NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

A SITUATIONAL-AWARENESS SYSTEM FOR NETWORKED INFANTRY INCLUDING AN ACCELEROMETER-BASED SHOT-IDENTIFICATION ALGORITHM FOR DIRECT-FIRE WEAPONS

by

Kiel A. Reese

September 2016

Thesis Advisor: Zachary Staples
Co-Advisor Xiaoping Yun

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704–0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2016	**3. REPORT TYPE AND DATES COVERED** Master's thesis	
4. TITLE AND SUBTITLE A SITUATIONAL-AWARENESS SYSTEM FOR NETWORKED INFANTRY INCLUDING AN ACCELEROMETER-BASED SHOT-IDENTIFICATION ALGORITHM FOR DIRECT-FIRE WEAPONS			**5. FUNDING NUMBERS**
6. AUTHOR(S) Kiel A. Reese			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Combat effectiveness is increased by decreasing uncertainty through shared situational awareness (SA) at all levels of command. A system that provides immediate knowledge of subordinates' locations and azimuths of fire when engaging the enemy increases the small-unit leader's SA, facilitating his coordination and execution of a course of action.

In this thesis, such a system was prototyped using commercial-off-the-shelf components related to the system's functional areas of shot identification, orientation, localization, data processing, and mapping. The primary focus of this project was the development of a shot-identification algorithm utilizing data collected from inertial sensors attached to Armalite Rifle 15 (AR15) variant weapons.

In spite of under-sampling limitations, the shot-identification algorithm was successful in classifying shots taken with three different AR15 rifles by six different shooters using a variety of stances and multiple engagement techniques. The full system was tested successfully with two rifle nodes passing localization and firing azimuth data in response to a simulated shot. This information passed over the Naval Postgraduate School network to a separate command operations center node where it was mapped in Google Earth.

Situational awareness, networked infantrymen, direct-fire weapons, AR15-variant weapons, shot identification, accelerometer, digital sampling, under sampling, Euler angles, firing azimuth, YEI, GPS, Google Earth, mapping	**15. NUMBER OF PAGES** 135
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540–01-280-5500 Standard Form 298 (Rev. 2–89) Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

A SITUATIONAL-AWARENESS SYSTEM FOR NETWORKED INFANTRY INCLUDING AN ACCELEROMETER-BASED SHOT-IDENTIFICATION ALGORITHM FOR DIRECT-FIRE WEAPONS

Kiel A. Reese
Captain, United States Marine Corps
B.S., United States Naval Academy, 2010

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2016

Approved by: Zachary Staples
Thesis Advisor

Xiaoping Yun
Co-Advisor

R. Clark Robertson
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Combat effectiveness is increased by decreasing uncertainty through shared situational awareness (SA) at all levels of command. A system that provides immediate knowledge of subordinates' locations and azimuths of fire when engaging the enemy increases the small-unit leader's SA, facilitating his coordination and execution of a course of action.

In this thesis, such a system was prototyped using commercial-off-the-shelf components related to the system's functional areas of shot identification, orientation, localization, data processing, and mapping. The primary focus of this project was the development of a shot-identification algorithm utilizing data collected from inertial sensors attached to Armalite Rifle 15 (AR15) variant weapons.

In spite of under-sampling limitations, the shot-identification algorithm was successful in classifying shots taken with three different AR15 rifles by six different shooters using a variety of stances and multiple engagement techniques. The full system was tested successfully with two rifle nodes passing localization and firing azimuth data in response to a simulated shot. This information passed over the Naval Postgraduate School network to a separate command operations center node where it was mapped in Google Earth.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

3D	three-dimensional
ADC	analog-to-digital converter
ADDRAC	alert, direction, description, range, assignment, and control
AHRS	Attitude Heading Reference System
AR15	Armalite Rifle-15
BFT	Blue Force Tracker
COC	command operations center
COTS	commercial-off-the-shelf
FBCB2	Force XXI Battle Command Brigade and Below
GCDC	Gulf Coast Data Concepts
G-M	Grid-Magnetic
GPS	Global Positioning System
IMU	inertial measurement unit
KML	Keyhole Markup Language
LW	Land Warrior
MEMS	micro-electromechanical system
MP15	Smith and Wesson Military and Police 15
NATO	North Atlantic Treaty Organization
NMEA	National Marine Electronics Association
NPS	Naval Postgraduate School
NW	Nett Warrior
Pub/Sub	Publication and Subscribe
Q-Comp	Quaternion Complementary
Q-grad	Quaternion Gradient Descent
RAM	random access memory
SA	situational awareness
SOP	standard operating procedure

TIC	troops-in-contact
TSS-DL	3-Space Sensor Data-logger
USMC	United States Marine Corps
YEI	Yost Engineering Incorporated

ACKNOWLEDGMENTS

I am truly indebted to the many individuals who assisted me in completing this thesis. Captain Caleb Khan was a mentor who brought me onboard the Reticle Project, helping me see the potential of networked infantrymen. He led me to my advisor, Commander Zachary Staples, who was instrumental in coordinating this work and allowing me to combine my love of shooting with my education. Dr. Xiaoping Yun graciously provided me with his wealth of technical knowledge. Dr. James Calusdian's support was pivotal in the lab; he patiently fielded my questions, and offered solutions and alternative ways forward. My colleagues, Major Adam Foushee and Mr. Cole Johnson, provided an important sounding board for ideas and significant comic relief. Due to California restrictions, this thesis would not have been possible without Mr. Travis Segura, Mr. Greg Bean, and the staff and shooting facilities of Soledad State Prison. Finally, my wife and daughter kept me on an even keel, and I am grateful they were understanding of this time-intensive project.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. UNITED STATES MARINE CORPS THEORY OF WAR

Marine Corps Doctrinal Publication 1 titled "Warfighting" provides the conceptual foundation of the United States Marine Corps' (USMC) understanding of warfare. This document describes war as a "violent struggle between two hostile, independent, and irreconcilable wills" that is characterized by friction, uncertainty, fluidity, disorder, and complexity [1]. The first two characteristics are linked in that friction, or the quality of war that makes the "simple difficult and the difficult seemingly impossible" [1], is caused in large part by the uncertainty of battlefield conditions. There are two options in overcoming friction to accomplish the mission. The first way is to lessen uncertainty, thereby increasing situational awareness (SA). In its most basic spatial form, SA breaks down into knowledge of self-location, friendly locations, and enemy locations. Even with increased knowledge, some friction remains, and units must fight through it, which leads to the second method of overcoming friction. This solution is the development of standard operating procedures (SOPs) defining immediate actions used to react quickly in spite of friction. The purpose of this thesis is to develop a means to increase SA and contribute to the automation of SOPs.

B. INFORMATION TECHNOLOGY TO INCREASE SA

Leveraging information technology as a means to increase SA and coordination on the battlefield has progressed since the 1990s. Northrup Grumman developed the Blue Force Tracker (BFT): Force XXI Battle Command Brigade and Below (FBCB2) system in 1995, and it was utilized heavily on the battlefields of Iraq and Afghanistan in recent years [2]. This system, primarily deployed on vehicles, employs both satellite and terrestrial communications as a means to pass friendly and enemy location data. Friendly location data is obtained via the Global Positioning System (GPS) receiver mounted on the vehicle roof and displayed on a digital map with other friendly units. Enemy units, once discovered, are displayed via manual entry into the system. The FBCB2 software allows for messaging across the BFT network. The positive effect of this common operating picture and

increased command and control capability is evident in a sharp decline in friendly direct ground fire incidents. In the major combat operations phase of Operation Iraqi Freedom, only one soldier was killed in such an incident, while 35 soldiers were killed and 72 wounded in similar incidents in Operation Desert Storm [2]. The second increment of the BFT system acquisition, under the title of Joint Battle Command-Platform, is currently being fielded [3]. This increment focuses on reducing latency in the system, increased encryption capabilities, and correcting identified shortfalls of the original BFT system.

The idea for implementing BFT-type capabilities for dismounted infantrymen spawned in the early 1990s but has been hamstrung by the state of technology, specifically with regard to form factor, power density, and computational speed [4]. Originally deemed the Land Warrior (LW) system, the current iteration of this concept is the Nett Warrior (NW) system. It is based on a chest-mounted, commercial-off-the-shelf (COTS) Samsung Galaxy Note smartphone, communicating via a Rifleman Radio (AN/PRC154A) and powered by a body conforming battery [5]. The smart phone provides data storage, display, and messaging capabilities similar to that of the BFT system. While NW is currently in Low Rate Initial Production [5], the earlier LW was used by several units in Afghanistan. These units lauded the increase in SA provided to dismounted infantry [4].

C. INTEGRATION OF SUPPORTING ARMS

Increasing the small-unit leader's SA, especially with regard to friendly and enemy locations, is also imperative to avoid fratricide when integrating with supporting arms. These supporting arms are typically aerial or indirect fires. Coordination between the small-unit leader and forward observer with aircraft, firing agency, or other fire support elements requires a precision that is difficult to obtain during enemy contact. Often, strategic SA assets such as unmanned aerial vehicles are unavailable, and the ground commander, command operations center (COC), pilot, and fire support agency have very different perspectives of the situation on the battlefield. This usually results in the unit leader on the ground providing aircraft with a "talk-on" to the target or by providing the fire support agency with a mensurated grid. Both of these actions take time to accomplish but are necessary to ensure every element has the same picture of battlefield geometries

before releasing ordnance. Testing of LW in Afghanistan demonstrated its usefulness to decision makers in the integration of supporting arms through increasing overall SA [4]. The implementations of BFT and LW demonstrate that a common perspective of the battlefield significantly reduces both fratricide and the amount of time to deliver ordnance downrange.

D. FIRE COMMAND SOP

Creating SOPs is a method of remaining effective in spite of friction. One of the most basic SOPs for the individual Marine infantryman is the issuance of a fire command. When under contact by enemy forces, the individual receiving incoming enemy fire calls out fire commands to inform squad members of a nearby target. The elements of a fire command are alert, direction, description, range, assignment, and control, referenced by the acronym ADDRAC [6]. Often with the friction inherent in a troops-in-contact (TIC) situation, this report may not be passed in a timely manner or, when passed, may contain incorrect elements. NW and BFT suffer from these issues, as their display of enemy locations is only possible via manual inputs to the system. With NW, this process takes five to ten seconds [7]. In a TIC, the individuals in contact focus on taking cover and returning fire, not on inputting accurate data into a device; thus, the small-unit leader and COC outside earshot or visual range may not even know about the situation to provide support. Addressing the automation of the Alert and Direction elements of ADDRAC to increase the SA of small-unit leaders rapidly and accurately is the aim of this thesis.

E. THESIS OBJECTIVE

At the small-unit level, the force that can react more quickly and in a more unified fashion has the advantage. This is accomplished by decreasing uncertainty through shared SA at the lowest levels, particularly with regard to friendly and enemy locations. The United States' military has pushed information technology to the vehicle level with BFT and is applying this concept even further by networking individual infantrymen with NW. Further applications will arise as the NW concept expands due to the reality of increasing computational speeds and higher power densities. One such application, which the NW program has yet to incorporate, is the data collection of the infantry weapons systems

themselves. This particular line of inquiry is being pursued at the Naval Postgraduate School (NPS) under the umbrella of the Reticle Project [8], [9]. Previous Reticle Project work investigated a potential application using the networked infantrymen's weapon orientation data to de-conflict geometries of fire in close-quarters combat [8].

In this Reticle Project thesis, the aim is to further the concept of individual weapons data collection as a means to increase SA while automating the first two elements of the ADDRAC SOP. Currently, the small-unit leader must have line-of-sight to both the friendly forces in contact and the enemy to obtain an accurate picture of the battlefield. These criteria are particularly difficult to meet in urban or wooded environments where visibility is limited. Additionally, distributed operations have become more frequent, putting fewer forces in larger areas of operation. This tends to spread forces out, further decreasing the likelihood that a leader can visually assess the whole battlefield. If the unit leader is unable to meet these criteria, he must act on his best estimate of the situation or wait for confirmation from his subordinates via radio.

A system that provides immediate knowledge of subordinates engaging the enemy along with those individuals' locations and directions of fire overlaid on a digital map would provide the small-unit leader with an accurate representation of battlefield geometries. This technological edge would allow the unit leader to plan and coordinate execution quickly and with confidence. Additionally, multiple shots at an enemy by different infantrymen, or a single moving infantryman, would make a position resection of the enemy's location immediately evident. This overlay of the intersections of multiple azimuths of fire would be invaluable to fire support agencies and assets attempting to locate the enemy for targeting. This information would also be crucial to the COC coordinating support during a TIC. In this thesis, the feasibility of such a system is investigated by first determining if a COTS Attitude Heading Reference System (AHRS) can accurately parse the shot acceleration profile of an Armalite Rifle-15 (AR15) style rifle. Additionally, a system that uses this shot information with a GPS receiver and map data from Google Earth to increase small-unit leader SA is prototyped.

Background information related to this particular application of networked infantrymen is presented in the next chapter. This includes the mechanics of the AR15

action, a discussion on digital sampling, a brief overview of micro-electromechanical system (MEMS) accelerometers, an explanation of the hardware selection for experimental data, and a review of related work. The progression of this project is detailed in Chapter III, which flows from hardware components and setup to data collection and algorithm development. The integration of the information with an orientation capability, a GPS receiver, and Google Earth is also presented. Finally, an attempt to translate the code for use on an embedded system is discussed. In Chapter IV, the results of implementing the algorithms and issues related to system integration are presented. In the final chapter, the summary of contributions for this project and recommendations for future work are presented.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

Discussion in this chapter lays the necessary conceptual foundations for this project. The topics presented include AR15 operation, Nyquist rate, MEMS accelerometers and AHRS, and related work.

