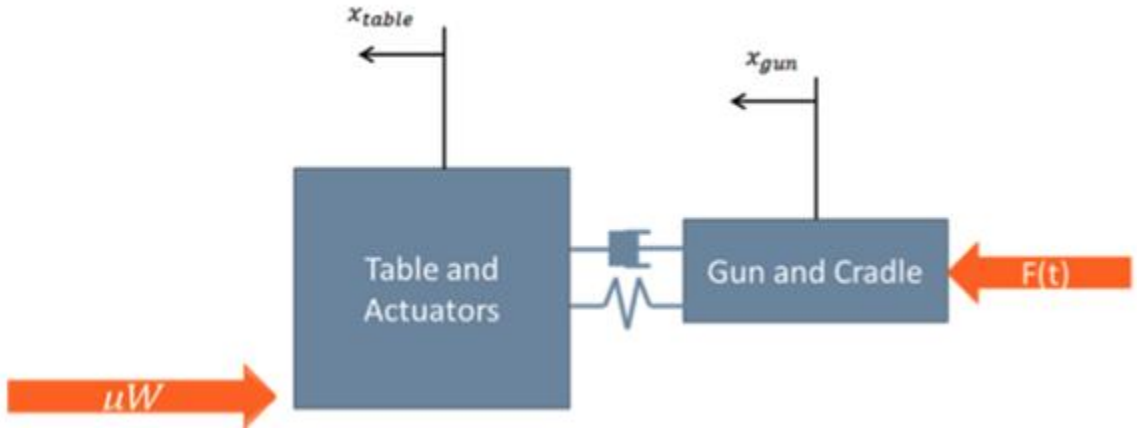

Figure 28: Spring and Damper Model of the Mount and Platform

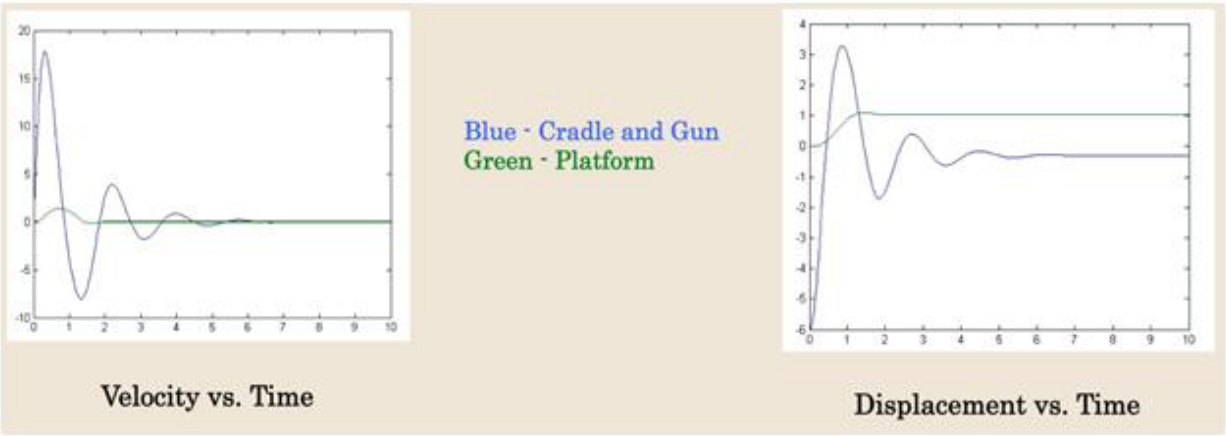

Figure 29: MATLAB Simulation Results

It can be seen that aluminum legs will not cause the system to tip over or move too far backwards. Using aluminum effectively reduced the weight of the legs by more than one-third making the system much easier to maneuver, and more importantly, does not put the user in harm's way.

Sighting System

The Sighting System concept from the Mobile Marksman III is repurposed with minor upgrades. The LCD screen used in the Mobile Marksman III proved to be sufficient during its use in field tests, but it still encountered some glare issues. An alternate LCD screen will be used for the Mobile Marksman IV: Pyle Hydra Series Marine Grade Water Resistant 7-Inch LCD Wide-Screen Monitor with an Anti-Glare Shield and Universal Stand Model PLMRM71W (See Figure 30). This 7" LCD monitor has high resolution color and is waterproof up to 3.3 feet. It also comes with a universal U-shaped stand with rotatable angle adjustment and position tension. The resolution of the screen is 800 x 400 pixels and is powered with 12-24V DC. The specific dimensions of the monitor screen are: 4.76" x 7.08" x 0.91". This component weights a total of 3.5lb. With additional features in comparison to the previous LCD screen, this display and built-in housing will resolve many glare issues present with the older version. This updated screen will be the only change to the Sighting System. Everything else will be repurposed from the previous projects.

Figure 30: LCD Screen used in Sighting System

The previous Mobile Marksman prototype used a Sony Mini 550 Res Helmet Camera (See Figure 31). It is 19 mm in diameter, 19mm in width, and 60mm in length. It is completely waterproof and is generally used as a helmet cam—meaning it can endure outdoor elements and rough usage. This device is powered by a 12V DC power source and draws only 90mA. It comes with a custom 6' cable with RCA connections to attach to an LCD screen.

Figure 31: Sony Bullet Camera

The camera mount concept from the Mobile Marksman III will be repurposed as well. The parts required to make this are all prefabricated and can be found at any local gun store. This means there is no manufacturing involved, only assembly. This mounting system requires one Picatinny rail and three scope rings. Two of the scope rings attach the Picatinny rail to the scope. Excess rail extends past the rear sight optical portion of the scope. The third scope ring is placed on this protruding portion of the Picatinny rail. The bullet camera is placed inside this third scope ring (See Figure 32). The camera must to be mounted at a distance that constitutes proper eye relief, ensuring a clear and complete image. Eye relief refers to the distance from the rear optical at which the shooter can see the target clearly. This distance varies depending on the biological nature of one's own eye, but it is estimated to be within the range of two to four inches. This range will also vary due to the curvature of the scope's lenses. The user will have to adjust the bullet camera to a distance that meets the particular scope specifications.

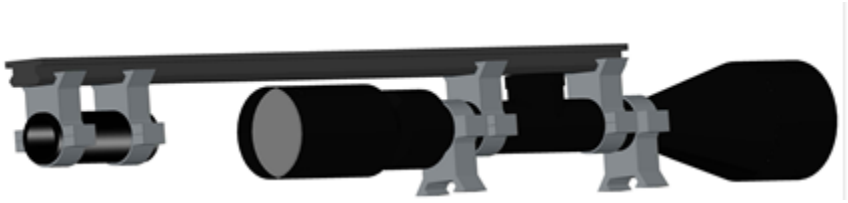

Figure 32: Camera Mount Model

A MAX7456 will be implemented into the Sighting System as well. This on-screen display controller is available for the Arduino platform. This receives data through the Serial Peripheral Interface bus to determine what to display on the screen. It also allows for the option of overlaying graphics on an existing video feed. The image from the Sony Bullet camera will be connected to the video input of the on screen display board. The Arduino will then superimpose data over the live camera feed. The data displayed to the screen will consist of the current operating mode, required actions to toggle to other operating modes, and live pitch and yaw values (See Figure 33).

Figure 33: Example of information displayed on screen

Power Source

Without the proper power source, the platform would not be able to operate correctly. The Mobile Marksman IV will use a 12V DC power source capable of providing the required amount of current. Calculations indicate that the platform will draw a maximum current of 4.03 amps during operation (see Figure 34). To be conservative, all calculations used 5 amps as the max current draw. This can be used to determine the battery life of potential battery systems. To find the battery life, the amp hour rating of the battery (provided by the manufacturer) is divided by the maximum current draw of the system. The resulting number will provide the number of hours the battery will operate at 12V. For the desired 3 hour run time, a battery with a 15 amp hour rating should be used. Other batteries with increased amp hour ratings were also considered to determine which would be the most cost effective. Some batteries examined are shown in Figure 35. A 40 amp hour battery will be used in the final design. It will provide nearly 3 times the requested battery life, while still remaining cost effective.

DEVICE	CURRENT (MAX) [A]
LCD Display	1.25
Actuator (x2)*	2
Camera	0.09
Arduino	0.03
Bluetooth	0.06
Trigger Servo	0.5
Screen Display Controller	0.1
TOTAL	4.03

Figure 34: Amp Requirements for entire system

AMP HOURS	BATTERY LIFE (HOURS)	COST	$/Hour
26	5.2	$ 55.00	$ 10.58
40	8	$ 85.00	$ 10.63
55	11	$ 130.00	$ 11.82
75	15	$ 170.00	$ 11.33

Figure 35: Battery Life Cost Comparison

Implementation Achievements

Targeting Control Input

A GY80 inertial measurement unit (IMU) sensor board is used to read the orientation of the user's head and provide directional targeting control (See Figure 36). Using the MM_TX code, (See Appendix 1) an Arduino microcontroller is able to compute pitch, roll, and yaw values from the GY80 readings. These computed pitch, roll, and yaw values are then sent to the Input Processor using an HC-05 Bluetooth module. This system will be attached to a pair of protective ear muffs to be securely fastened to the user's head.

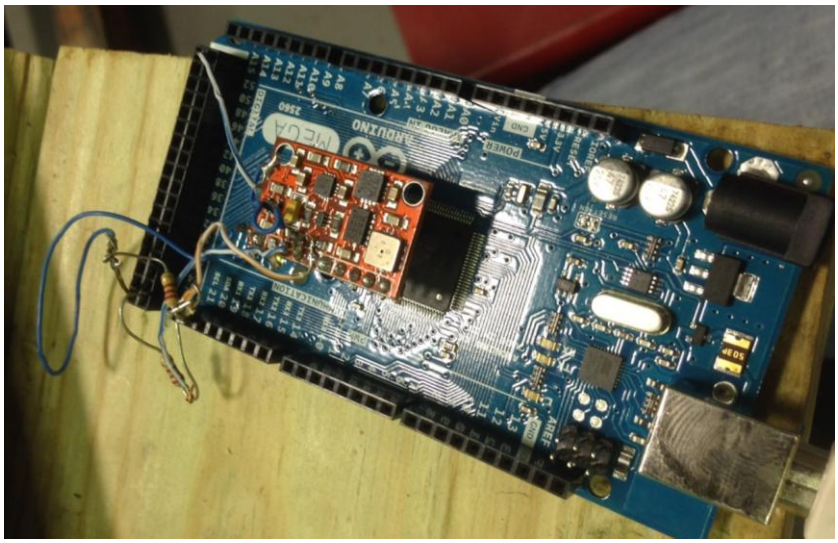

Figure 36: GY80 IMU and Arduino Connections

While setting up the IMU, some problems occurred. When the sensor board was rotated along the X-axis, the yaw readings from the compass were not correct between 90 and 180 degrees. After further inspection, it was determined that the on board compass of the GY80 needed to be calibrated. To accomplish this, the code was modified to display the raw sensor output. Using this data, the maximum and minimum values the compass were able to read were determined and were inserted into the code to provide for correct scaling along the X-axis (See Figure 37).