A. AR15 OPERATION

The research described in this thesis relies heavily on understanding the mechanics of infantry direct-fire weapons. The current direct-fire weapon systems used by conventional United States infantrymen are the M16 and M4 carbine platforms, chambered for 5.56 mm North Atlantic Treaty Organization (NATO) ammunition. Both weapons are built on the AR15 action developed by Eugene Stoner in the late 1950s. The operation of this action, shown in Figure 1, is detailed in the USMC Rifle Marksmanship Reference Publication 3–10A [10] and is condensed in the following paragraph.

The semi-automatic operation of the AR15 action relies on some of the expanding gasses from the cartridge propellant being rerouted by the gas block through the gas port toward the rear of the rifle down the gas tube. This gas enters the gas key on the bolt carrier assembly, which allows the rotation of the locking lugs of the bolt, releasing the bolt carrier assembly from the barrel extension locking lugs behind the chamber. The bolt carrier assembly moves to the rear of the weapon, ejecting the spent casing and re-cocking the hammer. The buffer and buffer (or action) spring in the stock of the lower receiver compresses and returns the bolt carrier assembly forward where another cartridge is fed from a spring-loaded magazine. This new cartridge feeds into the chamber and locks in place by the rotation of the bolt-locking lugs. Pressing the trigger a second time releases the now reset, spring-loaded hammer, which strikes the firing pin. The firing pin impacts the primer of the cartridge, igniting the propellant, which creates expanding gasses forcing the projectile down the barrel restarting the process [10]. The major internal components of an AR15 are shown in Figure 1 with the path of the gas highlighted.

Analysis of this cycle reveals several distinct acceleration events for the external body of rifle, which consists of the upper and lower receivers rigidly connected. These

events include the 1) hammer fall, 2) firing pin strike, 3) initial recoil while the bolt-locking lugs are locked into the barrel extension locking lugs, 4) bolt carrier assembly stopping at the rear of the stock, 5) bolt carrier assembly returning to lock into the barrel extension locking lugs, and 6) trigger reset.

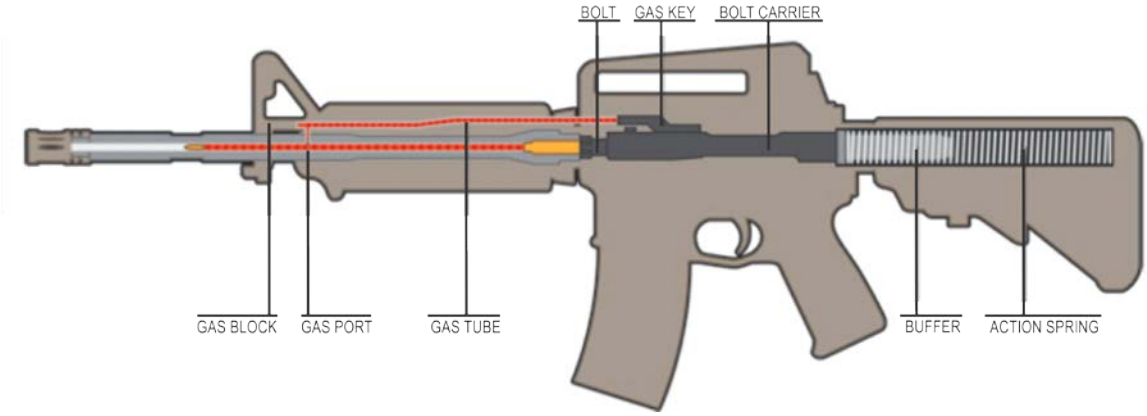

Figure 1. AR15 Operation Cycle. Adapted from [11].

Some of these events can be correlated to the signatures in Figure 2, which contains the force curve, in time, at the stock of an M4 during two shots. These two signatures vary in magnitude due to differing muzzle brakes [12]. The accelerations of the stock are directly proportional to the forces shown in Figure 2 via Newton's Second Law, $F = ma$. The first, black force signature is from an M4 with the standard military issue "birdcage" flash hider and is proportional to the desired acceleration signal that must be identified. Just before 0.1 s, there is a slight rise in rearward (positive) acceleration that is the frame of the rifle reacting according to Newton's Third Law to the hammer traveling forward after the trigger press releases it. The first large rearward peak is the initial recoil resulting from the expanding gasses forcing the projectile out of the barrel while the bolt-locking lugs are locked into the barrel extension lugs. At this instant, all the rifle components are rigidly connected, with the exception of the hammer striking the firing pin. The second positive peak, occurring at approximately 0.13 s, is the bolt carrier assembly pushing the body of the rifle backward once the buffer spring is completely compressed. The negative

acceleration prior to 0.2 s is the bolt carrier assembly returning forward and forcing another round into the chamber, and the final signature at 0.3 s is the trigger reset.

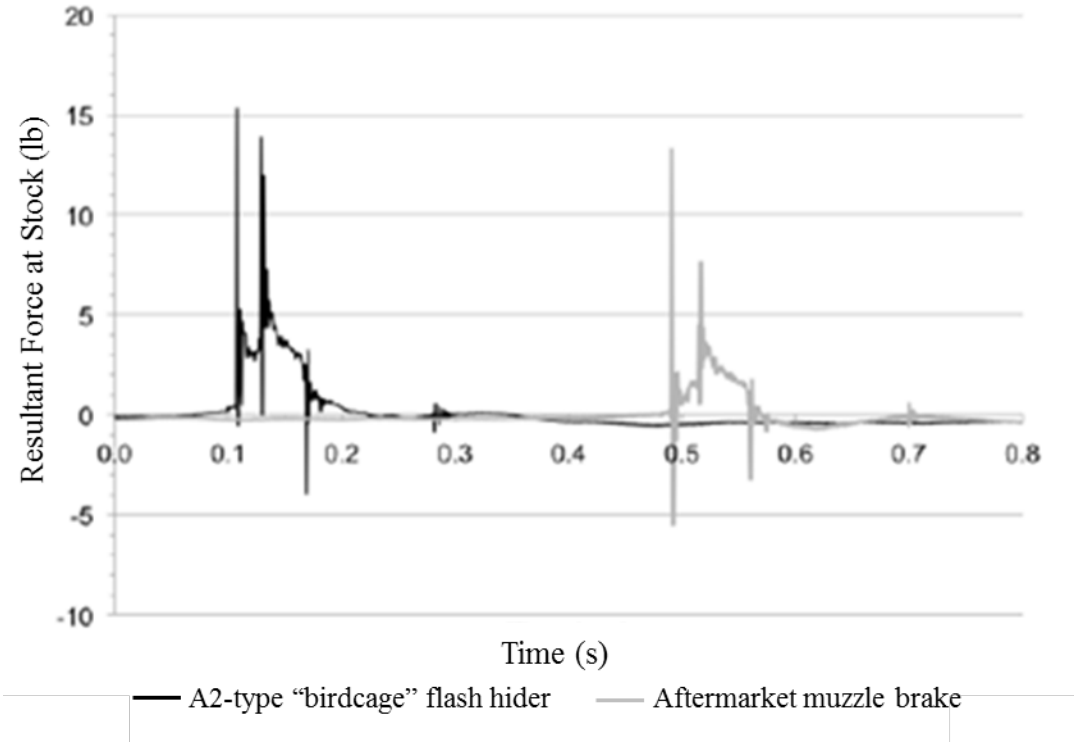

Figure 2. Force versus Time Curve for Two Shots from M4s with Different Muzzle Brakes. Adapted from [12].

Whether these events are measurable in digital form depends on the amount of time over which they take place. The force curves shown in Figure 2 suggest a cycle time of approximately 150 ms. In contrast, the Army operator's manual for 5.56 mm small arms lists the cyclic firing rate between 700–970 rounds per minute [13], or approximately one cycle every 61–86 ms. Regardless of this conflict, the individual events within the cycle occur over much shorter periods. For instance, the initial recoil of the weapon occurs while the projectile is traveling down the barrel [14]. Ignoring the complex gas physics and linearizing the projectile's velocity, which begins as stationary and increases to a certain muzzle velocity, yields an average velocity while in the barrel of 1/2 the muzzle velocity. Typical muzzle velocity for 5.56 mm projectiles is 3035 ft/s, and the typical barrel length

of an M4, the shortest barreled AR15 variant, is 14.5 inches [15]. With an average speed of 1517.5 ft/s, the projectile remains in the barrel for approximately 0.796 ms.

To validate this approximation of the time over which the initial recoil event occurs, the force curve of a 0.243 Winchester cartridge measured at the Army Research Laboratory is shown in Figure 3 [14]. This curve has an initial force, and therefore acceleration, that occurs over approximately 0.8 ms, confirming the approximation. The 0.243 Winchester cartridge is similar to the 5.56 mm (0.223 caliber) NATO round, and the 0.243 Winchester force curve, presented in Figure 3, provides a point of comparison not attainable from the 5.56 mm curves of Figure 2 due to the lack of temporal resolution. The 0.243 Winchester cartridge is typically used for hunting and, therefore, is not commonly used in semi-automatic rifles. The 0.243 Winchester force curve is from a single-shot rifle, which explains the single peak of initial recoil found in Figure 3, with no further peak forces generated by a reloading action.

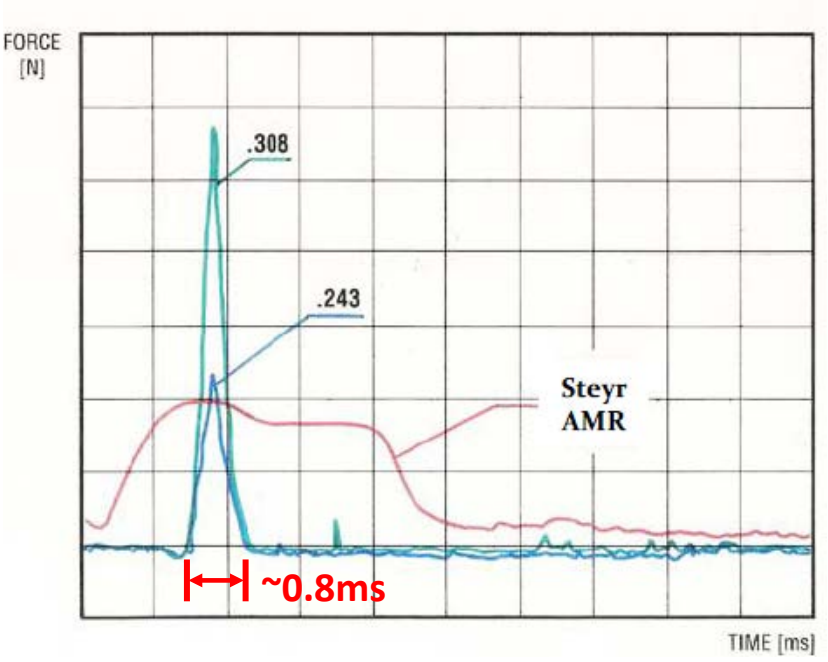

Figure 3. Force versus Time Curve for Three Different Weapons with Annotated Length of Initial Recoil Event. Adapted from [14].

To estimate the magnitude of this acceleration, a unit analysis was conducted. The recoil energy of a rifle based on certain parameters is

$$RE = \frac{W_g}{64.4}\left[\left(1.75W_p + W_b\right)\frac{MV}{7000W_g}\right]^2, \tag{2.1}$$

which was provided from [12]. In (2.1), *RE* is the recoil energy for the gun in ft-lb, W_g is the weight of the gun in lbs, W_p is the weight of the propellant in grains, W_b is the weight of the bullet, *MV* is the muzzle velocity in ft/s, 64.4 is twice the acceleration of gravity in ft/s^2, 1.75 is a unitless average of the effective velocity of propellant gasses, and 7000 is the conversion from grains to pounds. This recoil energy is converted into newton-meters with

$$RE(Nm) = 1.356RE\,(ft \cdot lb). \tag{2.2}$$

The units of a newton-meter are

$$Nm = kg\frac{m^2}{s^2}. \tag{2.3}$$

If the mass of the rifle and distance it travels during the event are known, the magnitude of the acceleration *a* in *m/s*^{2} can be obtained with

$$a = \frac{RE}{mRD}, \tag{2.4}$$

where *m* is the mass in kilograms and *RD* is the recoil distance of the rifle in meters. Using the mass of an M4 and a recoil distance of 0.5 inches yields an estimated rearward magnitude of acceleration of 18.72 g. It should be noted that the recoil distance is dependent on multiple factors including the specific shooter's stance, grip, and body composition as it relates to interaction with the rifle; therefore, while the shape of the shot profile remains consistent from shot to shot and shooter to shooter, the magnitude of the acceleration varies.

To summarize, the physical operation of the AR15 action provides several distinct and repeatable acceleration events, the shortest of which is the initial recoil event. This event is estimated to occur over 0.796 ms with an expected magnitude of approximately 18.72 g.

B. NYQUIST RATE

In this thesis, digital sampling is used to attempt measurement and identification of AR15 acceleration events. The cornerstone of digital sampling is the Nyquist Theorem. The Nyquist Theorem states that in order to obtain an accurate discrete representation of a continuous-time signal, one must sample at twice the rate of the continuous signal if that signal is band-limited [16]. Sampling more slowly creates an ambiguity as demonstrated in Figure 4. Three different signals return the same discrete representation when sampled at integers of period T.