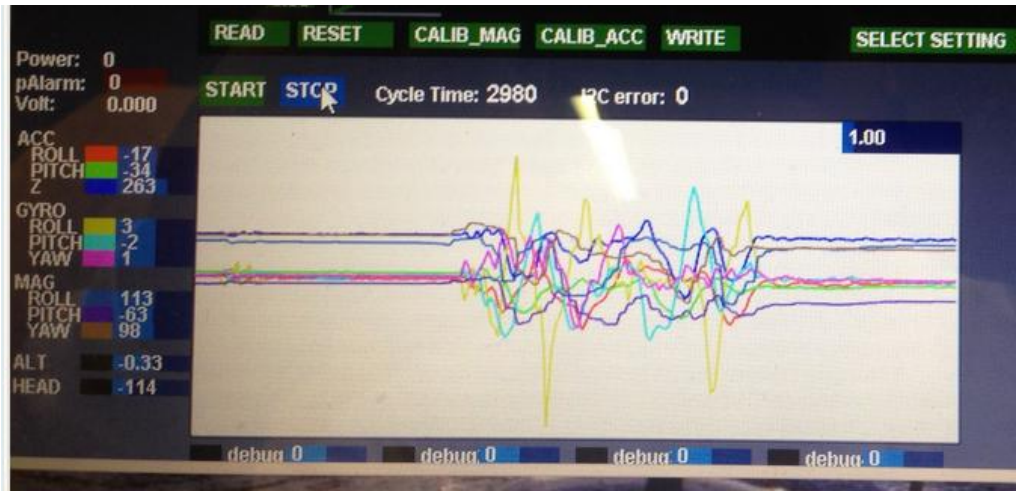

Figure 37: GY80 Roll, Pitch, and Yaw Calibrated Readings

Determining what method will be used to attach the head tracking system to a pair of protective ear muffs is left for the user to decide. Options to achieve this include: industrial strength Velcro, snap on fasteners, and rail mounting. Industrial strength Velcro seems to be the most viable option.

Firing Input and Mode Selection

The sip/puff device used in the Mobile Marksman IV is an Enabling Devices Sip and Puff with Gooseneck. Using two 3.5 mm mono connectors, this device provides two output channels—one for a sip and one for a puff. To simplify the assembly process, these two mono connectors were converted to a single stereo connector (See Figure 38). The stereo connection provides three pins while each mono connector only provides two pins. One terminal from the sip switch and one terminal from the puff switch were soldered together, and the other two terminals were left alone. The new common terminal was then soldered to a pin connected to ground, and the individual terminals were soldered to pull-up resistors. This setup allows the user to send both the sip and puff signal from a single connection instead of two separate connections as it was originally arranged.

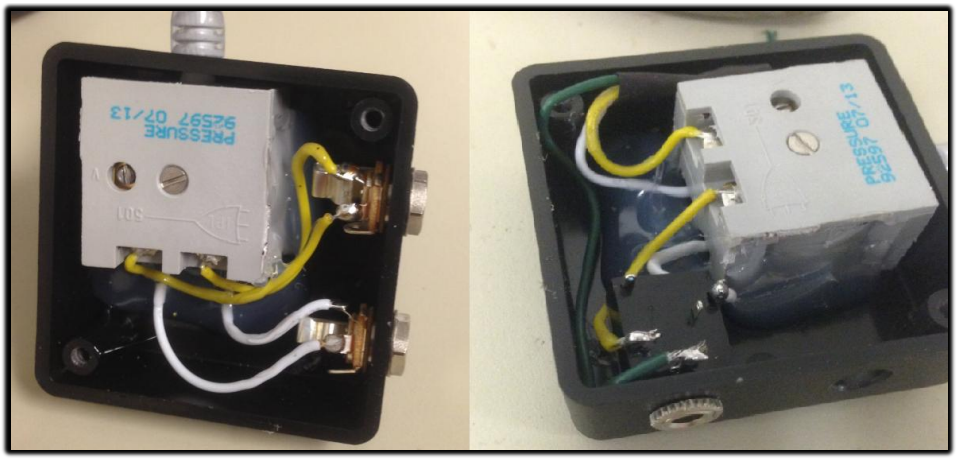

Figure 38: Conversion from Mono Connectors to Stereo Connector

These channels were connected to digital input pins of the Arduino. To ensure that the switches were connected properly and operated correctly, a testing code was written that used the Arduino serial monitor to display what action was being performed. A series of sips and puffs were then applied to the sip/puff device. The serial monitor display was then viewed to ensure the Arduino received the correct inputs (See Figure 39). The serial monitor displayed the appropriate actions which confirmed that the sip/puff device was connected properly.

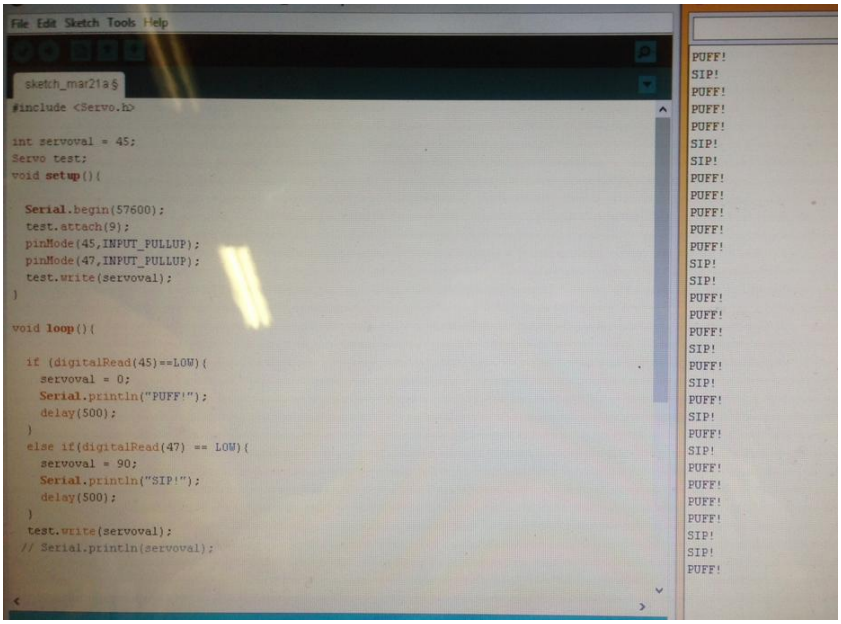

Figure 39: Serial Monitor Display of Acknowledged Inputs

Once the sip/puff switch was completed and tested, the safety pushbutton was implemented and tested in a similar way. The button was attached to a PVC pipe and capped on the other end allowing it to be hand-held (See Figure 40). A 1/4 inch audio jack was chosen as the connector for the safety switch. This is a two pin connector which is exactly what the pushbutton requires and allows the chaperone to safely stand behind the user with the button in hand.

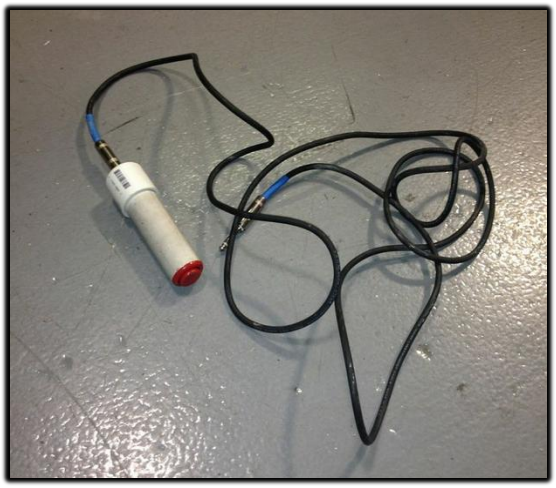

Figure 40: Safety Pushbutton Switch with PVC Pipe and 1/4" Audio Jack Connector

Both the sip/puff switch and safety pushbutton were modified to be connected using single cords and tested. Being able to simply connect these devices to the Input Processor via cords makes it very easy to assemble and makes the entire system much more user friendly.

Trigger Mechanism

To fire the weapon, a rotating servo was attached to a 3D printed mounting plate (See Figure 41). The servo chosen was an HS-755MG. A shaft and cam is attached to the servo in order to manipulate the trigger upon rotation.

Figure 41: Trigger Mounting Plate and Servo

In early testing, the servo was successfully interfaced with the Arduino Input Processor (See Figure 42). This servo uses PWM signals to determine what angle to orient the cam, and the Arduino has designated pins of outputting these PWM signals. Once the servo was connected to the Input Processor, the Arduino was programmed to orient the servo at different angles, and the servo was observed to ensure it was connected properly and operated correctly. The servo was offset to the appropriate angle each time, confirming successful implementation.

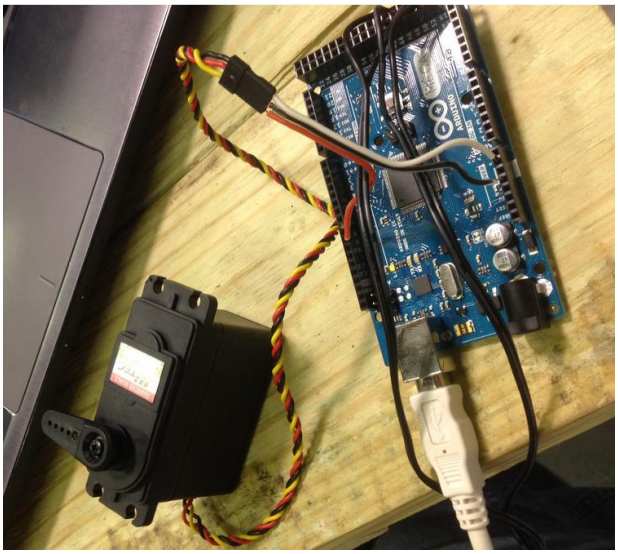

Figure 42: HS-755MG Connection to Arduino

The servo requires three pin connections to operate: one for ground, one for supply voltage, and one for the PWM orientation signal. To allow easy assembly, a USB connector was chosen to connect the servo to the Input Processor (See Figure 43). USB connectors allow for four pin connections and are very common. This provides more than the necessary pins and makes connecting the servo to the Arduino extremely simple.

Figure 43: Servo with USB Connector

The servo mounting plate was manufactured using a 3D printer. This mount is constructed of ABS plastic, and is designed to fit a wide range of firearms (See Figure 44). To hold the mount in place, a small screw is used to bind the mounting plate to the weapons trigger guard. Also, a small bolt and nut are used on the outer edge to clamp the guard in place.

Figure 44: Trigger Mechanism attached to Various Weapons

This configuration has been tested successfully, with only a few malfunctions. On higher caliber weapons, the trigger mount may slide up the guard after repeated fire. Minor modifications have been made to improve the stability of the mounting plate for these higher caliber weapons, but during use, the chaperone may need to reposition the mechanism after multiple uses to ensure proper operation.

Input Processor

This system is implemented using an Arduino Mega microcontroller, H-bridge motor driver, and MAX7456 video overlay chip (See Figure 45) with the MM_RX sketch (See Appendix 2). The Arduino wirelessly receives head orientation data from the IMU using an HC-05 Bluetooth module. It also receives inputs from the sip/puff device and pushbutton safety switch using separate digital input pins.

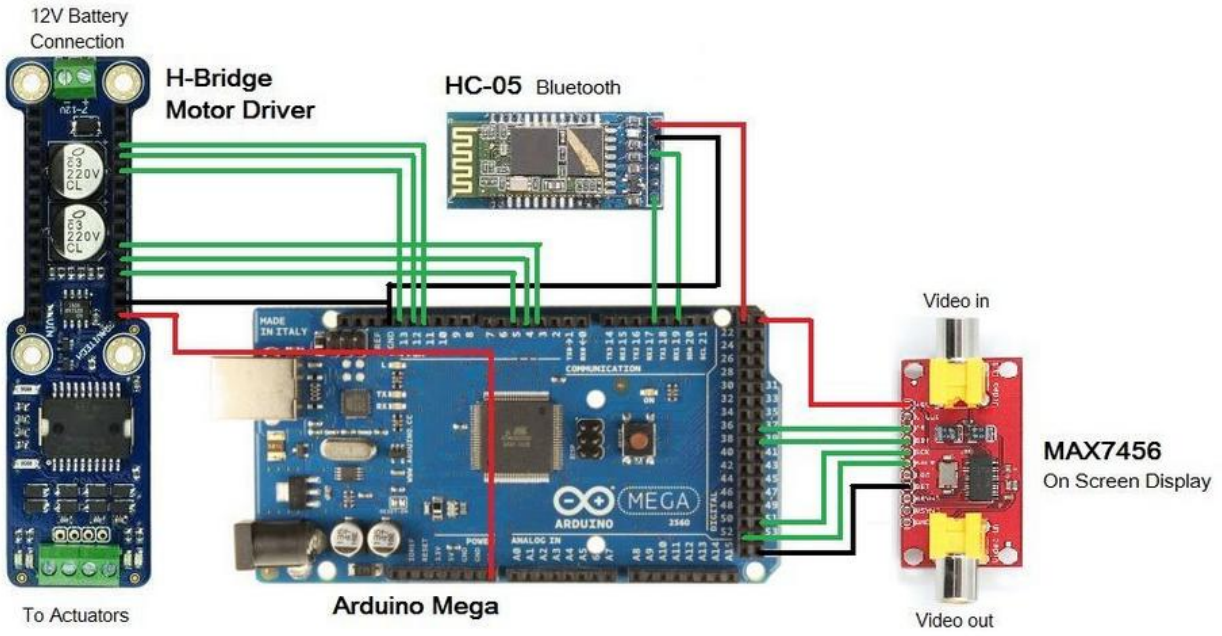

Figure 45: Input Processor Components and Connections

The Arduino Mega outputs to the MAX7456 serial on screen display chip, which overlays text and images onto an RCA video signal received from the camera. This is used with an LCD display to indicate the current mode and provide data for operating the system.

The Arduino Mega also outputs a signal to the H-bridge motor driver to control actuator movement. This board provides up to 2A of current for each actuator and provides variable speed and direction control that is manipulated by using the Arduino PWM channels. Based on the IMU orientation readings, the Input Processor will determine which direction the actuators need to move and then send the correct signal to the motor driver.

Finally, this system outputs a signal to the trigger servo to rotate to the correct position. To accomplish this, the servo library of the Arduino and a PWM channel are used to send position data to the trigger servo. The only time the servo position is set to pull the trigger is when the system is in Firing Mode. At all other times, the servo position is set in the "not pulled" position. This is critical for safety requirements.

A control box which provides different connectors for each input and output device connected to the Input Processor was created (See Figure 46). This box contains a connector for the sip/puff device, safety switch, actuators, trigger servo, video in, video out, joystick, and power source (See Appendix 3 for a table with each device and its respective connector). All these connectors are hard wired to the appropriate pins on the Arduino, H-bridge, and video overlay chip. The control box also contains a power switch that either applies or cuts power to the entire system and six LEDs. A large blue LED next to the power switch indicates when the Bluetooth connection is established with the head tracking IMU. Five LEDs—one green, two blue, one yellow, and one red—are aligned in the middle of the control box. These LEDs indicate when the system is in Safe Mode, High Speed Mode, Low Speed Mode, Target Lock Mode, and Firing Mode respectively.

Figure 46: Control Box with all Connectors

Once all the devices were connected to the control box, field tests were performed to ensure that all the connections were proper implemented. Everything worked as designed. The control box made assembling and making all the interconnections extremely quick and easy.

Movement Device

The system needs to have a sweep range of 18 degrees in each horizontal direction. Also, the vertical sweep range needed to achieve 9 degrees upwards and 13 degrees downwards. Using these requirements, the correct mounting points and actuator lengths were calculated. The horizontal actuator was chosen to have a 4 inch stroke, and was mounted 7 inches from the pivot point to provide less than a 1 inch movement for a target 100 yards down range. The vertical actuator was chosen to have a 2 inch stroke (See Figure 47). These met the desired range and precision constraints.

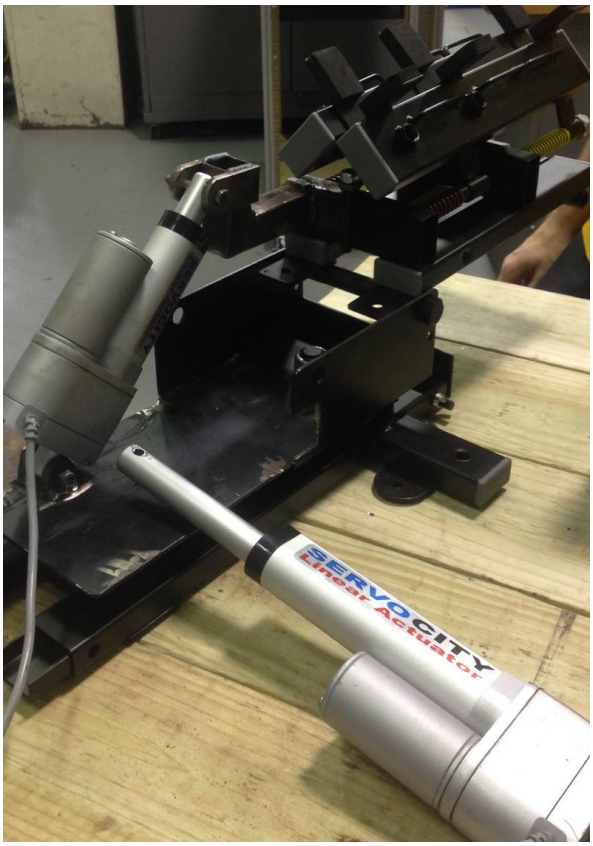

Figure 47: Actuator Sizes and Placement

Both the horizontal and vertical actuators were connected to H-bridge motor drivers which were connected to PWM outputs of the Arduino Mega Input Processor (See Figure 48). The H-bridge allows the Input Processor to determine which direction to move the actuators, and the length of the PWM pulse determines how fast the actuators will move. Different PWM signals were sent to the Actuators to ensure that they were connected properly and operated correctly, and the speed and direction of the actuator movements were observed. The actuators moved at the appropriate speeds and directions confirming successful implementation.

Figure 48: Actuator and H-bridge Connections

When securing the actuators to the table and mount, the diameter of the bolts was slightly smaller than the diameter of the hole. This created a lag period when first activating the actuators. To eliminate this, bushings were used to fill the small gap between the bolts and holes. This minimized the slack in the system, and increased the platforms overall precision.

For speed control, the Arduino was used to output a PWM signal to the actuators. For high speed, PWM values of 105 and 68 were used for horizontal and vertical movement

respectively. This corresponds to approximately 41% and 27% duty cycles. To create a low speed mode, the same PWM signals are sent to the actuators for a set period of 20ms. Then for a set period of 50ms, no signal is sent to the actuators. After this time, the signal is sent for 20ms and the whole process repeats. This creates a precise stepwise movement of the actuators.

An eight pin XLR connector was chosen to connect the actuators to the Input Processor (See Figure 49). Each actuator requires four pins to operate. Two pins provide position data of the actuator, and the other two are used for ground and supply voltage. With two actuators, eight pins are required. The XLR connector provides the correct number of pins and can also handle the power needed to move the actuators. This makes it an ideal connector.

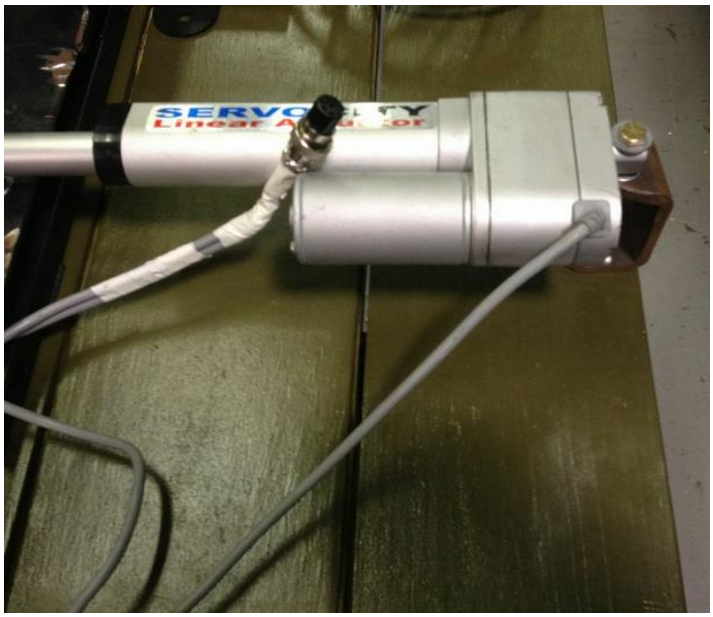

Figure 49: XLR Connector Attached to Actuators

Sighting System

The screen and camera have been purchased and tested. The Picatinny rail assembly from the last project was used to mount the camera to the scope. Once the camera was properly mounted and connected to the LCD screen, the image was displayed (See Figures 50 and 51).

This confirmed that the screen and camera are compatible and will be able to provide the image of the target to the user.

Figure 50: Camera Mounted to Scope with Picatinny Rail

Figure 51: LCD Screen Connection to Camera

Additional tests have been done with the new screen to ensure appropriate picture quality. The previous Mobile Marksman team conducted various quantitative tests to decide the overall quality of the screen. The test pertaining to glare issues was also conducted with the new Pyle LCD screen. The test consists of placing five 8”x11” sheets of paper with various symbols (See Figure 52) 200 yards down range. Volunteers were then asked to identify each symbol, and were

awarded points for correct responses. This test was done at different times throughout the day to ensure that the glare from the sun would not cause an issue with the screen.

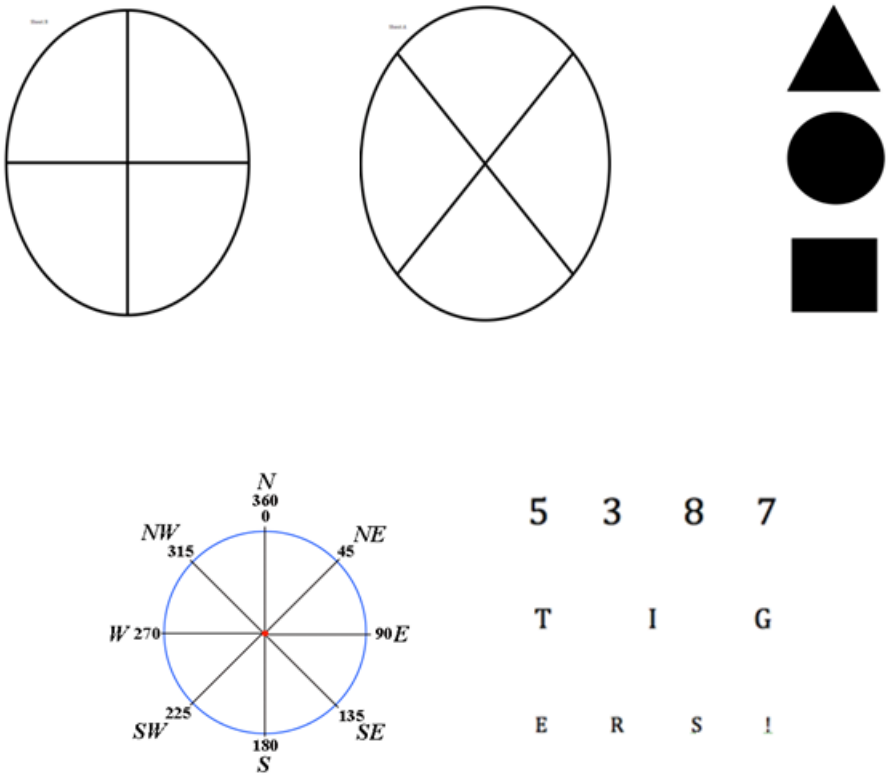

Figure 52: Sighting System Testing Sheets

The tests were successful. There was little difficulty for the volunteers to view the symbols. The most difficult time to view the screen was at noon, which is understandable. But even then, the glare was not over burdensome. Because hunting generally takes place in the early morning, the system will be used mainly when the sun is low and glare issues are minimized. This is another advantage for the screen because it will typically not have glare issues.