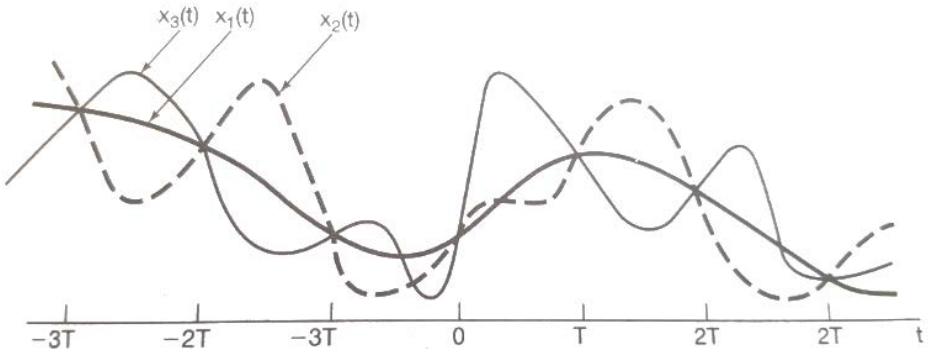

Figure 4. Three Continuous-Time Signals Sampled at Discrete Integers of T. Source: [16].

While no real signal is truly band limited, the Nyquist Theorem has important practical applications. As discussed in the last section, the period of the initial recoil event of the AR15 operation cycle is estimated at 0.796 ms; therefore, the frequency of this shortest acceleration event is the inverse of that period, or 1256 Hz. This requires a minimum sampling frequency of 2512 Hz to obtain accurate information about the initial recoil event. Even at 2512 Hz, the sample occurring during the initial acceleration event may not occur at the peak value, resulting in poor representation of the true signal.

The practical implementation of sampling is also relevant to this project. In many devices, sampling is accomplished through a zero-order hold scheme, where the most recent value is stored until the next value is recorded at the next integer value of period T

[16], as seen in Figure 5. If data is requested from a device whose digital output is the sample-and-hold data in Figure 5 at an interval smaller than T, the previous sampled value is provided.

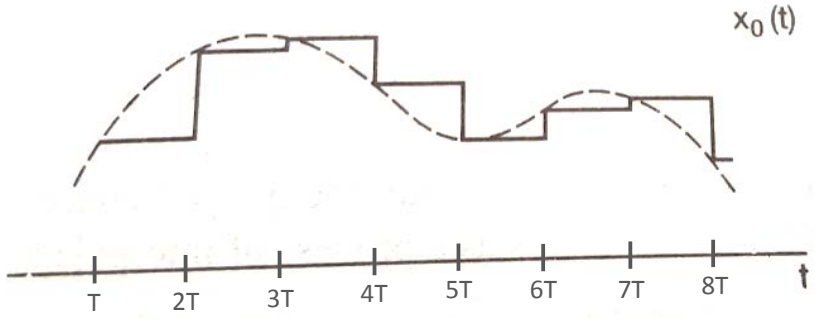

Figure 5. Example of a Sample and Hold Scheme. Adapted from [16].

The minimum sampling frequency of a device used to measure the initial recoil event of an AR15 is estimated at 2512 Hz. Attempting to query a device faster than it samples the signal will likely result in the previous value being resent.

C. MEMS ACCELEROMETERS AND AHRS

Many MEMS inertial measurement units (IMUs) on the market fuse data from accelerometers, magnetometers, and gyroscopes to provide orientation. Such devices are deemed AHRSs. The initial focus of this thesis relies upon accelerometer data. Later, the Euler yaw angle obtained through filtering of all three types of sensors is used during system integration to calculate the firing azimuth of the rifle. The basic theory of operation of the type of MEMS accelerometers found in the Yost Engineering Incorporated (YEI) 3-Space Sensor Data-logger (TSS-DL) used in this project is covered in this section. Additionally, a brief discussion of the decision to use the YEI TSS-DL is included.

1. Theory of Operation

MEMS accelerometers typically rely on either the piezoelectric effect or a change in capacitance caused by displacement of a proof mass. The piezoelectric effect is the phenomenon whereby certain crystalline materials under strain produce a voltage [17].

This voltage correlates to the amount of force causing the strain and, therefore, the acceleration via Newton's Second Law; however, the accelerometer used in this project relies on the capacitive change caused by a moving proof mass [18]. An example diagram of such a device is shown in Figure 6.

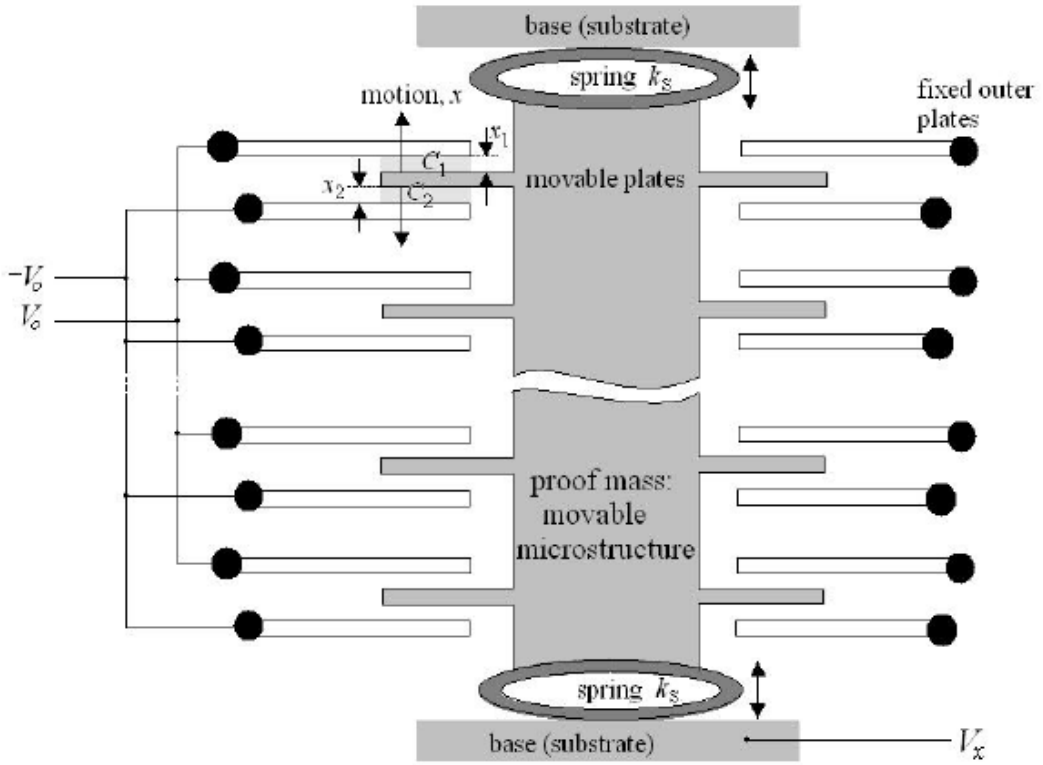

Figure 6. Capacitive-Type MEMS Accelerometer. Source: [17].

The method of obtaining a measureable quantity that is proportional to the acceleration of the proof mass is found in the derivation that follows, adapted from [17]. This derivation uses the terms shown in Figure 6 along with the original distance between the plates *d*. The displacements x_1 and x_2 are not used; instead, the displacement of the moveable plate from its original position x is used. The voltage V_o is applied to the stationary plates, and V_x is the voltage applied to the proof mass. The capacitances C_1 and C_2 are between the upper fixed plate and proof mass and the lower fixed plate and proof mass, respectively. The spring constant is depicted as k_s.

This derivation focuses on one set of plate capacitors, though real devices have many, as shown in Figure 6, to obtain a measureable change in voltage. First, the capacitance of parallel plates is defined as

$$C_0 = \varepsilon_0 \varepsilon \frac{A}{d} = \varepsilon_A \frac{1}{d}, \tag{2.5}$$

where ε_0 is the permittivity of free space, ε is the permittivity of the dielectric between the plates, *A* is the area of the plates, and *d* is the original distance between the plates. For compactness, ε_A is defined as the two permittivity values multiplied by the area. As acceleration acts in the plane of the device, one spring compresses and the other extends, resulting in the proof mass moving a distance *x*. This distance changes both capacitances, i.e., the capacitance between the top and bottom stationary plates and the proof mass moveable plate, as shown in

$$C_1 = \varepsilon_A \frac{1}{d + x} = C_0 - \Delta C \tag{2.6}$$

and

$$C_2 = \varepsilon_A \frac{1}{d - x} = C_0 + \Delta C. \tag{2.7}$$

If there is no acceleration in the plane of the device, then *x* is zero and C_1 and C_2 are equal. If there is acceleration, subtracting (2.6) from (2.7) yields

$$C_2 - C_1 = 2\Delta C = 2\varepsilon_A \frac{x}{d^2 - x^2}. \tag{2.8}$$

Solving the second equality of (2.8) yields the nonlinear equation

$$\Delta C x^2 + \varepsilon_A x - \Delta C d^2 = 0. \tag{2.9}$$

Because the displacement *x* is very small, the x^2 term is approximately zero and is neglected. With this approximation, the displacement is solvable. Additionally, the value of ε_A from (2.5) is substituted, yielding

$$x \approx \frac{d^2}{\varepsilon_A}\Delta C = d\frac{\Delta C}{C_0}. \tag{2.10}$$

The displacement x is now in terms of the change in capacitance and two constant values, the original capacitance and original separation of the plates. The charge of the three-plate system does not change, regardless of the movement of the middle plate, resulting in the relationship

$$\left(V_x + V_o\right)C_1 + \left(V_x - V_o\right)C_2 = 0. \tag{2.11}$$

Expanding (2.11) and rearranging the relationship between the proof mass voltage V_x and the stationary plate voltage V_o, we get

$$\frac{V_x}{V_o} = \frac{C_2 - C_1}{C_2 + C_1}. \tag{2.12}$$

Subtracting and adding (2.6) and (2.7) in the numerator and denominator of (2.12) yields (2.12) in terms of the change in capacitance

$$\frac{V_x}{V_o} = \frac{\Delta C}{C_0}. \tag{2.13}$$

Next, the mechanical system is analyzed. Hook's Law states that the spring force is proportional to the displacement x by the spring constant k_s. Dividing by the mass of the proof mass m results in its acceleration

$$a = \frac{k_s}{m}x. \tag{2.14}$$

Substituting in the right-hand side of (2.10) for x yields the acceleration in terms of the change in capacitance,

$$a = \frac{k_s d}{mC_0}\Delta C. \tag{2.15}$$

The final substitution of (2.13) into (2.15) gives the desired result of acceleration in terms of the voltage of the proof mass,

$$a = \frac{k_s d}{mV_o}V_x. \tag{2.16}$$

This voltage can be measured. The remaining terms of (2.16) are all measureable constants. This voltage signal of the proof mass is sampled by an analog-to-digital converter (ADC) for conversion into a useable digital signal. A calibration using the known acceleration due to gravity allows the MEMS device to provide the value of acceleration.

2. Device Selection

Several MEMS IMUs and one accelerometer were investigated for use in this project. In order to be considered, a device must provide accessible digital output without requiring hardware development. This criterion, which allowed for quick data analysis and rapid prototyping, resulted in sensors that are primarily USB or RS232 capable. Size was also considered, as the sensor needs to fit on a rifle without impeding the rifleman.

Consideration of these two requirements resulted in the sensors listed in Table 1; however, all but one of these sensors have output data rates less than the 2,512 Hz Nyquist rate of the initial recoil event as discussed in Section II.B. The sole exception is the Gulf Coast Data Concepts (GCDC) X16-1D sensor. This sensor utilizes an Analog Devices ADXL345 accelerometer with an output data rate of 3200 Hz; however, GCDC notes that above 400 Hz, inconsistent sampling begins to occur [19]. Additionally, the range of accelerations for all sensors falls below the estimated 18.72 g expected during the initial recoil event. If sampling could occur at a fast enough rate, allowing the true peak recoil acceleration to be detected, saturation would be expected in all of these sensors. The noise density and temperature sensitivity of these devices are all on the same order of magnitude. Even the worst-case noise density of the GCDC device operating at the highest frequency results in a noise floor of 16.97 mg, which for this application is a negligible quantity. Additionally, these sensors' ADCs ranged from 10 bits to 16 bits, which, in combination with the ranges provided, results in sensitivities between less than 0.1 mg to 3.91 mg. All of these resolutions are more than adequate for an application based on 18.72 g.

Table 1. Comparison of Sensing Devices. Adapted from [19]–[23].

Name of Sensor	Data Rate (Hz)	Range (g)	Cost ($)	Size (in3)	Weight (gr)	Data Access	Real Time	Noise Density (µg/√Hz)	Sensitivity (g)	Temp Sensitivity (%/oC)
YEI TSS Data Logger	1350	+/- 8	255	1.92	28	USB	Yes	99	.00096	+/-.008
Microstrain 3DM-GX4-25	1000	+/- 16	~3000	0.61	16	RS232	Yes	100	<0.0001	+/-.05
XSENS Mti-10 IMU	2000	+/- 15	~1000	3.51	55	USB	Yes	80-150	.000458	Not provided
Sparton IMU 10	2000	+/- 15	~2000	3.27	61	UART	Yes	200	Not provided	Not provided
GCDC X16-1D	3200	+/- 16	~80	~2.50	55	USB	No	~300	.00391	+/- .01

As none of these sensors can satisfy the Nyquist rate or the acceleration range necessary to uniquely identify the initial recoil event, an approach taking into account the entire firing and reloading cycle of the rifle was necessary. Choosing a device rested on cost, real-time capability, and previous experience within the department. Even though it was the lowest cost, the GCDC X16-1D was rejected due to its inability to utilize its data in real time. Of the remaining sensors, only the YEI TSS-DL and the XSENS Mti-10 IMU cost less than $1,000, putting them on the order of current rifle equipment such as the PEQ-15 laser designator and the Trijicon Advanced Combat Optical Gunsight. In addition, the YEI TSS-DL was used in previous Reticle Project work, resulting in working knowledge of the sensor within the department. Ultimately, the YEI TSS-DL was selected, allowing data collection and analysis to begin with equipment already in the laboratory.

D. RELATED WORK

This thesis work was conducted as part of the NPS Reticle Project. This project, focusing on networked infantrymen, originated with Captain Caleb Khan, USMC, who developed the concept based on the need for real-time de-confliction of geometry of fires in close-quarters combat scenarios [8]. In his thesis, an initial kinematic model of an M4 was developed to calculate rifle orientations in relation to one another to avoid friendly fire. Khan also investigated whether the YEI TSS-DL yaw angle error was sufficiently low for use in this application. Khan experimentally determined the dynamic error of YEI sensor's orientation readings to be +/- 4.07 degrees. Testing of his model showed that calculating the necessary riflemen offsets was both feasible and realistic given the error determined for the YEI TSS-DL. In addition, he successfully prototyped this de-confliction application with Ensign Jonathan Driesslein in a Publication and Subscribe (Pub/Sub) network established among multiple rifle nodes [8], [9]. This work also provided the three-dimensional (3D) printed brackets that attached the YEI TSS-DL to the Picatinny rail system found on most AR15 style rifles.

Conversations with Khan led to the question of other applications of networked infantrymen and to the stated research questions of this project. Conducting a literature review revealed that academic work regarding the intersection of firearms and electronics

is scarce due to its politically charged nature. Only one academic study had direct relevance to inertial measurement of firearms activity. Dr. Charles Loeffler, a criminologist at the University of Pennsylvania, conducted that recent study, which included tests to establish the acceleration profile of a wrist during the firing of a handgun with a data recording IMU watch [24]. His proposed application was the monitoring of criminals on probation and focused specifically on whether wrist accelerations from a handgun firing were distinct from other impulsive events on the wrist, such as using a hammer. Loeffler first identified potential shots by looking for peak accelerations immediately following stabilized periods required for aiming. He then looked at a 52.5 ms period of data around these spikes, taking various statistical parameters. He used a logistic regression model on these parameters and was able to classify 98.9% of shots accurately, with only 0.4% probability of producing a false positive on 693 "confusable spikes" [24]. While Loeffler post-processed his data and focused solely on handguns, his successful results show the feasibility of applying IMU data for shot identification of firearms.

Loeffler's envisioned application focused solely on the legal system. A wider application of the intersection of electronics and firearms would focus on the consumer market. In this market, the term "smart gun" typically refers to a firearm that is disabled unless operated by a particular person or subset of people [25]. One company in particular, Armatix, attempted to bring a .22 caliber pistol to market in the United States but failed due to political fallout, including threats against businesses preparing to sell the firearm [25]. Founded in 2013, Yardarm Technologies is a startup company that came close to suffering a similar fate to that of Armatix. Yardarm's focus has been on real-time weapons tracking. A first attempt at marketing to consumers immediately resulted in online death threats [26]. For this reason, Yardarm's marketing has been directed at law enforcement, security firms, and the military. Their system uses an AHRS embedded in the grip of a handgun that records orientation data and reports to the law enforcement dispatch system via the officer's smartphone during significant events. Their product informs dispatch when a firearm is removed from its holster or is fired. Yardarm has been field-testing the system with the Santa Cruz Police Department since November 2014 and is currently working with the Department of Homeland Security and the Dutch military [26]. A

conversation with Yardarm's Chief Executive Officer Bob Stewart on 29 January 2016 revealed that the company has branched out to AR15 variant weapons; however, its software is proprietary and was not made available for this thesis.

E. SUMMARY

The foundation of this project was laid in this chapter by first detailing the mechanics of the AR15 action. These mechanics provide the shape of the acceleration signal that is expected and a general idea of the magnitude of accelerations involved in firing an AR15. The topics of Nyquist rate and sample-and-hold schemes were presented due to their practical impact on hardware selection and algorithm development. Additionally, the means of deriving an acceleration signal from the voltage provided by a digital MEMS accelerometer was provided. The synthesis of the theoretical work for this thesis suggests that no COTS accelerometer will meet the Nyquist sampling criteria or have the dynamic range to measure the full recoil dynamics of AR15-variant weapons. In light of these shortcomings, the YEI TSS-DL was chosen due to its adequate rate for an under sampled approach and cost relative to other infantry electronics.

Finally, related work was discussed to put this thesis in the context of other Reticle Project work and show the feasibility of the research goals. Based on the limited academic work on firearm acceleration profiles, the concept proposed is novel for two reasons. First, it addresses real-time processing not discussed in the literature. Second, it extends Loeffler's [24] concept of using inertial measurements for shot-identification to semiautomatic rifles.

THIS PAGE INTENTIONALLY LEFT BLANK

III. EXPERIMENTAL DESIGN

A description of the hardware used throughout this project and details of the workflow are described in this chapter. Initial data sets were collected and a post-processing algorithm to identify shots was developed. Because this algorithm was built with complete previous knowledge of the data sets, it is deemed the "a priori" algorithm. Once this initial algorithm was tested, the code was reconfigured to identify shots in real time. With this real-time algorithm complete, the project shifted to system integration with an orientation sensing capability and a GPS receiver, along with COC code for receiving information from rifle nodes for mapping. Finally, the real-time code was translated for use on an embedded system.

A. HARDWARE

Selected hardware for this project include YEI TSS-DLs, several AR15-variant weapons, a GPS receiver and a Raspberry Pi microcontroller. A description of each follows.

1. YEI TSS-DL

The YEI TSS-DLs are used for Reticle Project applications due to their low weight, small form factor, and low cost. A YEI TSS-DL is shown in Figure 7. These AHRSs are capable of returning full orientation estimates, normalized data, corrected data, and raw data from triaxial accelerometers, gyroscopes, and magnetometers [20]. The YEI TSS-DL logs the data in its on-board memory as a text file or passes the data in real time in either a read-write configuration or a streaming configuration [20]. This project utilized all three operating methods to obtain data. The modes of the YEI TSS-DL are based on how the orientation estimates are computed. The orientation estimates are calculated via a Kalman filter, Quaternion Complementary (Q-Comp) filter, or Quaternion Gradient Descent (Q-Grad) filter [20]. These modes have differences in accuracy of estimation and in output data rate. The final mode of device is the IMU mode, which provides access to individual triaxial sensor data, but orientation estimates are not calculated. It should also be noted that the YEI TSS-DL can provide individual triaxial sensor data in every mode, but the increase

in data provided per sample slows down the effective sample rate while in the data logging configuration.

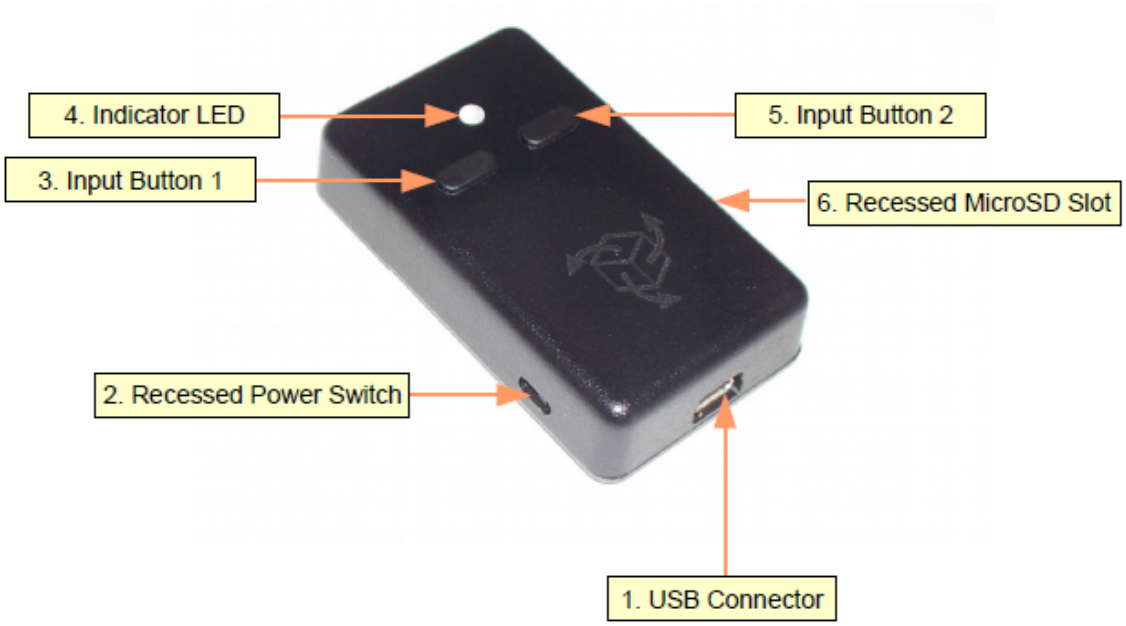

Figure 7. YEI TSS-DL. Source: [20].

The primary focus of this thesis is concerned with the corrected accelerometer data of the devices. The YEI TSS-DL utilizes the Freescale Semiconductor MMA8451Q 3-Axis, 14-Bit/8-Bit, Digital Accelerometer [27]. The MMA8451Q3 is a capacitive-based accelerometer that can provide data at up to 800 Hz [18]. The YEI TSS-DL itself claims to provide readings at 250 Hz when used in Kalman filter mode and up to 1350 Hz in IMU mode [20]. Experimental data logging sessions revealed that the actual frequency of output was related to the device mode, the type and amount of data requested from the device, and whether individual triaxial sensors were activated. This testing also revealed that in IMU mode, attempting to stream at the maximum data rate resulted in inconsistent data rates between 800–1350 Hz. For these reasons, initial data was collected while requesting only a microsecond timestamp and acceleration data. In Kalman mode, with this data requested, consistent time steps were obtained ranging between 4.3 ms to 4.9 ms. Data was also collected in IMU mode, with an averaged time step of 0.79 ms and in Kalman mode, with gyroscopes and magnetometers disabled, with a consistent data rate of 2.6 ms per sample.

2. Rifles and Setup

Multiple AR15-variant weapons were used throughout this project to ensure that any algorithm developed would be robust enough to deal with the variations of rifles found in infantry units. The rifles used included an M16A2, M4, a civilian AR15, and a California-legal AR15. The primary difference between the M16A2 and the M4 is the barrel length, which are 20 inches and 14.5 inches, respectively. The civilian AR15 lacks the fully automatic capability of its military brethren but is otherwise the same rifle with a barrel length of 16 inches. The California-legal AR15 is the same as a normal civilian AR15 but lacks a collapsible butt stock, pistol grip, and flash suppressor. All of these weapons have the same cycle of operations described in the previous chapter.

Previous Reticle Project work [8] provided 3D printed brackets for the YEI TSS-DL that were designed for the Picatinny rail system that most AR15 rifles utilize on the upper receiver and handguards. The YEI TSS-DL was attached to the handguards with the X-axis aligned with the barrel, along the direction of fire, as seen in Figure 8.

Figure 8. Rifle Setup with YEI TSS-DL Attached to an M16A2

This setup relies on two assumptions. The first is that the accelerations of the shot and reloading cycle act primarily along the axis of fire. Both the path of the bullet and the major moving pieces of the rifle are on this axis, making this a valid assumption. The main exception is the expanding propellant gasses being forced upward by the gas block into the gas tube; however, this acceleration should be negligible. The second assumption is that

the external frame of the rifle and the attached YEI TSS-DL are a single rigid body. This rigid-body assumption is likely valid as the upper and lower receivers of AR15 variant weapons are connected via two breakdown pins and the YEI TSS-DL case is screwed to the Picatinny rail. This assumption also allows for the force curves discussed in the last chapter to be applicable and YEI TSS-DL to be attached at any point on the external frame of the rifle as long as the alignment of the X-axis is correct.

3. GPS Receiver

The GPS receiver selected for the latter portion of the project is the USGlobalSat Incorporated BU-353S4. This receiver is "plug and play" via serial-over-USB 2.0 and has a small form factor of just 2.55 cubic inches and 2.2 ounces [28]. Additionally, the receiver provides data in the standard National Marine Electronics Association (NMEA) 0183-protocol sentences of GPGGA, GPGSA, GPRMC, and GPGSV. The GPGGA sentence provides the required fix data in the protocol shown in Figure 9.

GGA - essential fix data which provide 3D location and accuracy data.
$GPGGA, 123519,4807.038,N,01131.000,E,1,08,0.9,545,4,M,46.9,M,,*47

Where:

GGA	Global Positioning System Fix Data
123519	Fix taken at 12:35:19 UTC
4807.038,N	Latitude 48 deg 07.038' N
01131.000,E	Longitude 11 deg 31.000' E
1	Fix quality: 0 = invalid 1 = GPS fix (SPS) 2 = DGPS fix 3 = PPS fix 4 = Real Time Kinematic 5 = Float RTK 6 = estimated (dead reckoning)(2.3 feature) 7 = Manual input mode 8 = Simulation mode
08	Number of satellites being tracked
0.9	Horizontal dilution of position
545.4,M	Altitude, Meters, above mean sea level
46.9, M	Height of geoid (man sea level) above WGS84 ellipsoid
(empty field)	time in seconds since last DGPS update
(empty field)	DGPS station ID number
*57	the checksum data, always begins with *

Figure 9. GPGGA Sentence Protocol. Adapted from [29].