The video overlay was applied to the RCA signal from the camera using the MAX7456 chip (See Figure 53). The information overlaid onto the signal includes the current mode of the system and what a sip or a puff will do. This information makes it so that the user will always be

aware of what the system is and will be doing base on his next inputs. Users should still practice and review how to operate the system before actually firing the weapon. The information is very helpful, but cannot completely inform the user if he is not familiar with what it means. Safety is the number one priority, and the better the user understands how to operate the Mobile Marksman, the safer he will be.

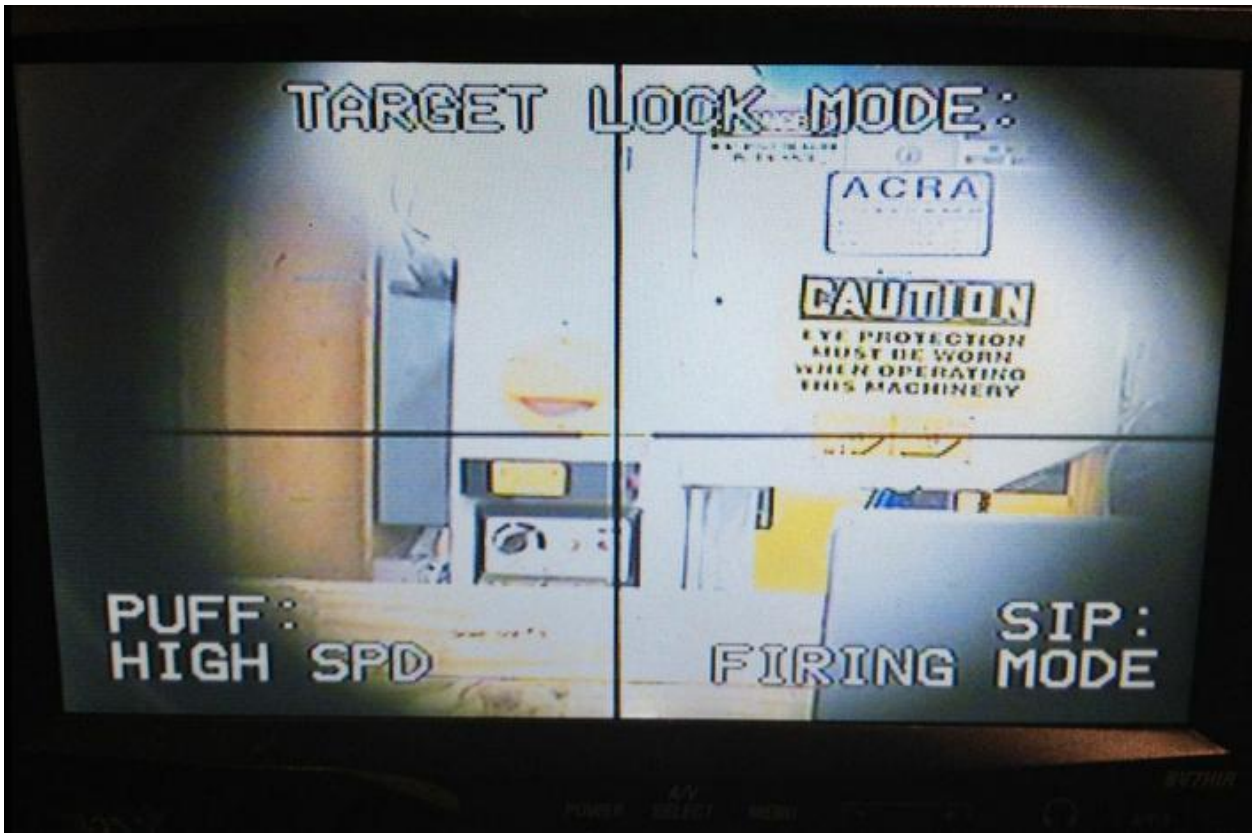

Figure 53: Information Overlay onto LCD Screen

Platform/Mount

The tabletop was constructed from five 2.5' x 2" x 6" pieces of treated lumber (See Figure 54). Two support boards were then industrially glued and screwed perpendicularly to the bottom of the tabletop to connect and support each board (See Figure 55). These boards were 2" x 2.5" and cut to the width of the tabletop. They were installed 6 inches from the edge to leave

room for the aluminum plates for the legs. Each support board took eleven screws for proper stability.

Figure 54: Tabletop Construction

Figure 55: Support Board Construction

The Dead-Eye Varmint Rest developed by Hyskore was purchased and modifications were made in accordance to the Mobile Marksman III report. The first modifications were done to accommodate the horizontal linear actuator. All of these modifications were done to the central piece of the Dead-Eye mount. When the actuator pans to aim right, the right side wall plate makes contact with the actuator causing it to jam. Because of this a portion of the plate was cut out to provide clearance for the actuator (See Figure 56). A hole was also drilled in the center of the base plate, 7 inches behind the pivot point (See Figure 57). This hole is used to secure the horizontal actuator to the mount.

Figure 56: Wall Plate Clearance Modification

Figure 57: Base Plate Horizontal Connection Hole

The next modifications made to the mount were made to accommodate the vertical linear actuator. The dampening assembly needed to be moved forward and attachments needed to be welded onto the mount for actuator installation. The dampening assembly's mounting ring, which is used to attach the dampening assembly to the rotational central piece of the platform, was removed from the center of the assembly and welded to the rear (See Figure 58). This allows space for the vertical actuator to be installed between the central piece and the dampening assembly.

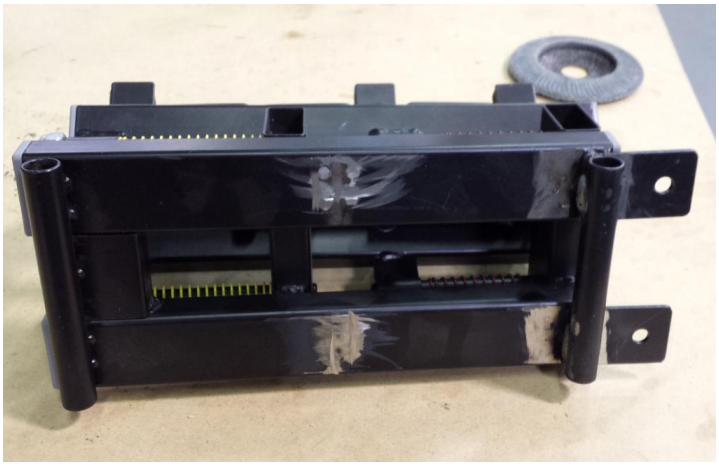

Figure 58: Mounting Ring Removed and Welded to back of Dampening Assembly

Next, mounting tabs (See Figure 59) and an L-shaped arm (See Figure 60) were created to be welded to the mount. These modifications provide locations for the vertical actuator to connect to the mount. The L-shaped arm was made of 0.25 inch squared tubing and is 1.25 inches tall and 4 inches long. These dimensions were used so the dampening assembly would be able to absorb the maximum amount of recoil without the arm or actuator interfering with movement.

Figure 59: Mounting Tabs

Figure 60: L-Shaped Mounting Arm

Once the tabs and arm were constructed, the longer tabs were welded to the arm and the arm was welded to the left-rear of the dampening system (See Figure 61). The smaller tabs were welded to the central piece of the mount 4 inches from the wall plate (See Figure 62). These modifications provide the two locations for the vertical actuator to secure to the mount around the vertical pivot point. Figure 63 shows the mount with all modifications.

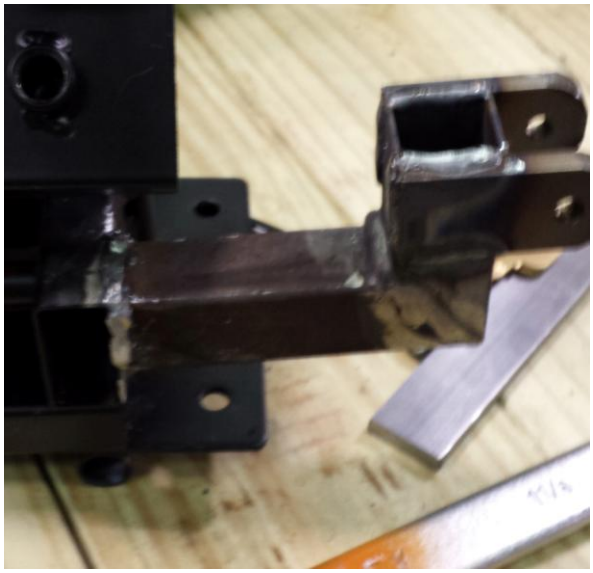

Figure 61: Arm Welds to Tabs and Dampening Assembly

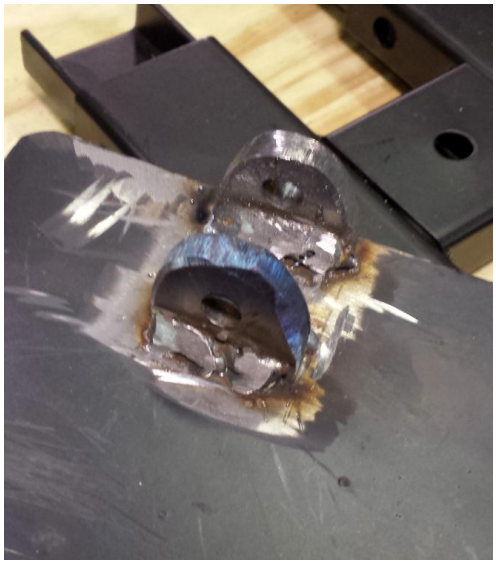

Figure 62: Tab Welds to Central Plate

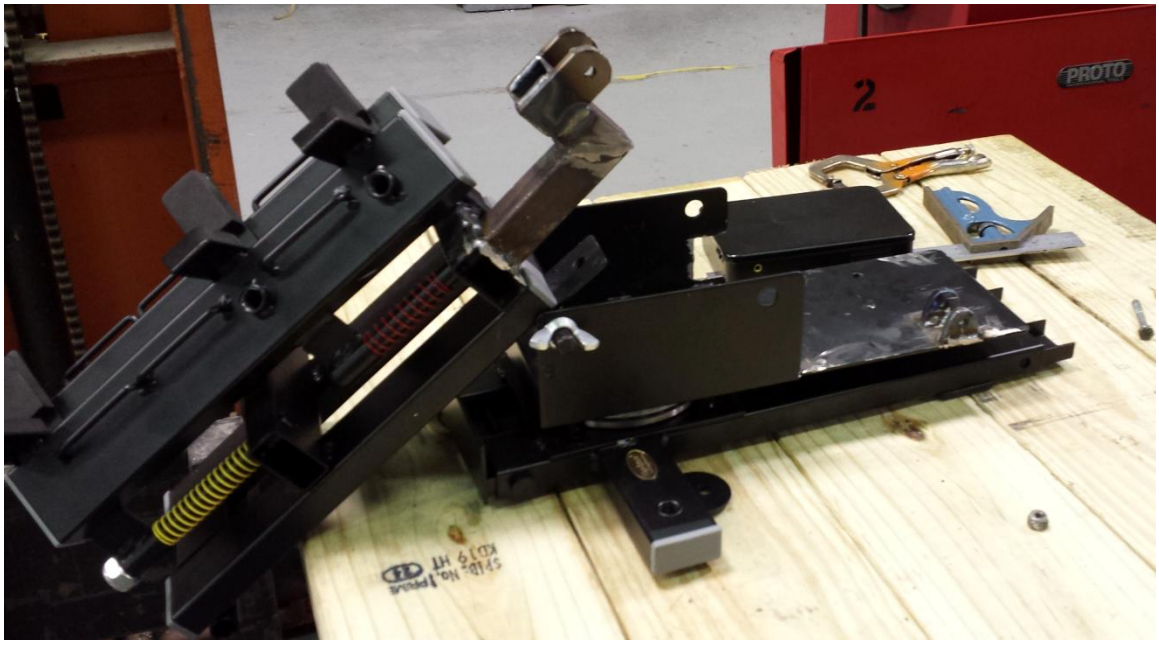

Figure 63: Completed Modifications to Mount

Once the tabletop and mount were completed, the tabletop was painted and a tab for the horizontal actuator was added (See Figure 64). This could only be done after the mount modifications were complete to get an accurate placement for the tab. This tab allows the actuator sit completely parallel with the tabletop and prevents any jamming.