4. Raspberry Pi

In the latter portion of this project, a Raspberry Pi Model B+ was used as a test bed for implementing the shot-finding algorithm on an embedded system. The Raspberry Pi is a full computer with a Broadcom BCM2835 central processing unit with 512 megabytes of Random Access Memory (RAM) and storage provided by a 32-gigabyte MicroSD card [30]. The system is Linux based with multiple modules of the Python programming language already available for use. The Raspberry Pi has four USB 2.0 ports, making it an ideal candidate for this project, which ultimately incorporated two YEI TSS-DLs and the BU-353S4 GPS receiver, all of which provide data over USB connections. Additionally, the system is only 3.35 inches by 2.20 inches [30]. With its small size and significant computing capability, the Raspberry Pi provides a good test bed for assessment of the feasibility of the rifle node subsystem.

B. DATA COLLECTION AND A PRIORI ALGORITHM DEVELOPMENT

The strategy for algorithm development stemmed from sampling below the Nyquist rate. The entire operations cycle of the rifle, therefore, was taken into account. Data was collected to determine if slower acceleration events could be identified within the operations cycle of the rifle during a shot.

1. Range Day 1

Shooting occurred at Soledad State Prison under the cognizance of Travis Segura, the Naval Support Activity Monterey Police Department Firearms Instructor. The first data collection was designed to emulate the USMC Marksmanship Program's Table 1 course of fire that emphasizes stable firing platforms for focus on marksmanship fundamentals [31]. This ensured that shot profiles would be evident without accelerations due to shooter movement complicating identification. This initial data would be used for establishing a baseline algorithm. Three separate individuals, including one novice, fired from the prone, kneeling, and standing positions from 100 yards at man-sized targets. Different shooters firing from various positions creates variability in the shot profiles, ensuring the initial algorithm would be robust. Additionally, a final set of data was taken that included a magazine change and several rapid-fire sequences to provide a more realistic data set with

which to test the baseline algorithm. It should also be noted that the output data rate of the YEI TSS-DL throughout the day was approximately 227 Hz.

As expected, the shot profiles were easily identifiable due to the static, stable firing positions. The X-axis accelerations plotted against time from one of the sets of ten standing shots is shown in Figure 10. When the greater context is removed and each shot profile is individually examined, the effect of under sampling becomes evident. The shot profiles vary significantly due to under sampling.

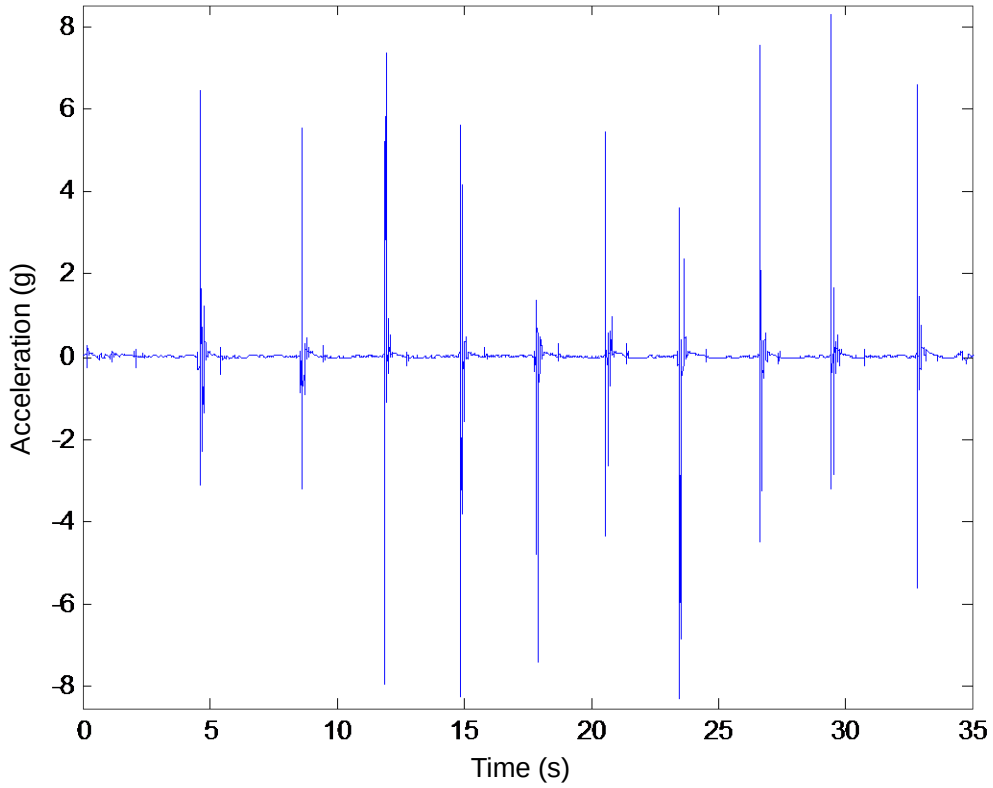

Figure 10. X-Axis Accelerations versus Time for Ten Standing Shots

The acceleration profiles for the first six shots of Figure 10 are shown in Figure 11. In the six plots of Figure 11, there are varying numbers and amplitudes of peak accelerations. Even more significant is the direction of the first large peak, which in plots b and c indicates the rifle moving forward vice the actual rearward acceleration of the initial recoil event. This is likely caused by the invalidation of the rigid body assumption

discussed in Section III.A.2. The explosive nature of the rifle firing and the metal-on-metal impacts of the reloading cycle create vibrations, which cause relative movement at one or more of three interfaces. These interfaces are between the YEI TSS-DL and its case, between the case and the upper receiver, and between the upper and lower receivers. Any algorithm developed must be capable of dealing with this level of ambiguity.

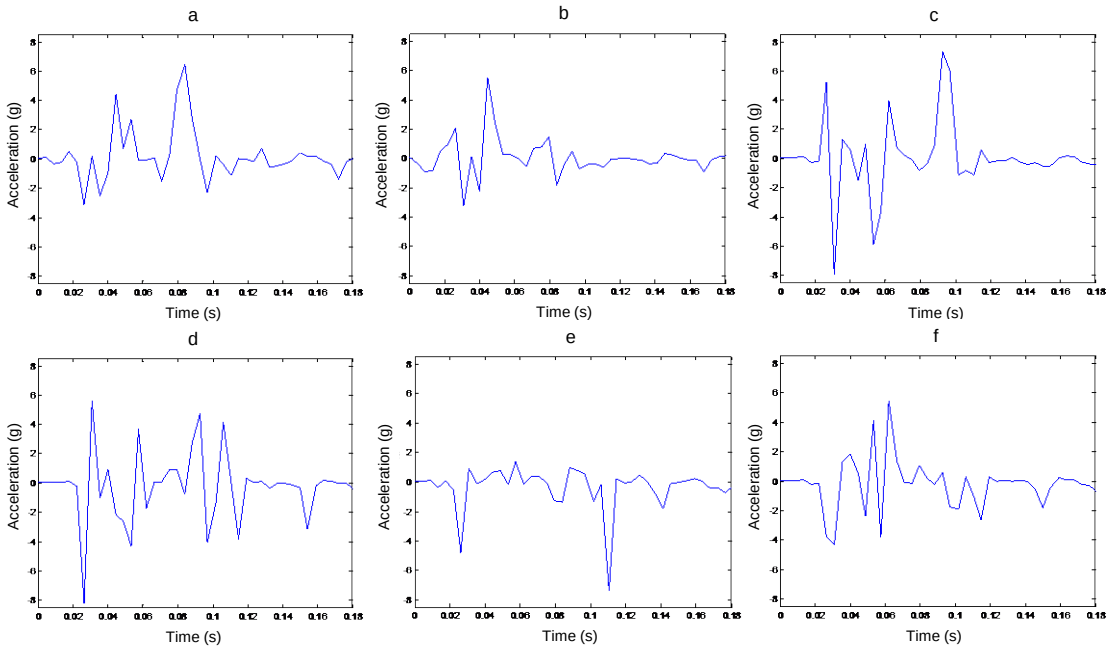

Figure 11. Acceleration Profiles of Six Individual Shots

2. Range Day 2

The second data collection involved dynamic shooting inspired by USMC Marksmanship Program's Tables 2 and 3 [31]. These courses of fire involve shooting while moving, rapid shots, and magazine changes. These actions provided a closer approximation to combat shooting and a better opportunity to evaluate any shot-identification algorithm developed. In two sequences of fire, inert dummy rounds were interspersed with live rounds. These dummy rounds feed properly into the chamber, but lacking propellant, do not fire when struck by the firing pin. Immediate actions are required to clear the weapon and re-engage the target with live rounds. These immediate action drills, along with dry firing, provided data sets

with the potential for producing false positives. Due to the likelihood that shots would not be immediately apparent from the data, each sequence of shooting was recorded on video so the timing of shots could be correlated with the data to identify the shots. The courses of fire, the sets of which were executed by two different shooters, are contained in Table 2. The courses of fire in Table 2 were chosen to provide realistic limits for parameters used in the algorithm. For instance, successfully engaging a man-sized target from ten yards requires much less aiming time and stability than from 100 yards. Firing while moving, referred to as a "combat glide," also provides insight into maximum boundaries for any aiming acceleration threshold.

Table 2. Range Day 2 Courses of Fire

Action	Number of Rounds	Firing Position	Distance (Yd)
Eject 30 rounds	0	Standing	N/A
Dry fire 6 times	0	Standing	N/A
Change Magazines 10x	0	Standing	N/A
Walk, Stop, Fire	10	Standing	100-50
Run, Stop, Fire	10	Standing	100-25
Quick Reaction, Rapid Shots	20	Standing	10
Combat Glide	10	Standing	25-10
Immediate Action 1	5	Standing	25
Immediate Action 2	5	Standing	25
Prone	10	Prone	100

3. Algorithm Development

A post-processing, shot-identification algorithm was written based on several assumptions and parameters and tested on the data sets obtained from the first two range days. The first underlying assumption is that all the data about the shot is available and known before analysis. For this reason, it is termed the "a priori" algorithm. The next assumption is that aiming occurs before each shot or series of rapid-fire shots. This assumption is the same as in Loeffler's work [24] and is the basis of marksmanship. In fact, the idea of "well-aimed shots" is foundational to the Marine Corps ethos of "Every Marine a rifleman" [32] and ingrained in Marines at entry-level training. Additionally, any rapid

follow-on shots occur before regaining a sight picture, without aiming. In the USMC Marksmanship Program, these two shots fired in quick succession are referred to as hammer pairs [33]. Finally, there are no more than two rapid follow-on shots taken without regaining a sight picture. This final assumption is based on service rifles having a three round burst capability. This is actually a wider bound than necessary as the USMC Marksmanship Program does not include training in three-round burst firing and only specifies two shots without regaining a sight picture [34]. Even these hammer pairs are only advised in extremely close engagements, as recoil causes a steep decline in accuracy for any shot taken without a steady sight picture immediately preceding. Only proper body positioning allows recoil management to keep the second shot of a hammer pair reasonably accurate [33].

These assumptions, along with analysis of the data sets, led to the selection of parameters needed for shot identification. These include time parameters, acceleration thresholds, and a differential power threshold. The time parameters include the complete cycle time for a shot and reloading, a rapid-fire window, a hammer-fall window, an aiming check time, an aiming length, and a recent-aiming window. All parameter values chosen in the algorithm were experimentally determined from the data sets obtained.

The cycle time is self-explanatory and is set at 162.8 ms. The rapid-fire window is the amount of time, from the beginning of the shot, during which the follow-on shot can occur without regaining another sight picture. It is established at 308.0 ms. The hammer fall window is the period prior to the hammer striking the firing pin but after the trigger press has released the hammer. This window is set at 13.2 ms. The aiming check time is how far back to assess the aiming acceleration threshold, and the aiming length is how long the rifle must be within that aiming threshold to be considered as aiming. Finally, the recent-aiming window is the maximum amount of time allowed prior to the shot breaking during which aiming must occur. The shot profile in Figure 12 shows the acceleration data of a shot with several of these parameters overlaid.

The shot-identification algorithm acts as a state machine, determining which state the rifle is in at each data point. The initial, default state is the “not aiming” state. Nine different criteria based on the previously discussed parameters are used to determine the

final state of each data sample. As the criteria are met, the data sample cascades through the states of aiming, shot candidate, aimed shot, first rapid shot, and second rapid shot.

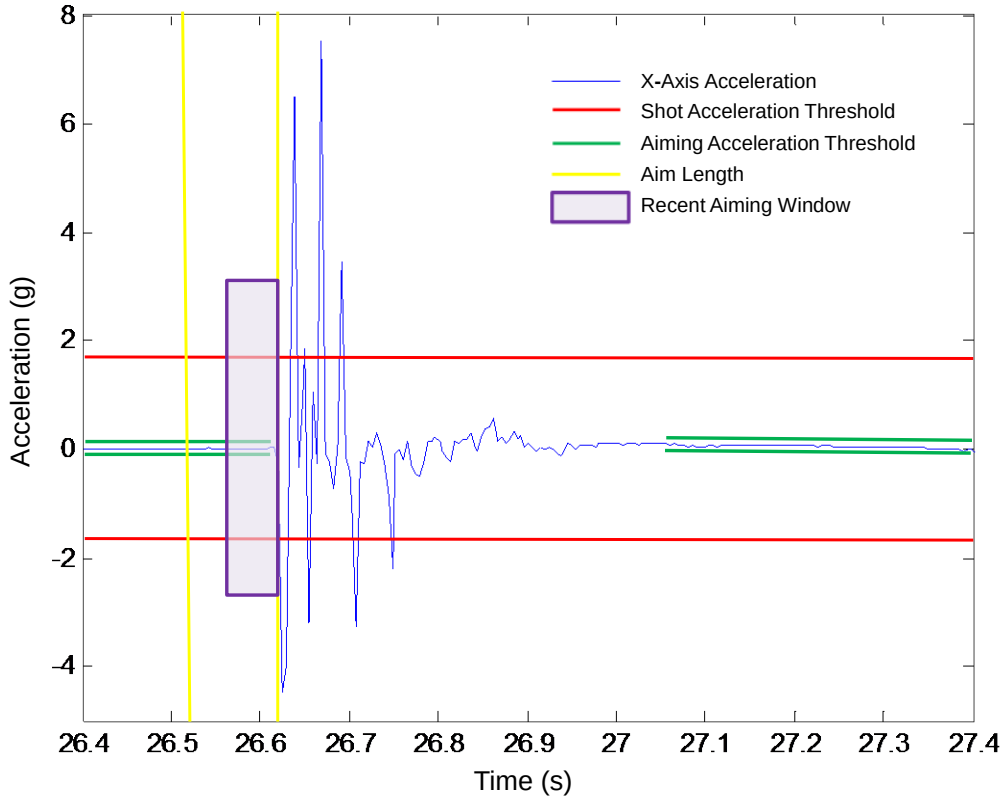

Figure 12. Shot Profile with Several Parameters Overlaid

The flow chart of Figure 13 shows this explicitly. The lowest state reached is that data sample's final state, which is denoted on future figures with a color-coded asterisk. The arrows going backward to the "not aiming" state indicate the next iteration. It should be noted that shot candidates and any shot state are cleared out within one cycle time of any shot state, which is discussed in detail in the following paragraphs. The criteria used for state determination is now discussed at length.