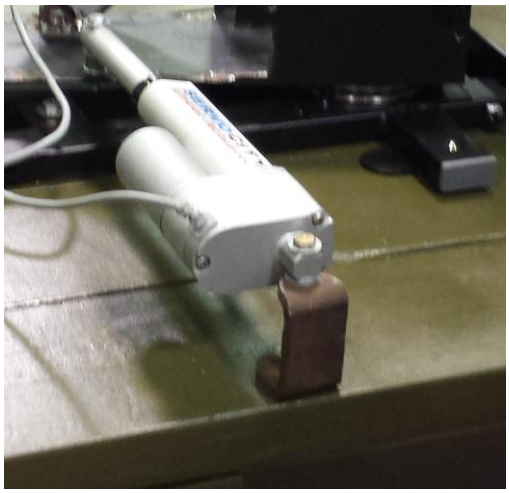

Figure 64: Horizontal Actuator Tabletop Tab

The last requirement for the Platform/Mount was the creation of the legs. Thes specifications were adopted from the Mobile Marksman III prototype with a few modifications. Instead of steal, 6063 Aluminum is used to reduce the overall weight. Eight base plates used for the leg sleeves were cut to 6” x 6” x 1/4”. For the front legs, four 6” pipe sleeves were welded perpendicularly to the base plates. For the rear legs, four 6” pipe sleeves were welded at an 11˚ angle to the base plates. This created eight feet for the legs (See Figure 65). Four of the feet were secured underneath the tabletop in its four corners. The other four feet were threaded to allow the legs to be screwed into them. This threading allows provides a way to adjust the length of the legs by simply screwing or unscrewing which will be necessary when placed on uneven surfaces.

Figure 65: Aluminum Feet for Legs

After the feet were created, four 30” aluminum pipes with outer diameter of 2” were cut for the legs. Each leg received 6 inches of thread on one end (See Figure 66). This end will screw into the feet that will be placed on the ground. The other end of the legs was not threaded which allows the legs to simply slide into the feet connected underneath the table. Figure 67 shows the entire platform and mount after completion.

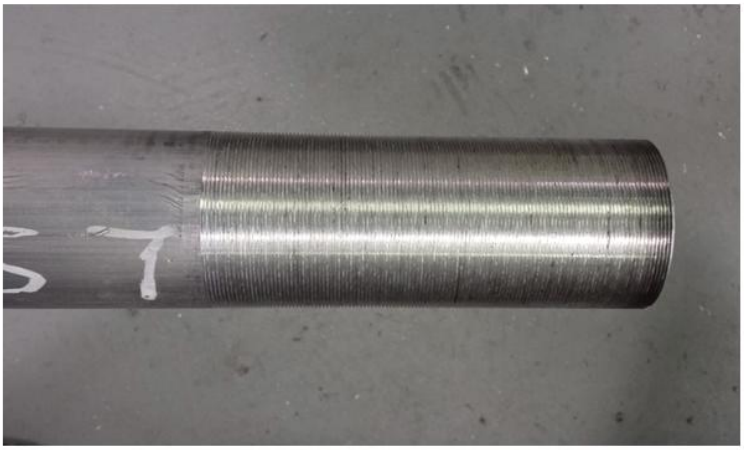

Figure 66: Threading on Legs

Figure 67: Finalized Platform and Mount

Power Source

To power the system, a deep cycle 12V battery will be used. The entire system was calculated to operate at a maximum current of 3.43 Amps. Using this calculation, it was concluded that a 40 amp-hour battery would be sufficient. This will provide a battery life of 11.5 hours, and this battery can also be recharged using a standard trickle charging system. The battery will be connected to the input processor using two alligator clips wired to a standard 12V socket connection (See Figure 68). This connector has an in-line 5A fuse to protect from any power surges. The input processor then distributes the power to the remaining subsystems.

Figure 68: 12V Battery Connector

IRB

One of the main goals of the Mobile Marksman IV Project is to have quadriplegics test the system once it has been completed. To obtain permission for this, an Institutional Review Board (IRB) proposal was submitted. The original proposal was to allow quadriplegic testers to operate a fully functional Mobile Marksman; this included firing a gun. This application was

denied for expedited approval on the grounds of great risk of discharging a firearm. To counter this, two more proposals were submitted. One was for expedited approval and included only using a laser pointer and scope in lieu of a gun. There would be no firearm discharge of any sort during the testing and would only test if the quadriplegic users could operate the head tracking and sip/puff devices. The other application was for the full IRB approval process which requires going before a full IRB board (a much slower process). This one includes the use of a firearm during testing. Since there is some minor risk of injury, a consent form was created for all potential testers.

IRB approval for testing without a firearm was received and some testing was done with four different quadriplegic volunteers. Approval for testing with a firearm will not be received before the end of the semester, and therefore no shot groupings can be received from the volunteer testers. However, fire testing will be done internally, and if it can be shown that the system can meet the accuracy requirements, and quadriplegics can operate the system, then it can be concluded that quadriplegics will be able to meet the accuracy requirements.

Grant

It was discovered that the NRA Foundation denied the grant request for the Mobile Marksman in June of 2012. The NRA was contacted in hopes that the old grant could be altered to be used for this semester's project. After weeks of negotiations with the NRA, the grant was finally approved, but the funds would not be received until June of 2014. Fortunately, the ME department was willing to bankroll the project until the grant money was received. Unfortunately, this was not completed until March 10, 2014, almost two months into the semester and caused a major delay of manufacturing.

Budgeting Analysis

The cost of creating one Mobile Marksman was kept under $1,600 (See Figure 69). This was one of the major goals of the project. The sponsor requested for a completely new Mobile Marksman to be built and for the old prototype to be updated. This required extra funding which came out to a total of $2,749.23 (See Figure 70). Everything has successfully remained under budget.

Mobile Marksman IV			
Item	Quantity	Price/Item	Totals
Scope Rings	2	11.32	22.64
Picatinny Rail	1	41.98	41.98
Sony Bullet Camera	1	189	189
LCD Screeen	1	64.95	64.95
2x6x12	1	7.97	7.97
2x6x10	1	6.97	6.97
2" Galv. Conduit	4	11.87	47.48
2" couplings	2	4.15	8.3
6"x6"x1/4" Plate	8	3.45	27.6
Hyscore Mount	1	250	250
Actuators	2	130	260
Battery Box	1	20	20
Battery	1	30	30
Arduino Micro	1	29.99	29.99
Gy80 gyroscope	1	14.99	14.99
Hc05 Bluetooth Module	2	5.99	11.98
Arduino Mega	1	34.99	34.99
H-bridge motor driver	1	19.99	19.99
Sip Puff Device	1	173.95	173.95
HS-755 MG servo	1	39.99	39.99
Plus Materials			325.88
Total Price of MM4	1		1302.77

Figure 69: Cost of Mobile Marksman IV

Mobile Marksman IV			
Item	Quantity	Price/Item	Totals
Scope Rings	2	11.32	22.64
Picatinny Rail	1	41.98	41.98
Sony Bullet Camera	2	189	378
LCD Screeen	2	64.95	129.9
2x6x12	2	15	30
2x6x10	2	15	30
Legs	2	135	270
2" couplings	4	10	40
6"x6"x1/4" Plate	12	5	60
Hyscore Mount	1	250	250
Actuators	2	130	260
Battery Box	1	20	20
Battery	2	30	60
Arduino Uno	3	29.99	89.97
Gy80 gyroscope	3	14.99	44.97
Hc05 Bluetooth Module	6	5.99	35.94
Arduino Mega	3	34.99	104.97
H-bridge motor driver	2	19.99	39.98
Sip Puff Device	2	173.95	347.9
HS-755 MG servo	2	39.99	79.98
Nuts Bolts Screws	1	40	40
.223 Ammo	1	100	100
30-06 Ammo	1	100	100
Wiring & Components	1	50	50
Ear Muffs (Safety)	3	25	75
Bushings	4	12	48
Total Price of MM4	1		2749.23

Figure 70: Total Cost of Mobile Marksman IV and Update of Mobile Marksman III

System Testing

Three quadriplegic volunteers agreed to participate in system testing. Due to a lack of IRB approval, they were not allowed to actually fire a weapon. Instead, only a scope was used. They were briefed on how to use the head tracker and sip/puff and given about ten minutes to become familiar with the system. Once they felt comfortable five time trials were done with each tester. A target was placed 100 yard down range. The system was placed somewhere off target, and the tester was timed to see how long he took to get on target. Three other testers (non-handicapped) went through the same time trials. Figure 71 shows the results of these trials.

Quadriplegic Time Trials			
	Subject 1	Subject 2	Subject 3
Trial 1	25	7	9
Trial 2	18	10	30
Trial 3	12	26	12
Trial 4	16	14	23
Trial 5	11	12	20
Avg	16.4	13.8	18.8
		Overall Avg	16.33333

Personal Time Trials			
	Carl	Chris	Casey
Trial 1	13	10	10
Trial 2	9	11	9
Trial 3	17	11	14
Trial 4	12	16	15
Trial 5	7	8	14
Avg	11.6	11.2	12.4
		Overall Avg	11.73333

Figure 71: Time Trial Recordings

The results of the time trials were very pleasing. The goal was to have an average on-target time of less than 20 seconds. The quadriplegic testers were able to average 16.3 seconds with all three testers under the 20 second goal. The non-handicapped testers were also able to meet this goal. These tests also show that the quadriplegic testers were less than 5 seconds slower than the non-handicapped testers. This result implies that the system is nearly as easy to use for quadriplegics as it is for anyone else.

After testing that the system could be operated by quadriplegics, it was tested to for accuracy requirements. Two targets were placed 100 yards down range and five shots were fired into each one. The groupings were kept and recorded (See Figure 72).

Figure 72: Shot Groupings

The inner circles of these targets have 2 inch diameters. The accuracy goal for the Mobile Marksman IV is a 4 inch grouping at 200 yards. This corresponds to a 2 inch grouping at 100 yards. As seen in the targets, this goal was met because all shots were within the inner circles.

Conclusion

There are three main goals for the Mobile Marksman: accommodation of quadriplegics, total cost of less than $1,600, and accuracy. The head tracking and sip/puff input devices provide a strictly head and neck control scheme. The time trail tests with actual quadriplegics

showed that this scheme was very capable of accommodating all their needs. A breakdown of the budget shows that the entire system can be manufactured under the target goal of $1,600. And the shot testing shows that the system achieved the accuracy requirements. With all these goals met, the Mobile Marksman IV proves to be a success.