The aiming portion of the algorithm takes the current acceleration and compares it with the acceleration sample from 22.0 ms earlier. If the difference is within the set aiming threshold of 0.115 g, the rifle at that data point is listed as potentially aiming. This is not an absolute threshold, as one might infer from Figure 12, as it is based on a difference in acceleration data points, not the acceleration data itself. There is the potential for significant

spikes in acceleration between the 22.0 ms aiming check period. This is addressed by ensuring that there are at least 21 samples (92.4 ms) of consecutive potentially aiming points before a particular data sample is placed in the aiming state. This classification is implemented by a tracking array, which is binary.

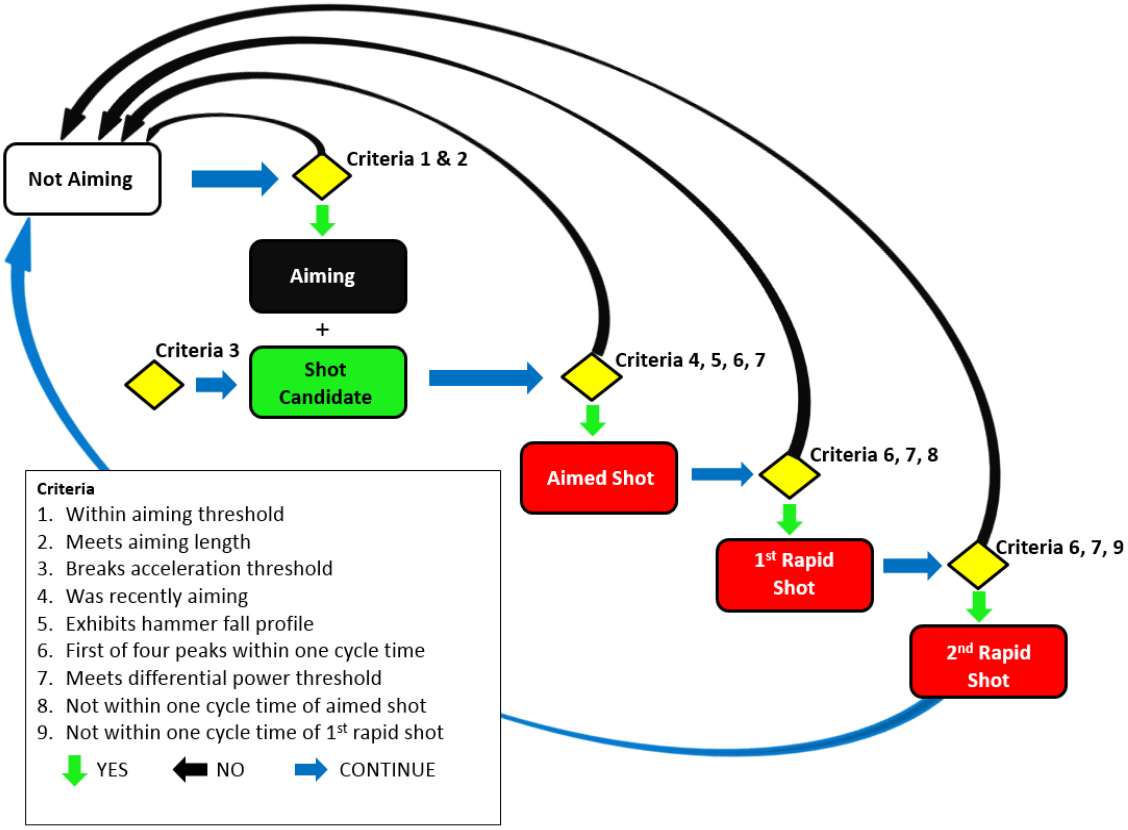

Figure 13. Algorithm State Flow

Next, the algorithm lists any sample outside the acceleration threshold of +/- 1.75 g as a shot candidate. This threshold is only a magnitude due to the ambiguity of the first peak's direction as discussed in Section III.B.1 and evidenced in the plots of Figure 11. It is set approximately ten times lower than the estimated value of the initial recoil acceleration due to the effects of under-sampling. Shot candidates are designated as aimed shots if they meet four conditions. The peak must be the first in a series of at least four peaks that occur within one cycle time, is within a recent-aiming window of 48.4 ms of a

data sample classified as aiming, exhibits a hammer fall profile immediately prior, and has an accumulated differential power above 87.95 W.

The first two criteria are self-explanatory, but the latter two are more complicated. The hammer-fall profile involves the absence of an aimed sample immediately prior to the shot candidate or a rapid change in acceleration in the previous three data samples (13.2 ms). This second requirement is necessary due to the time of the hammer fall being shorter than the aiming check time. This issue is illustrated in Figure 14.

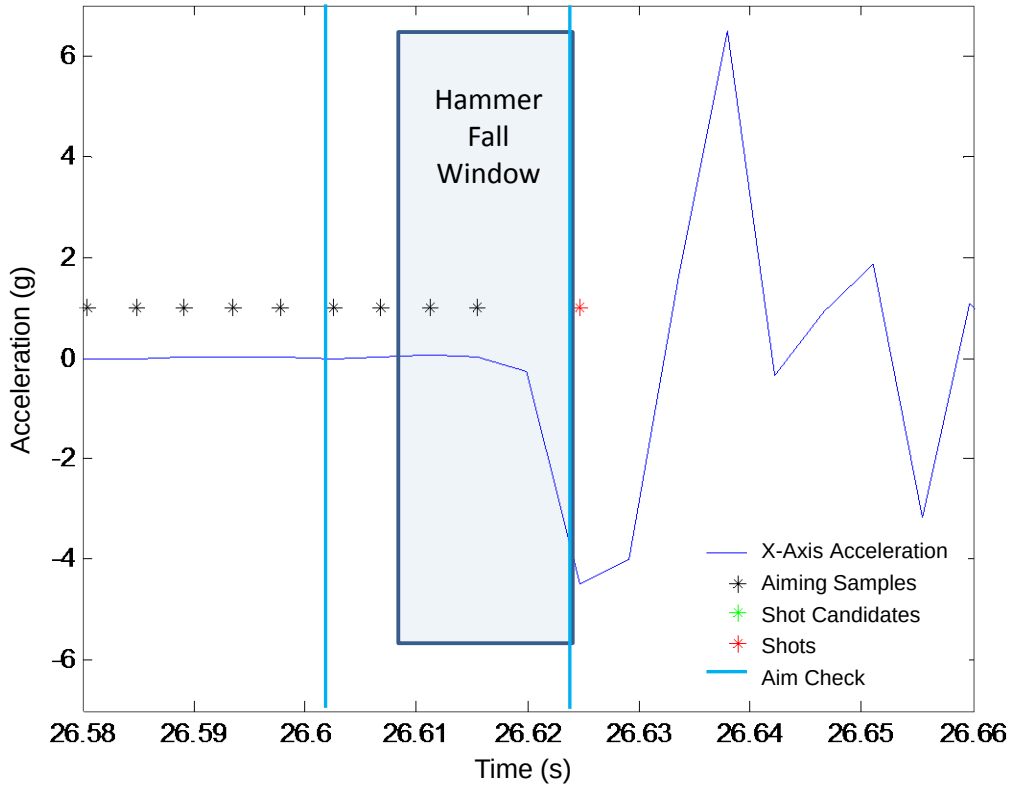

Figure 14. Hammer Fall Window Within the Aiming Check Period

The differential power is obtained by subtracting the previous acceleration data point from the current sample, squaring the result, and then summing these values over the previous cycle time. This essentially creates a moving, windowed value that is an indication of the total amount of change in the signal during a cycle time. This differential power eliminates all but the most significant acceleration events. The signal presented in Figure 15

demonstrates this effect, with the left plot showing three shots interspersed with running and the right plot showing the differential power of the signal over the same period. Note the portions of the differential power signal during running are negligible. It should also be noted that the differential power and number of peak accelerations are related, but differential power gives more weight to higher peak accelerations. Both of these parameters serve to distinguish actual shots from misfires or dry fires, which are part of a proper function check following maintenance. A misfire profile would be in the aiming state prior to a hammer fall and show a peak acceleration caused by the hammer striking the firing pin. Without the peak requirement and differential power parameter, a misfire would be identical to an actual shot. The differential power threshold was required due to the small magnitude of the acceleration peak threshold, an issue stemming from under sampling.

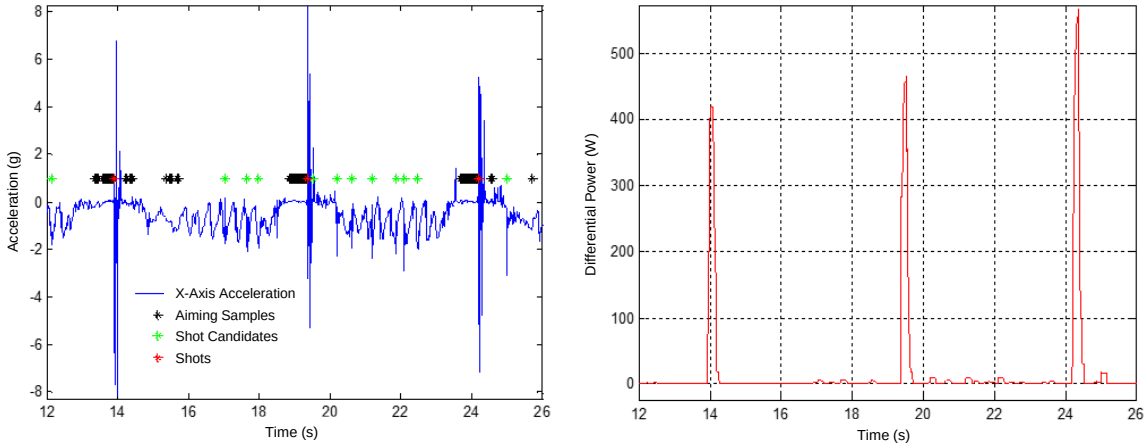

Figure 15. Original Acceleration Signal and Differential Power Signal

If the four criteria are met, the data sample is labeled as a shot; however, there is the possibility that more than one data sample meets these requirements during one reload cycle. In order to avoid duplication during the same shot, all labeled shots and shot candidates are erased if they occur within one cycle time of a previous shot. This ability to eliminate future classifications is only possible due to the a priori nature of post processing.

Nothing in the above logic takes into consideration rapid follow-on shots occurring immediately after an aimed shot. These shots do not occur within a recent-aiming window.

In order to account for this issue, the rapid-fire window parameter was introduced at 308.0 ms. Any shot candidate occurring after one cycle time of a previous shot, but within the rapid-fire window, is labeled as a first rapid follow-on shot. This same logic is applied to identify second follow-on rapid shots within 308.0 ms of a first rapid follow-on shot. This windowing is demonstrated on a signal of three rapid shots in Figure 16.