Appendices

Appendix 1: MM_TX Code

```
#define OUTPUT_BAUD_RATE 115200

struct SEND_DATA_STRUCTURE{
  //put your variable definitions here for the data you want to send
  //THIS MUST BE EXACTLY THE SAME ON THE OTHER ARDUINO
  int yaw_tx;
  int pitch_tx;
  int yaw_rawtx;
  int cal_mode;
};

SEND_DATA_STRUCTURE mydata;
// Sensor data output interval in milliseconds
// This may not work, if faster than 20ms (=50Hz)
// Code is tuned for 20ms, so better leave it like that
#define OUTPUT_DATA_INTERVAL 20  // in milliseconds

#include <EasyTransfer.h>
#include <Wire.h>
//create object
EasyTransfer ET;

// SENSOR CALIBRATION
/*****************************************************************/
// How to calibrate? Read the tutorial at http://dev.qu.tu-berlin.de/projects/sf-razor-9dof-ahrs
// Put MIN/MAX and OFFSET readings for your board here!
// Accelerometer
// "accel x,y,z (min/max) = X_MIN/X_MAX  Y_MIN/Y_MAX  Z_MIN/Z_MAX"
#define ACCEL_X_MIN (-250.0f)
#define ACCEL_X_MAX (250.0f)
#define ACCEL_Y_MIN (-250.0f)
#define ACCEL_Y_MAX (250.0f)
#define ACCEL_Z_MIN (-250.0f)
#define ACCEL_Z_MAX (250.0f)

// Magnetometer
// "magn x,y,z (min/max) = X_MIN/X_MAX  Y_MIN/Y_MAX  Z_MIN/Z_MAX"
#define MAGN_X_MIN (-600.0f)
#define MAGN_X_MAX (600.0f)
#define MAGN_Y_MIN (-600.0f)
#define MAGN_Y_MAX (600.0f)
#define MAGN_Z_MIN (-600.0f)
```

```
#define MAGN_Z_MAX (600.0f)

// Gyroscope
// "gyro x,y,z (current/average) = .../OFFSET_X  .../OFFSET_Y  .../OFFSET_Z
#define GYRO_X_OFFSET (0.0f)
#define GYRO_Y_OFFSET (0.0f)
#define GYRO_Z_OFFSET (0.0f)


// Altymeter
#define ALT_SEA_LEVEL_PRESSURE 102133


// Calibration example:
// "accel x,y,z (min/max) = -278.00/270.00  -254.00/284.00  -294.00/235.00"
#define ACCEL_X_MIN ((float) -278)
#define ACCEL_X_MAX ((float) 270)
#define ACCEL_Y_MIN ((float) -254)
#define ACCEL_Y_MAX ((float) 284)
#define ACCEL_Z_MIN ((float) -294)
#define ACCEL_Z_MAX ((float) 235)

// "magn x,y,z (min/max) = -511.00/581.00  -516.00/568.00  -489.00/486.00"
#define MAGN_X_MIN ((float) -511)
#define MAGN_X_MAX ((float) 581)
#define MAGN_Y_MIN ((float) -516)
#define MAGN_Y_MAX ((float) 568)
#define MAGN_Z_MIN ((float) -489)
#define MAGN_Z_MAX ((float) 486)

//"gyro x,y,z (current/average) = -32.00/-34.82  102.00/100.41  -16.00/-16.38"
#define GYRO_AVERAGE_OFFSET_X ((float) -34.82)
#define GYRO_AVERAGE_OFFSET_Y ((float) 100.41)
#define GYRO_AVERAGE_OFFSET_Z ((float) -16.38)



// Sensor calibration scale and offset values
#define ACCEL_X_OFFSET ((ACCEL_X_MIN + ACCEL_X_MAX) / 2.0f)
#define ACCEL_Y_OFFSET ((ACCEL_Y_MIN + ACCEL_Y_MAX) / 2.0f)
#define ACCEL_Z_OFFSET ((ACCEL_Z_MIN + ACCEL_Z_MAX) / 2.0f)
#define ACCEL_X_SCALE (GRAVITY / (ACCEL_X_MAX - ACCEL_X_OFFSET))
#define ACCEL_Y_SCALE (GRAVITY / (ACCEL_Y_MAX - ACCEL_Y_OFFSET))
#define ACCEL_Z_SCALE (GRAVITY / (ACCEL_Z_MAX - ACCEL_Z_OFFSET))

#define MAGN_X_OFFSET ((MAGN_X_MIN + MAGN_X_MAX) / 2.0f)
```

```
#define MAGN_Y_OFFSET ((MAGN_Y_MIN + MAGN_Y_MAX) / 2.0f)
#define MAGN_Z_OFFSET ((MAGN_Z_MIN + MAGN_Z_MAX) / 2.0f)
#define MAGN_X_SCALE (100.0f / (MAGN_X_MAX - MAGN_X_OFFSET))
#define MAGN_Y_SCALE (100.0f / (MAGN_Y_MAX - MAGN_Y_OFFSET))
#define MAGN_Z_SCALE (100.0f / (MAGN_Z_MAX - MAGN_Z_OFFSET))

// Gain for gyroscope
#define GYRO_GAIN_X (0.06957f)
#define GYRO_GAIN_Y (0.06957f)
#define GYRO_GAIN_Z (0.06957f)

#define GYRO_X_SCALE (TO_RAD(GYRO_GAIN_X))
#define GYRO_Y_SCALE (TO_RAD(GYRO_GAIN_Y))
#define GYRO_Z_SCALE (TO_RAD(GYRO_GAIN_Z))

// DCM parameters
#define Kp_ROLLPITCH (0.02f)
#define Ki_ROLLPITCH (0.00002f)
#define Kp_YAW (1.4f)//orig = 1.2f
#define Ki_YAW (0.0002f)//orig = (0.00002f)

// Stuff
#define GRAVITY (256.0f) // "1G reference" used for DCM filter and accelerometer calibration
#define TO_RAD(x) (x * 0.01745329252)  // *pi/180
#define TO_DEG(x) (x * 57.2957795131)  // *180/pi

// RAW sensor data
float accel[3];  // Actually stores the NEGATED acceleration (equals gravity, if board not
moving).
//float accel_min[3];
//float accel_max[3];

float magnetom[3];
//float magnetom_min[3];
//float magnetom_max[3];

float gyro[3];
//float gyro_average[3];
//int gyro_num_samples = 0;

float temperature;
float pressure;
float altitude;

// DCM variables
float MAG_Heading;
```

```
float Magn_Vector[3]= {
  0, 0, 0}; // Store the magnetometer turn rate in a vector
float Accel_Vector[3]= {
  0, 0, 0}; // Store the acceleration in a vector
float Gyro_Vector[3]= {
  0, 0, 0}; // Store the gyros turn rate in a vector
float Omega_Vector[3]= {
  0, 0, 0}; // Corrected Gyro_Vector data
float Omega_P[3]= {
  0, 0, 0}; // Omega Proportional correction
float Omega_I[3]= {
  0, 0, 0}; // Omega Integrator
float Omega[3]= {
  0, 0, 0};
float errorRollPitch[3] = {
  0, 0, 0};
float errorYaw[3] = {
  0, 0, 0};
float DCM_Matrix[3][3] = {
  {
    1, 0, 0  }
  , {
    0, 1, 0  }
  , {
    0, 0, 1  }
};
float Update_Matrix[3][3] = {
  {
    0, 1, 2  }
  , {
    3, 4, 5  }
  , {
    6, 7, 8  }
};
float Temporary_Matrix[3][3] = {
  {
    0, 0, 0  }
  , {
    0, 0, 0  }
  , {
    0, 0, 0  }
};

// Euler angles
float yaw, pitch, roll;
```

```
// DCM timing in the main loop
long timestamp;
long timestamp_old;
float G_Dt; // Integration time for DCM algorithm

// More output-state variables
int num_accel_errors = 0;
int num_magn_errors = 0;
int num_gyro_errors = 0;

void ReadSensors() {
  Read_Pressure();
  Read_Gyro(); // Read gyroscope
  Read_Accel(); // Read accelerometer
  Read_Magn(); // Read magnetometer
  ApplySensorMapping();
}

// Read every sensor and record a time stamp
// Init DCM with unfiltered orientation
// TODO re-init global vars?
void reset_sensor_fusion()
{
  float temp1[3];
  float temp2[3];
  float xAxis[] = {
    1.0f, 0.0f, 0.0f  };

  ReadSensors();

  timestamp = millis();

  // GET PITCH
  // Using y-z-plane-component/x-component of gravity vector
  pitch = -atan2(Accel_Vector[0], sqrt(Accel_Vector[1] * Accel_Vector[1] + Accel_Vector[2] *
Accel_Vector[2]));

  // GET ROLL
  // Compensate pitch of gravity vector
  Vector_Cross_Product(temp1, Accel_Vector, xAxis);
  Vector_Cross_Product(temp2, xAxis, temp1);
  // Normally using x-z-plane-component/y-component of compensated gravity vector
  // roll = atan2(temp2[1], sqrt(temp2[0] * temp2[0] + temp2[2] * temp2[2]));
  // Since we compensated for pitch, x-z-plane-component equals z-component:
  roll = atan2(temp2[1], temp2[2]);
```

```
  // GET YAW
  Compass_Heading();
  yaw = MAG_Heading;

  // Init rotation matrix
  init_rotation_matrix(DCM_Matrix, yaw, pitch, roll);
}

// Apply calibration to raw sensor readings
void ApplySensorMapping()
{
  // Magnetometer axis mapping
  Magn_Vector[1] = -magnetom[0];
  Magn_Vector[0] = -magnetom[1];
  Magn_Vector[2] = -magnetom[2];

  // Magnetometer values mapping
  Magn_Vector[0] -= MAGN_X_OFFSET;
  Magn_Vector[0] *= MAGN_X_SCALE;
  Magn_Vector[1] -= MAGN_Y_OFFSET;
  Magn_Vector[1] *= MAGN_Y_SCALE;
  Magn_Vector[2] -= MAGN_Z_OFFSET;
  Magn_Vector[2] *= MAGN_Z_SCALE;

  // Accelerometer axis mapping
  Accel_Vector[1] = accel[0];
  Accel_Vector[0] = accel[1];
  Accel_Vector[2] = accel[2];

  // Accelerometer values mapping
  Accel_Vector[0] -= ACCEL_X_OFFSET;
  Accel_Vector[0] *= ACCEL_X_SCALE;
  Accel_Vector[1] -= ACCEL_Y_OFFSET;
  Accel_Vector[1] *= ACCEL_Y_SCALE;
  Accel_Vector[2] -= ACCEL_Z_OFFSET;
  Accel_Vector[2] *= ACCEL_Z_SCALE;

  // Gyroscope axis mapping
  Gyro_Vector[1] = -gyro[0];
  Gyro_Vector[0] = -gyro[1];
  Gyro_Vector[2] = -gyro[2];

  // Gyroscope values mapping
  Gyro_Vector[0] -= GYRO_X_OFFSET;
  Gyro_Vector[0] *= GYRO_X_SCALE;
  Gyro_Vector[1] -= GYRO_Y_OFFSET;
```

```
  Gyro_Vector[1] *= GYRO_Y_SCALE;
  Gyro_Vector[2] -= GYRO_Z_OFFSET;
  Gyro_Vector[2] *= GYRO_Z_SCALE;
}



void setup()
{
  // Init serial output
  //Serial.begin(57600);
  //Serial1.begin(9600);
  Serial1.begin(115200);
  //Serial1.begin(115200);


  // Init sensors
  delay(50);  // Give sensors enough time to start
  I2C_Init();
  Accel_Init();
  Magn_Init();
  Gyro_Init();
  Pressure_Init();

  // Read sensors, init DCM algorithm
  delay(20);  // Give sensors enough time to collect data
  reset_sensor_fusion();
  ET.begin(details(mydata), &Serial1);
  //calibration();
}


float yaw_new;
float yaw_offset = 0;
float pit_offset = 0;
float yaw_map;
// Main loop
void loop()
{

  //delay(50);
  // Time to read the sensors again?
  if ((millis() - timestamp) >= OUTPUT_DATA_INTERVAL) {
    timestamp_old = timestamp;
    timestamp = millis();
```

```
    if (timestamp > timestamp_old)
      G_Dt = (float) (timestamp - timestamp_old) / 1000.0f; // Real time of loop run. We use this
on the DCM algorithm (gyro integration time)
    else
      G_Dt = 0;

    ReadSensors();

    // Run DCM algorithm
    Compass_Heading(); // Calculate magnetic heading

      Matrix_update();

    Normalize();
    Drift_correction();
    Euler_angles();


  }

   yaw_map = map(TO_DEG(yaw),-180,180,0,360);
   mydata.yaw_rawtx = TO_DEG(yaw);
   yaw_new = yaw_map-yaw_offset;

   //Serial.print(" YAW_NEW = ");
   Serial.print(yaw_new);
   if(yaw_new > 180){
     yaw_new = yaw_new - 360;
     //Serial.print("SUB  ");
   }
   else if(yaw_new < -180){
     yaw_new = yaw_new + 360;
     //Serial.print("ADD  ");
   }
   else{   }



  mydata.yaw_tx = yaw_new;
  mydata.pitch_tx = TO_DEG(pitch) + pit_offset;




  Serial.print("YAW: "); Serial.print(mydata.yaw_tx); Serial.print("   PITCH: ");
Serial.println(mydata.pitch_tx);
```

```
  ET.sendData();
  delay(50);

   if ( digitalRead(4) == HIGH){
     mydata.cal_mode = 1;
     calibration();
   }
   else{
     mydata.cal_mode = 0;
   }

}

void calibration(){
  Serial.print("CALIBRATION MODE :   ");
  yaw_map = map(TO_DEG(yaw), -180,180,0,360);
  yaw_offset = yaw_map;
  pit_offset = 0-(TO_DEG(pitch));
      Serial.print(yaw_offset);Serial.print(", ");Serial.println(pit_offset);

}
```