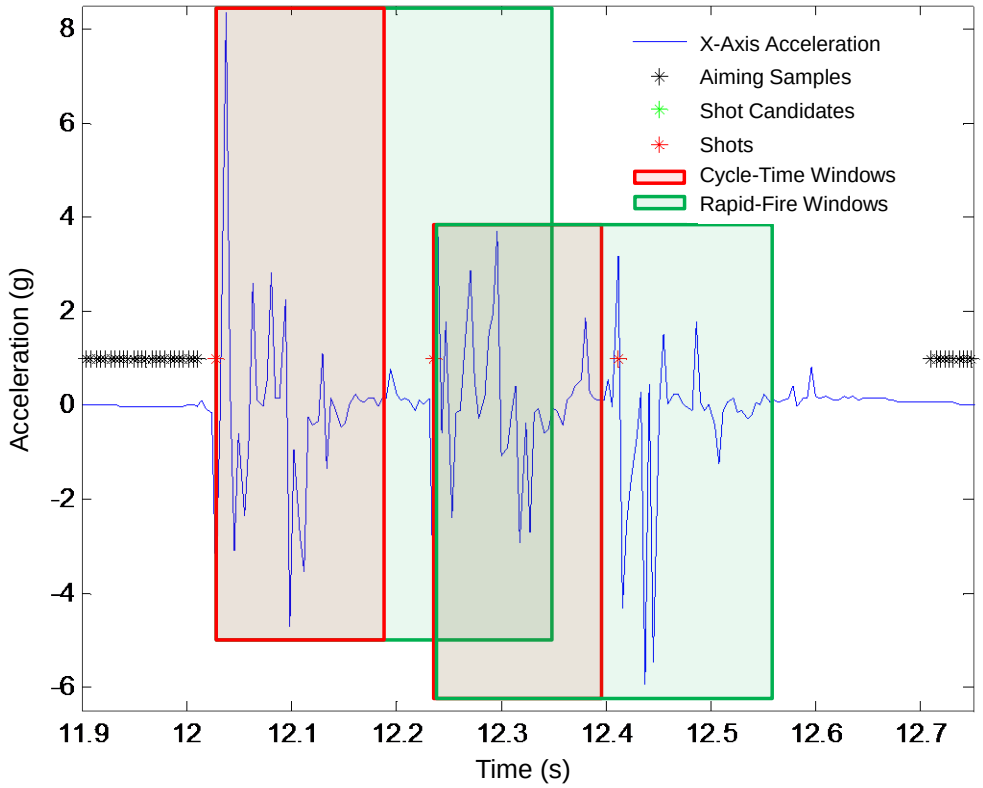

Figure 16. Example of Rapid-Fire Windowing

While a detailed discussion of the a priori results is found in the next chapter, from the red asterisks in Figures 15 and 16, it can already be seen that the a priori algorithm successfully identifies shots while running and shots in a rapid three-shot sequence. The MATLAB code for the a priori algorithm is found in Appendix A.

4. Further Data Collection

Following completion of the algorithm, additional data was collected from the YEI TSS-DL in IMU mode and in Kalman mode with the gyroscopes and magnetometers

disabled to obtain a faster data rate. These two modes returned data at average time steps of 0.79 ms and 2.6 ms, respectively. These faster data rates were used in hopes of obtaining digital data with greater fidelity to the actual continuous-time signal. The time-based parameters of the algorithm were adjusted due to the new average data rates and their effect on indexing. These separate courses of fire were conducted with a civilian variant of the AR15, a Smith and Wesson Military and Police 15 (MP15) rifle. These courses of fire were video recorded and are found in Table 3. The results of the a priori algorithm on each course of fire are found in Appendix B.

Table 3. MP15 Courses of Fire

IMU Mode			
Action	**Number of Rounds**	**Firing Position**	**Distance (Yd)**
Combat Glide	5	Standing	25
Kneeling	10	kneeling	50
Magazine Change	10	Standing	25
Quick Reaction, Rapid Fire	10	Standing	10
Run, Stop, Fire	5	Standing	50-25
Standing	10	Standing	50
Walk, Stop, Fire	5	Standing	50-25
Kalman 2.6 ms Mode			
Action	**Number of Rounds**	**Firing Position**	**Distance (Yd)**
Kneeling	10	Kneeling	50
Kneeling	10	Kneeling	50
Kneeling	10	Kneeling	50
Magazine Change	10	Standing	25
Eject 5 Rounds	0	Standing	N/A
Fire, Clear Misfeed	1	Standing	50
Standing to Kneeling	5	Standing to Kneeling	50

C. REAL-TIME ALGORITHM CONVERSION

All sample codes provided for the YEI TSS-DL and real-time code developed in the previous Reticle Project work [9] were written in Python and C++. MATLAB was used

for post-processing data and never for interacting with the YEI TSS-DL in real time. Before the a priori code could be converted, code for real-time streaming of data from the YEI TSS-DL was built and tested incrementally. Interacting with the sensors requires sending serial commands listed in the back of the YEI manual [20]. These commands are formatted as 8-bit unsigned integers and are assembled into packets. The packets are comprised of a start byte, command byte, any data associated with the command, and a checksum. Any data returned in response to a sent packet must be parsed appropriately. In order to obtain timestamped data, a response header must be set in the device. After it is set, any data requested with a header start byte has the header prepended. This header start byte is used with the "start streaming" command so that all streamed data will have the header. While this command does not provide a response, a header is still sent, which must be parsed before the first iteration of data. Additionally, timing parameters of interval, duration, and delay can be set [20]. The interval was set to a minimum to allow the sensor to send data at the maximum rate. The duration was set to allow continuous streaming, while the delay was set to zero, instructing the YEI TSS-DL to stream immediately. With correct timestamped accelerations streaming, transfer of the a priori code to a real-time architecture can proceed.

The a priori algorithm developed in the previous section relied on having the full data set available for analysis. At any particular point in time, future data can be used to assess previous data or can be set to a particular value without being overwritten during the next iteration. For implementation into a system requiring real-time data, this assumption is invalid. This inability to access and clear future data required setting additional conditions for shot identification. For example, instead of clearing any aimed shots one cycle time after the first aimed shot, an aimed shot classification now required that there are no aimed shots within the previous cycle time. This ensures that any new aimed shot is not an artifact of a previous shot. The largest consequence of this change in structure relates to the indexing of all the tracking arrays at the beginning of the algorithm. The largest time parameter was the rapid-fire window at 308.0 ms. Because the logic relies on the previous 308.0 ms of data, the initial 308.0 ms of data cannot be assessed at startup.

In addition to this restructuring, timing was again considered. The YEI manual indicates that the various modes of the device have different output data rates for USB operation and data logging [20]. It was experimentally determined that the fastest data rate obtainable with the YEI TSS-DL that provided consistent data steps was 833 Hz. This was achieved in Q-Comp mode with only the accelerometer enabled. This aligns closely with the 800 Hz value provided in the MMA8451Q accelerometer specification sheet [18]. This time step of 1.2 ms was initially used for the real-time algorithm. Additionally, there are 13 tracking arrays, each growing at this rate. In order to limit data size, a size of 50,000 samples was chosen as a reset point, resulting in a reset approximately every minute. For analytic purposes, all previous reset data is captured, though in a real-world application this data would not be stored on the rifle node.

The full system, which is described in the next section, was tested in real time on a California-legal AR15 variant. This "featureless" weapon is shown in the Figure 17. It is cosmetically different from the weapons used for data collection during a priori code development, but the AR15 action remains the same.

Figure 17. Featureless Rifle with Both the Shot-Identifying YEI TSS-DL and Orientation YEI TSS-DL

After encountering issues during this first real-time test, the system was tested at a slower data rate of approximately 220 Hz on the civilian MP15 rifle. The issues from the first test and detailed discussion of the system setup in both tests are found in the following chapter. The final versions of the real-time MATLAB codes are found in Appendix A along with codes that re-create the shooting events from both the COC and rifle nodes.

D. INTEGRATION WITH GPS, ORIENTATION, AND COC MAPPING

With the initial testing of the real-time algorithm completed, work turned to answering the question of how this information could be used to increase SA. In order for the Alert and Direction portions of ADDRAC to be satisfied, both the location of the infantryman and the firing azimuth of the rifle at the time of the shot must be communicated to the COC. Additionally, the unit leader, or COC node, must have a means to put the information in context. These issues are addressed in the following sections.

1. GPS

For dismounted infantry operations outdoors, GPS is the simplest localization method currently available. Other Reticle Program projects have focused on localization in indoor and GPS-denied environments [27], [35], but this was not the focus of this thesis. For this prototype system, code was written to parse the GPGGA sentences provided by the BU-353S4 to provide the latitude and longitude for the rifle node. The GPS MATLAB code is found in the real-time rifle node code of Appendix A.

2. Orientation

The orientation of the rifle was found via two methods. First, a second YEI TSS-DL was used. This was the configuration used in the first real-time testing, as seen in Figure 17. A second sensor was initially thought necessary due to the significant difference in data rates caused by requesting full orientation data from the sensor. This assumption, however, was based on data rates obtained in data-logging mode. Further testing showed that when streaming, an increase in the amount of data requested from a YEI TSS-DL had a minor effect on data rate. The real-time algorithm was converted to obtain acceleration data and orientation data from one YEI TSS-DL. The orientation code is found in the real-time rifle node code of Appendix A.

The Euler yaw angle of orientation provided by the YEI TSS-DL is based off magnetic north. This angle is corrected to true north with the geographically dependent Grid-Magnetic (G-M) angle, which for Monterey, California, is 13.38333 degrees. Another option is to sight the rifle on true north and tare the YEI TSS-DL. This taring applies a

constant correction accounting for the G-M angle. Due to the right-handed nature of the YEI TSS-DL reference frame assigned, the yaw angles to the east are negative values. East angles are negated to obtain the correct azimuth. Additionally, the Euler yaw angles range from $-\pi$ to π vice zero to 2π. This requires the positive west values to be subtracted from 360 degrees to provide the correct azimuth.

Unfortunately, solving the direction portion of ADDRAC is not as simple as the previous paragraph portends. There is an inherent inaccuracy in the sensor, as tested in previous Reticle Project work [8]. This inaccuracy is caused by lagging due to computation time and by noise, which is present in all sensors. Compounding these issues is the influence of variations in the local magnetic field on the magnetometers of the YEI TSS-DL, which cannot be known in advance. For laboratory purposes, the orientation sensor was calibrated in the position used for testing on NPS in Spanagel Hall and then tared at true north. The means of obtaining an approximate ground truth and applying further correction factors can be found in Appendix D.

3. COC Mapping

SA cannot increase without information placed into context; thus, the manner in which data is displayed for the unit leader or COC is important. MATLAB's Mapping Toolbox provides a web-mapping function to overlay data on maps and imagery; however, this function requires internet access to obtain data from the chosen map server. This function also proved time consuming. Instead, Google Earth was chosen. This COTS product is user friendly and caches up to two gigabytes of data. Any raster imagery recently viewed is stored locally and can be accessed when offline, which would be necessary for real world operations.

Google Earth uses Keyhole Markup Language (KML) files to display geographic information overlaid onto imagery. The MATLAB functions *kmlwrite* and *kmlwriteline* were used to create these overlays, which include rank and name specific images for marking rifle node locations. In addition to the GPS location and firing azimuth with timestamp, a GPS error ring and error azimuths are included. All of the azimuths were made 800.0 m in length, as this is the doctrinal maximum effective range of the M16 for

an area target [13]. The sizing of the GPS error ring and azimuth error is discussed in the next chapter. An example display in Google Earth, generated after a simulated shot, is seen in the screen shot of Figure 18. The weight of the lines in Figure 18 has been increased from the original screenshot for clarity.

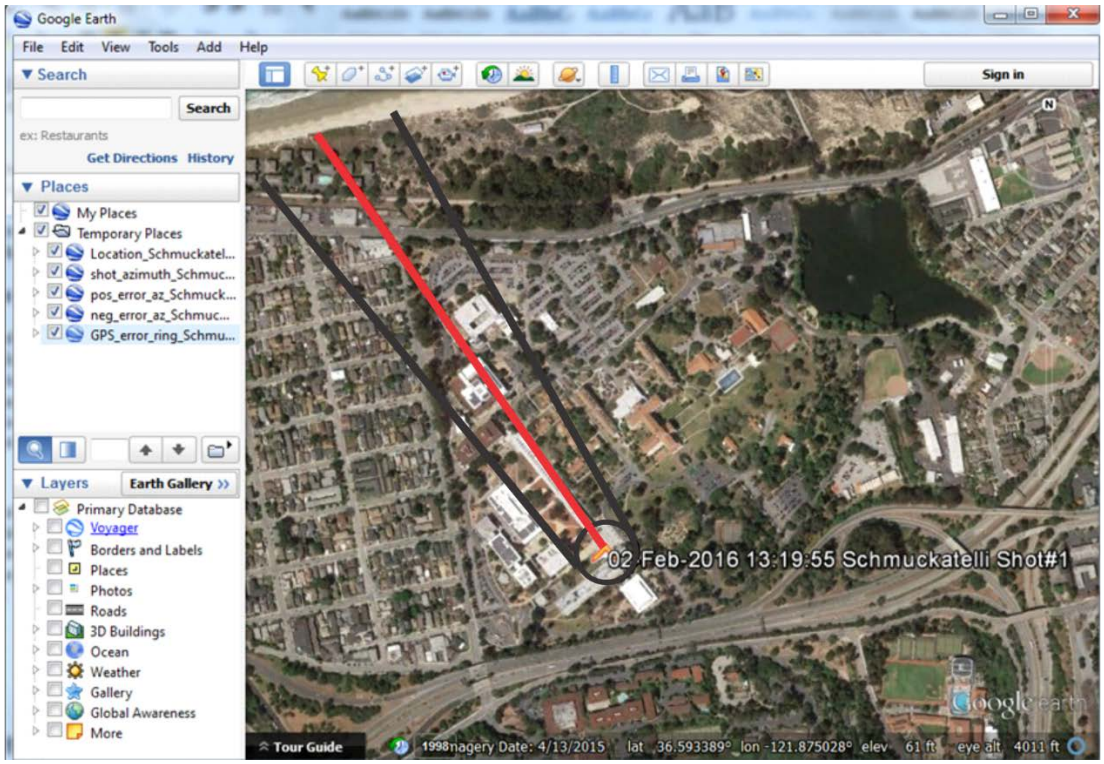

Figure 18. Google Earth Overlay After a Simulated Shot

In order to generate the KML files, individual latitudes and longitudes along the lengths of the azimuths and around the GPS error ring had to be calculated. To find the endpoints of the azimuths, with the exception of the rifle node location provided by the GPS receiver, the MATLAB function *reckon* was used to calculate new latitude and longitudes at a provided range from the initial set of coordinates. Next, the function *track2* calculates 100 coordinate points between the pairs of endpoints along the World Geodetic System 1984 (WGS84) ellipsoid model of the earth. Similarly, the function *scircle1* is used to generate 100 coordinate points at a provided radius around the rifle node coordinates. These series of coordinates are passed in the KML file format to Google Earth for plotting.