Appendix 2: MM_RX Code

Main Code:

```
#include <SPI.h>
#include <MAX7456.h>
#include <EasyTransfer.h>
#include <EEPROM.h>
#include <Servo.h>;
// initialize the library with the numbers of the interface pins


// Pin Mapping ////////////////////////////////////////////////////////////////

  // pinValue = 0 means "not connected"

  //  Max7456 +5V   [DVDD,AVDD,PVDD]  ---  Arduino  VCC      (AVCC,VCC)
  //  Max7456 GND   [DGND,AGND,PGND]  ---  Arduino  GND      (AGND,GND)
  //  Max7456 CS    [~CS]          <---  Arduino  10  [PB2](SS/OC1B)
  //  Max7456 CS    [~CS]          <---  Mega2560 43  [PL6]
  const byte osdChipSelect            =          43;
  //  Max7456 DIN   [SDIN]         <---  Arduino  11  [PB3](MOSI/OC2)
  //  Max7456 DIN   [SDIN]         <---  Mega2560 51  [PB2](MOSI)
  const byte masterOutSlaveIn         =                  MOSI;
  //  Max7456 DOUT  [SDOUT]        --->  Arduino  12  [PB4](MISO)
  //  Max7456 DOUT  [SDOUT]        --->  Mega2560 50  [PB3](MISO)
  const byte masterInSlaveOut         =                  MISO;
  //  Max7456 SCK   [SCLK]         <---  Arduino  13  [PB5](SCK)
  //  Max7456 SCK   [SCLK]         <---  Mega2560 52  [PB1](SCK)
  const byte slaveClock               =                  SCK;
  //  Max7456 RST   [~RESET]        ---  Arduino  RST      (RESET)
  const byte osdReset                 =          0;
  //  Max7456 VSYNC [~VSYNC]       -X-
  //  Max7456 HSYNC [~HSYNC]       -X-
  //  Max7456 LOS   [LOS]          -X-
////////////////////////////////////////////////////////////////////////////////



//create object
EasyTransfer ET;
MAX7456 OSD( osdChipSelect );
Servo trigger;


int x_delay;
```

```
int y_delay;

float yaw_new;
float yaw_offset = 0;
float pit_offset = 0;
float yaw_map;

//const int buttonPinLeft = 48;      // pin for the Up button
//const int buttonPinRight = 40;    // pin for the Down button
const int sip_pin = 27;//buttonPinUp = 46;     // pin for the Esc button
const int saftey_pin = 31; //buttonPinEnter = 44;   // pin for the Enter button
const int puff_pin = 29; //buttonPinDown = 42;

const int joy_U = 11;
const int joy_D = 10;
const int joy_L = 12;
const int joy_R = 13;

const int safe_led = 5;
const int hi_led = 8;
const int lo_led = 9;
const int lock_led = 6;
const int fire_led = 3;
const int blue_led = 36;

int pinI1=30;//define I1 interface
int pinI2=28;//define I2 interface
int speedpinA=2;//enable motor A
int pinI3=26;//define I3 interface
int pinI4=24;//define I4 interface
int speedpinB=4;//enable motor B

int spead = 250;//define the spead of motor
int speadb;
int delaya;
int delayb;

int lo_mode = 0;

int pit_new;
int pitch_raw;
int pit_gray;
int yaw_gray;
int pot_yaw = A7;
int pot_pit = A8;
```

```
int act_posx = A0;
int act_posy = A2;


int acc_sw = 40;
int x_curr;
int y_curr;

int x_speedhi = 220;
int y_speedhi = 220;
int x_speedlo = 105;
int y_speedlo = 68;

int sip_delay = 1500;

int fire_enable = 0;
int target_enable = 0;

int x_speed;
int y_speed;

int main_mode = 1;

struct RECEIVE_DATA_STRUCTURE{
  //put your variable definitions here for the data you want to receive
  //THIS MUST BE EXACTLY THE SAME ON THE OTHER ARDUINO
  int yaw_tx;
  int pitch_tx;
  int ytx_raw;
  int ptx_raw;
  int cal_flag;
};

//give a name to the group of data
RECEIVE_DATA_STRUCTURE mydata;

void setup(){

  trigger.attach(7);
  trigger.write(10);

  SPI.begin();
  SPI.setClockDivider( SPI_CLOCK_DIV2 );      // Must be less than 10MHz.

    // Initialize the MAX7456 OSD:
  OSD.begin();                          // Use NTSC with default area.
```

```
   //OSD.setCharEncoding( MAX7456_ASCII );       // Only needed if ascii font.
  OSD.display();

  Serial2.begin(115200);
  Serial.begin(57600);
  //start the library, pass in the data details and the name of the serial port. Can be Serial, Serial1,
Serial2, etc.
  ET.begin(details(mydata), &Serial2);

  pinMode(blue_led,OUTPUT);
  pinMode(safe_led,OUTPUT);
  pinMode(hi_led,OUTPUT);
  pinMode(lo_led,OUTPUT);
  pinMode(lock_led,OUTPUT);
  pinMode(fire_led,OUTPUT);

  pinMode(pinI1,OUTPUT);
  pinMode(pinI2,OUTPUT);
  pinMode(speedpinA,OUTPUT);
  pinMode(pinI3,OUTPUT);
  pinMode(pinI4,OUTPUT);
  pinMode(speedpinB,OUTPUT);

  pinMode(acc_sw, INPUT_PULLUP);


  pinMode(joy_U, INPUT_PULLUP);
  pinMode(joy_D, INPUT_PULLUP);
  pinMode(joy_L, INPUT_PULLUP);
  pinMode(joy_R, INPUT_PULLUP);

  pinMode(34,INPUT);

  pinMode(saftey_pin, INPUT_PULLUP);
  pinMode(sip_pin, INPUT_PULLUP);
  pinMode(puff_pin,INPUT_PULLUP);

}
void get_readings()
{
 if(ET.receiveData()){

   yaw_map = map(mydata.ytx_raw,-180,180,0,360);
   yaw_new = yaw_map-yaw_offset;
   pit_new = mydata.ptx_raw - pit_offset;
```

```
  if(yaw_new > 180){
    yaw_new = yaw_new - 360;
    //Serial.print("SUB  ");
  }
  else if(yaw_new < -180){
    yaw_new = yaw_new + 360;
    //Serial.print("ADD  ");
  }
  //else if(yaw_new =< 180 || yaw_new => -180){
  //}
  else{}
 }

 else{
   //yaw_new = 0;
   //pit_new = 0;
 }

 if(digitalRead(34) == LOW){
   yaw_new = 0;
   pit_new = 0;
   //Serial.println("BT Not Connected");
   digitalWrite(blue_led, LOW);
 }
 else if(digitalRead(34 == HIGH)){
   digitalWrite(blue_led, HIGH);
   //Serial.println("BT Connected");
 }

 //  Serial.println(pit_new);
 //  Serial.print(" YAW_NEW = ");
 //  Serial.println(yaw_new);
}


int cs_yaw;
int delay_flag = 0;

void loop(){
  cs_yaw = 0;
  for(int i = 0; i <150; ++i){
    cs_yaw = cs_yaw + analogRead(A15);
  }
  cs_yaw = cs_yaw/150;
  cs_yaw = map(cs_yaw, 0,200,0,1000);
   Serial.println(cs_yaw);//Serial.print("     "); Serial.println(analogRead(A13));
```

```
switch(main_mode){

  //SAFE MODE

  case 1:{

    lo_mode = 0;
    //Serial.println("SAFE");
    fire_enable = 0;
    target_enable = 0;
       OSD.setCursor(0,0);
       OSD.println("       SAFETY MODE:   ");
       OSD.setCursor(0,4);
       OSD.println("    PRESS SAFTEY BUTTON ");
       OSD.setCursor(0,5);
       OSD.print("  TO ENTER TARGETING MODE");

      //  OSD.setCursor(0,8);
       // OSD.print("    ");
      //  OSD.setCursor(0,8);
      //  OSD.print(pit_new);

        MODE_LED(1);


        if(delay_flag == 1){
          delay(2000);
          delay_flag = 0;
        }
        else{}

    if(digitalRead(sip_pin) == LOW || digitalRead(puff_pin) == LOW){
      OSD.setCursor(0,7);
      OSD.print("CALIBRATING..");
      Calibrate();
      OSD.setCursor(0,7);
      OSD.print("CALIBRATED..  ");
      delay(1000);
      OSD.setCursor(0,7);
      OSD.print("              ");
      delay(sip_delay);
    }

     else if (digitalRead(saftey_pin) == LOW){
```

```
    main_mode = 2;
    OSD.clear();
    delay_flag = 1;
   }

break;
}


//high speed
case 2:{
  lo_mode = 0;
    MODE_LED(2);
    //Serial.println("HIGH SPEED");

  // while (OSD.notInVSync());                  // Wait for VSync to start to
                                        //   prevent write artifacts.
      OSD.setCursor(0,0);
      OSD.println("      TARGETING MODE:   ");
      OSD.setCursor(0,1);
      OSD.println("       (HIGH SPEED)     ");
      OSD.setCursor(0,10);
      OSD.print("         PUFF FOR  ");
      OSD.setCursor(0,11);
      OSD.print("        LOW  SPEED  ");


    x_speed = x_speedlo;
    y_speed = y_speedlo;
    x_delay = 0;
    y_delay = 0;
    target_enable = 1;
    fire_enable = 0;

    if(digitalRead(puff_pin) == LOW){
      fire_enable = 0;
      target_enable = 1;

      main_mode = 3;
      OSD.clear();
      delay(sip_delay);
    }

    else if(digitalRead(saftey_pin) == LOW){
      fire_enable = 0;
      target_enable = 0;
```

```
      main_mode = 1;
      OSD.clear();
      delay_flag = 1;
    }
  break;
}

//low speed
case 3:{
    MODE_LED(3);
   // Serial.println("Low Speed");
    lo_mode = 1;
      OSD.setCursor(0,0);
      OSD.println("      TARGETING MODE:   ");
      OSD.setCursor(0,1);
      OSD.println("        (LOW SPEED)     ");
      OSD.setCursor(0,10);
      OSD.print("         PUFF FOR  ");
      OSD.setCursor(0,11);
      OSD.print("       TARGET  LOCK  ");


    fire_enable = 0;
    target_enable = 1;
    x_speed = x_speedlo;
    y_speed = y_speedlo;
    x_delay = 50;
    y_delay = 50;

    if(digitalRead(puff_pin) == LOW){
      main_mode = 4;
      fire_enable = 0;
      target_enable = 0;
      OSD.clear();
      delay(sip_delay);
  }

    else if(digitalRead(saftey_pin) == LOW){
      fire_enable = 0;
      target_enable = 0;
      OSD.clear();
      main_mode = 1;
      delay_flag = 1;
    }

  break;
```

```
}


//Target Lock
case 4:{
    // Serial.println("Target Lock");
     MODE_LED(4);
    lo_mode = 0;
    OSD.setCursor(0,0);
     OSD.println("     TARGET LOCK MODE:   ");
     OSD.setCursor(0,10);
     OSD.print("PUFF:                SIP:");
     OSD.setCursor(0,11);
     OSD.print("HIGH SPD       FIRING MODE  ");

     fire_enable = 0;
     target_enable = 0;

   if(digitalRead(puff_pin) == LOW){
     main_mode = 2;
     fire_enable = 0;
     target_enable = 1;
     OSD.clear();
     delay(sip_delay);
   }
   else if(digitalRead(sip_pin) == LOW){
     main_mode = 5;
     target_enable = 0;
     OSD.clear();
     delay(sip_delay);
   }

   else if(digitalRead(saftey_pin) == LOW){
     fire_enable = 0;
     target_enable = 0;
     OSD.clear();
     main_mode = 1;
       delay_flag = 1;
   }
  break;
}

//Fire!!!!!!!!!!
case 5:{
  MODE_LED(5);
 // Serial.println("Fire");
```

```
   lo_mode = 0;
   OSD.setCursor(0,0);
      OSD.println("       FIRING MODE:   ");
      OSD.setCursor(0,10);
      OSD.print("        PUFF TO  ");
      OSD.setCursor(0,11);
      OSD.print("          FIRE!  ");


   fire_enable = 1;
   target_enable = 0;

   if(digitalRead(puff_pin) == LOW){
     OSD.setCursor(7,6);
     OSD.print("   FIRING..");
     Fire();
     fire_enable = 0;
     target_enable = 0;
     main_mode = 1;
     OSD.clear();
     delay(sip_delay);
   }

   else if(digitalRead(sip_pin) == LOW){
    target_enable = 1;
    fire_enable = 0;
    main_mode = 2;
    OSD.clear();
    delay(sip_delay);
   }

   else if(digitalRead(saftey_pin) == LOW){
      fire_enable = 0;
      target_enable = 0;
      OSD.clear();
      main_mode = 1;
      delay_flag = 1;
     }
   break;
  }

}

  get_readings();

  if( digitalRead(joy_R) == LOW &&  digitalRead(joy_U) == LOW){
```

```
    out();
    up();
    delay(10);
    }
  else if( digitalRead(joy_R) == LOW &&  digitalRead(joy_L) == LOW){
    out();
    down();
    delay(10);
    }
  else if( digitalRead(joy_D) == LOW &&  digitalRead(joy_U) == LOW){
    in();
    up();
    delay(10);
    }
  else if( digitalRead(joy_D) == LOW &&  digitalRead(joy_L) == LOW){
    in();
    down();
    delay(10);
    }


//  while(yaw_new < -23){
//  out();
//  get_readings();
//  }
//  while(yaw_new > 23){
//    in();
//    get_readings();
//    read_buttons()
//  }
//  while(pit_new > 20){
//    down();
//    get_readings();
//  }
//  while(pit_new < -20){
//    up();
//    get_readings();
//  }
//
//    stop();
//

  int yaw_avg = 0;
  int pit_avg = 0;
  for(int p = 0; p<20; ++p){
      yaw_avg = yaw_avg + map(analogRead(pot_yaw),0,1022,30,10);
```

```
    pit_avg = pit_avg + map(analogRead(pot_pit),0,1022,30,10);
  }
  yaw_gray = yaw_avg/20;
  pit_gray = pit_avg/20;

  //Serial.print("yaw gray: "); Serial.print(yaw_gray);Serial.print("  pit gray: ");
Serial.println(pit_gray);
   //Serial.println(analogRead(7));
   //Serial.println(digitalRead(acc_sw));

  if(digitalRead(joy_R) == LOW || yaw_new < -(yaw_gray)){
    out();
    if(lo_mode == 1){
      delay(20);
      stop();
      delay(x_delay);
    }
    else{
      delay(10);
    }//stop();
  }
  else if(digitalRead(joy_L) == LOW || yaw_new > yaw_gray){
    in();
   if(lo_mode == 1){
      delay(20);
      stop();
      delay(x_delay);
    }
    else{
    delay(10);
    }
    //stop();
  }
  else if(digitalRead(joy_U) == LOW || pit_new < -(pit_gray)){
    up();
    if(lo_mode == 1){
      delay(20);
      stop();
      delay(y_delay);
    }
    else{
    //delay(10);
    }
    //delay(y_delay);
    //stop();
  }
```

```
  else if(digitalRead(joy_D) == LOW || pit_new > pit_gray){
    down();
    if(lo_mode == 1){
      delay(20);
      stop();
      delay(y_delay);
    }
    else{
    //delay(10);
    }
    //delay(y_delay);
    //stop();
  }
  else{
    stop();
  }
}


void Fire(){

  if (fire_enable == 1) {
    Serial.print("FIRE!");
    trigger.write(100);
    delay(2000);
    trigger.write(10);
    delay(2000);
  }

  fire_enable = 0;

}

void Calibrate(){
  Serial.print("Calibrate!");
    yaw_map = map(mydata.ytx_raw, -180,180,0,360);
    yaw_offset = yaw_map;
    pit_offset = (mydata.ptx_raw);
  delay(1000);
}

void MODE_LED(int mode_light){
  switch (mode_light){
    case 1:
      digitalWrite(safe_led,HIGH);
      digitalWrite(hi_led,LOW);
```

```
      digitalWrite(lo_led,LOW);
      digitalWrite(lock_led,LOW);
      digitalWrite(fire_led,LOW);
      break;
    case 2:
      digitalWrite(safe_led,LOW);
      digitalWrite(hi_led,HIGH);
      digitalWrite(lo_led,LOW);
      digitalWrite(lock_led,LOW);
      digitalWrite(fire_led,LOW);
      break;
    case 3:
      digitalWrite(safe_led,LOW);
      digitalWrite(hi_led,LOW);
      digitalWrite(lo_led,HIGH);
      digitalWrite(lock_led,LOW);
      digitalWrite(fire_led,LOW);
      break;
    case 4 :
      digitalWrite(safe_led,LOW);
      digitalWrite(hi_led,LOW);
      digitalWrite(lo_led,LOW);
      digitalWrite(lock_led,HIGH);
      digitalWrite(fire_led,LOW);
      break;
    case 5:
      digitalWrite(safe_led,LOW);
      digitalWrite(hi_led,LOW);
      digitalWrite(lo_led,LOW);
      digitalWrite(lock_led,LOW);
      digitalWrite(fire_led,HIGH);
      break;
  }
}
```

Actuator Functions:

```
void out()    //horizontal actuator out
{
  if (target_enable == 1){
  analogWrite(speedpinA,x_speed);//input a simulation value to set the speed
  analogWrite(speedpinB,y_speed);
  digitalWrite(pinI2,LOW);//turn DC Motor A move anticlockwise
  digitalWrite(pinI1,HIGH);
```

```
  digitalWrite(pinI4,LOW);//turn DC Motor B move clockwise
  digitalWrite(pinI3,LOW);
  //Serial.println("Right");
  //delay(x_delay);
  }
  else if(target_enable == 0){}
}
void in()// horiz act in
{
  if (target_enable == 1){
  analogWrite(speedpinA,x_speed);//input a simulation value to set the speed
  analogWrite(speedpinB,y_speed);
  digitalWrite(pinI2,HIGH);//turn DC Motor A move clockwise
  digitalWrite(pinI1,LOW);
  digitalWrite(pinI4,LOW);//turn DC Motor B move clockwise
  digitalWrite(pinI3,LOW);
  //delay(x_delay);
  //Serial.println("Left");
}
   else if(target_enable == 0){}
}
void up()//
{
  if (target_enable == 1){
  analogWrite(speedpinA,x_speed);//input a simulation value to set the speed
  analogWrite(speedpinB,y_speed);
  digitalWrite(pinI4,HIGH);//turn DC Motor B move clockwise
  digitalWrite(pinI3,LOW);
  digitalWrite(pinI2,LOW);//turn DC Motor A move clockwise
  digitalWrite(pinI1,LOW);
  //Serial.println("UP");
  }
   else if(target_enable == 0){}
}
void down()//
{
  if (target_enable == 1){
  analogWrite(speedpinA,x_speed);//input a simulation value to set the speed
  analogWrite(speedpinB,y_speed);
  digitalWrite(pinI4,LOW);//turn DC Motor B move anticlockwise
  digitalWrite(pinI3,HIGH);
  digitalWrite(pinI2,LOW);//turn DC Motor A move clockwise
  digitalWrite(pinI1,LOW);
  //Serial.println("Down");
  }
   else if(target_enable == 0){}
```

```
}
void stop()//
{
  digitalWrite(speedpinA,LOW);// Unenble the pin, to stop the motor. this should be done to avid
damaging the motor.
  digitalWrite(speedpinB,LOW);
    digitalWrite(pinI4,LOW);
  digitalWrite(pinI3,LOW);
  digitalWrite(pinI2,LOW);
  digitalWrite(pinI1,LOW);
  //delay(10);

}
```

Appendix 3: Device Connector Chart

Device	Connector
Sip/Puff	Stereo Jack
Safety Switch	1/4” Audio Jack
Power Source	12V DC Receptacle (Cigarette Lighter)
Camera	RCA Jack
LCD Screen	RCA Jack
Trigger Servo	USB
Actuators	XLR
Joystick	Ethernet

Sources

IMU:
http://electroiq.com/blog/2010/11/introduction-to-mems-gyroscopes/
http://www.dimensionengineering.com/info/accelerometers
http://www.starlino.com/dcm_tutorial.html
Sip-puff:
http://enablingdevices.com/catalog/capability_switches/sip-puff-breath-switches/sip-and-puff-switches-accessories
http://www.beadaptive.com/brian/index.html
Actuators
http://www.servocity.com/html/115_lbs__thrust__linear_actuat.html
Servo:
http://www.servocity.com/html/hs-755mg_1_4_scale.html
Arduino:
Arduino.cc
http://arduino.cc/en/Main/arduinoBoardMega
Trigger lock:
http://www.masterlock.com/products/product_details/90DSPT
Displays:
http://www.antonline.com/p_Pyle--PLMRM71W--Pyle-PLMRM71W-Hydra-Series-Marine-Grade-Water-Resistant-7-Inch-LCD-Wide-Screen-Monitor-with-Anti-Glare-Shield-and-Universal-Stand-%28White%29-_1078891.htm
Camera:
http://stuntcams.com/shop/sony-mini-helmet-camera-smallest-bullet-p-115.html
Mount:
http://hyskore.com/wp/2013/06/21/deadeye-managed-recoil-shooting-varmint-rest/