The code developed for calculating these overlays and for mapping in Google Earth is found in Appendix A.

4. Event Based Architecture and Networking

Previous Reticle Project work laid out a Pub/Sub network architecture for networked infantry units [9]. Under this setup, the small-unit leader and COC can simply subscribe to the GPS location and firing azimuth topics of subordinates. Logic in a broker node is then responsible for linking the events temporally at the time of a shot for display purposes. For this project, Pub/Sub architecture presents the issue of timing as it relates to the integration of multiple, simultaneous operations. GPS provides readings every second, while the YEI TSS-DL(s) have significantly faster data rates. MATLAB is a poor choice for parallel computing, which is needed to have GPS acquisition and, in one system configuration, orientation finding operations running in the background of the shot-identification algorithm. For this reason, an event-based architecture was built.

In the two YEI TSS-DL version of the system, the rifle node continuously tracks X-axis accelerations but only obtains GPS location and firing azimuth upon detection of a shot. This system setup has the advantage of slightly faster data rates and less memory usage at the expense of more hardware. The single YEI TSS-DL version continuously tracks both acceleration and yaw angle of the rifle. This allows greater accuracy of the firing azimuth, as the yaw angle passed to the COC is obtained from the last index classified as aiming prior to the shot breaking vice after the identification of a shot. While this is only a difference of several milliseconds, as the shot is identified before the reloading cycle of the rifle is complete, it removes any unintended changes in the orientation of the rifle caused by the dynamics of the shot. The single YEI TSS-DL system, therefore, uses more memory, less hardware, and is more accurate.

The connection for this prototype system is Transmission Control Protocol/ Internet Protocol, allowing use of the NPS network. Any computer can be set up in the server role to act as the COC, accepting data from any number of rifle nodes. The rifle nodes are set in the client role, requiring the Internet Protocol address of the COC. The system was tested successfully with two rifle nodes passing data to a COC computer for mapping. In field

testing, two separate instances of MATLAB were opened, one running the COC node and one running the rifle node. Data is passed between the two instances of MATLAB using the same *tcpip* function but with the IP address of the COC listed as "local host."

5. Move to Embedded System

With the full system prototype coded in MATLAB on a desktop computer, the shot-identification code was translated into Python for operation on the Raspberry Pi B+ microcontroller, which is seen in Figure 19 with two YEI TSS-DLs and GPS receiver attached.

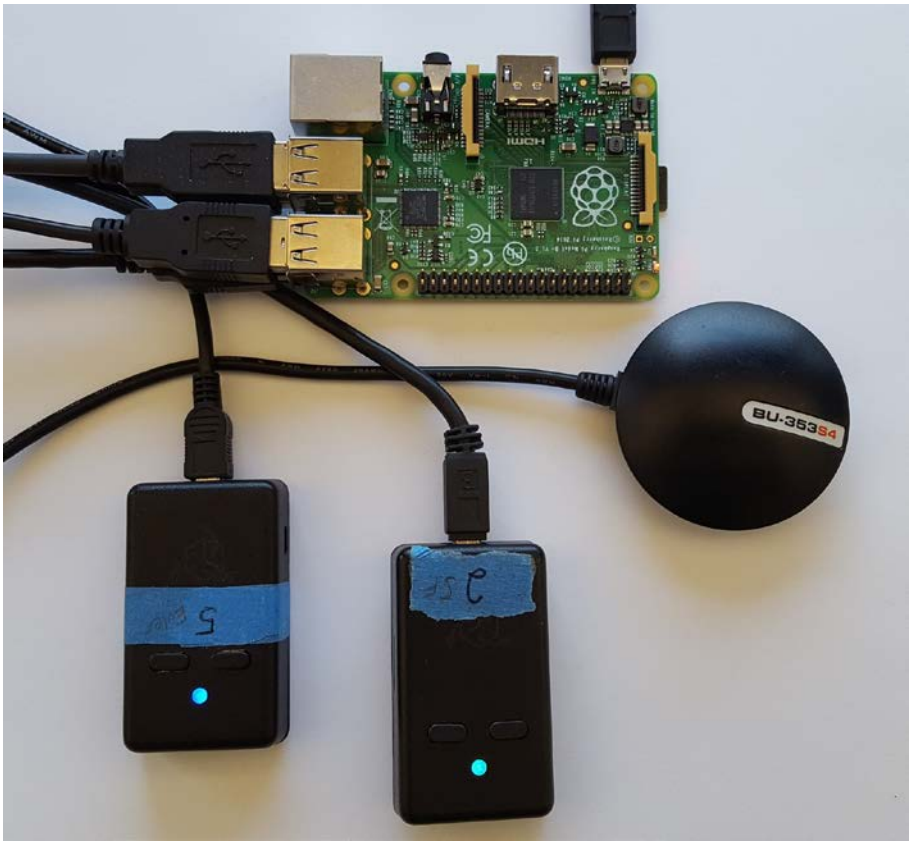

Figure 19. Raspberry Pi with YEI TSS-DLs and GPS

The layout of the Python algorithm has several notable differences from the MATLAB program. First, in order to simulate the mathematics, serial, and timing capabilities inherent in MATLAB, the modules *serial*, *struct*, *time*, *numpy*, and *math* were

imported. Next, the GPS code and firing azimuth codes are written as their own functions prior to the main program. In the main program, interaction with the shot-identification YEI TSS-DL is conducted in a read/write paradigm vice the streaming nature of the MATLAB code. While streaming code was written for the Raspberry Pi, the read/write interaction was more reliable. Additionally, building the various tracking arrays is done through appending a new value vice replacing a previous value as in MATLAB. Due to the limitations of the Raspberry Pi, each tracking array is limited in size to 100 values. This avoids unnecessary memory usage. This length of 100 values is based on a timing between samples of 5.0 ms; however, even with this limited data usage and relaxed data rate, timing became an issue. This timing issue of the embedded system is discussed in the next chapter. The rifle-node Python code is found in Appendix A.

E. SUMMARY

The progression of this thesis was covered in this chapter by first describing the hardware used during the project. Next, the bulk of the work, collecting data and developing the shot-identification algorithm, was discussed. The conversion of that algorithm into a real-time architecture was detailed, followed by its integration in a full system that includes an orientation capability, GPS receiver, and COC node for mapping. Finally, the rifle node program was translated into Python for use on an embedded system.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. RESULTS

The organization of the previous chapter is followed in this chapter. The results and issues associated with each step of the work progression are detailed, and the a priori algorithm results, the real-time results, system integration issues, and embedded system conversion results are discussed.

A. A PRIORI ALGORITHM

The a priori code was successful despite lacking full information about the acceleration signal due to sampling below the Nyquist minimum rate. The full results are contained in Tables 6-10 in Appendix B. During the first two range days, 170 rounds were fired with the YEI TSS-DL operating in Kalman mode with sampling rate of approximately 227 Hz. An M16A2 was used during the first range day, while on the second range day an M4 was used. In post processing, the a priori algorithm accurately identified every shot and returned zero false positives. Parameter values used in achieving these results are found in Table 4.

Table 4. A Priori Parameter Values

Parameter	Value
aiming_threshold	0.115 g
accel_threshold	1.75 g
aim_length	92.4 ms
aim_check	22.0 ms
cycle_time	162.8 ms
rapid_time	308.0 ms
recent_aiming	48.4. ms

Additionally, the algorithm overlays on the acceleration data of ten shots from several rapid-fire sequences with a magazine change in between are shown in Figure 20.

The algorithm successfully parsed all ten shots, as shown by the red markers in the plot while rejecting the accelerations caused by the magazine change.

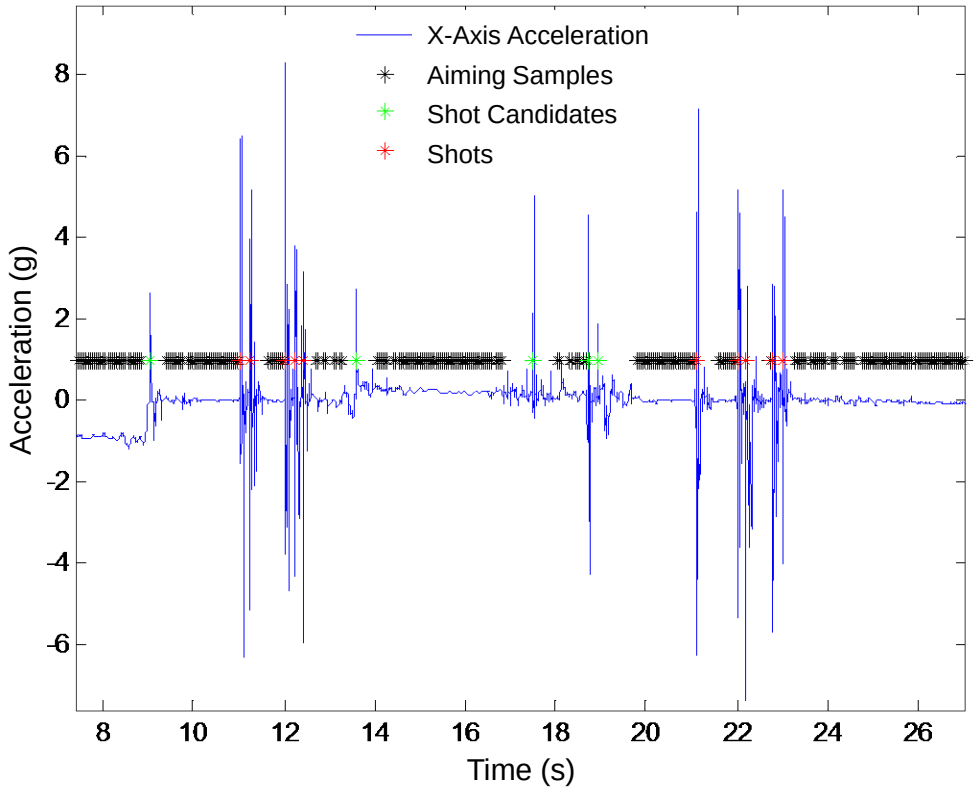

Figure 20. Algorithm Overlays on the Acceleration Signal from a Magazine Change Between Rapid-Fire Sequences

Later, data collected on the civilian MP15 rifle with the YEI TSS-DL set for sampling at an average of 1266 Hz and 384 Hz initially returned three false positives and one missed shot out of the 101 shots fired. This represents a 99.1% success of identification with a 2.97% false positive rate and a 0.99% miss rate. Two of the false positives and one missed shot occurred during processing of acceleration data collected at 1266 Hz. At this rate, increasing the number of shot candidate peaks during a cycle time from four to seven eliminates one of these false positives. This is an intuitive fix as sampling at a greater rate increases the likelihood that peaks will be detected. The other false positive is eliminated if the peak acceleration threshold is increased to 3g. Again, an increased sampling rate

means a greater chance of accurately recording peak values and provides greater resolution between actual shots and signal profiles that are similar. Neither of these parameter changes affected the results of the algorithm on the other sequences of fire. The missed shot in the IMU data was a rapid-fire shot that occurred significantly outside the rapid-fire window but without aiming. This is attributed to poor shooter performance. With a rapid follow-on shot occurring that long after the initial shot without regaining a second sight picture, the shot was likely not accurate. Perfect shooter performance cannot be expected during combat, but extension of the rapid-fire window increases the likelihood of false positives due to the less stringent requirements for shot identification in this window. Inside the rapid-fire window, the aiming requirement is necessarily removed, and any peak breaking the +/-1.75 g threshold is considered a shot. Further, the hammer fall requirement is not taken into account in the rapid-fire window as this low acceleration event is only noticeable due to the stability of aiming that precedes the hammer fall; however, both the number of peaks requirement and differential power thresholds are applied to follow-on shots.

Another technical observation that deserves comment is the limitation of the YEI TSS-DL in IMU mode. The sensor is commanded to return data at its maximum rate, claimed to be 1350 Hz, but the MMA8451Q accelerometer only provides data at approximately 800 Hz. This results in the YEI TSS-DL frequently returning a previous sample value as discussed in Section II.C. Many examples of this occurrence are found in the data shown in Figure 21, giving this data its “blocky” nature in comparison to the plots of Figure 11. Additionally, the varying size of time steps makes using array indices for timing purposes an invalid assumption but not one that adversely affected the results.

The one false positive collected during the Kalman, 384 Hz data set occurred at the end of a firing sequence. When clearing the rifle, the bolt carrier assembly was manually brought to the rear of the rifle and released while the rifle was oriented at the ground. The rifle was determined to be in the aiming state due to lack of acceleration caused by tension in the sling supporting the rifle. The hammer fall profile logic also happened to be satisfied. Multiple peak accelerations were caused when the charging handle was released, returning the bolt carrier assembly forward. While these peak accelerations would normally not have met the +/- 1.75 g threshold, the acceleration due to gravity provided an additional -1 g.

This false positive is removed if the acceleration threshold is set to +/- 3 g, as in the IMU algorithm, and the recent-aiming window is reduced from ~49 ms to ~44 ms. Lowering the recent-aiming window, however, results in missed shots in the 4.4-ms Kalman data sets, requiring that this false positive be accepted. Intuitively, the recent-aiming window should be set slightly longer than the hammer fall profile; however, this does not allow for poor trigger control by the shooter, colloquially defined as “jerking” or “pulling on” the trigger, or the likelihood of the first peak acceleration threshold not being captured due to under sampling. This false positive does reveal the limitation of the increased likelihood of false positives when the rifle is oriented along the gravity gradient with the lower acceleration threshold of +/-1.75 g.

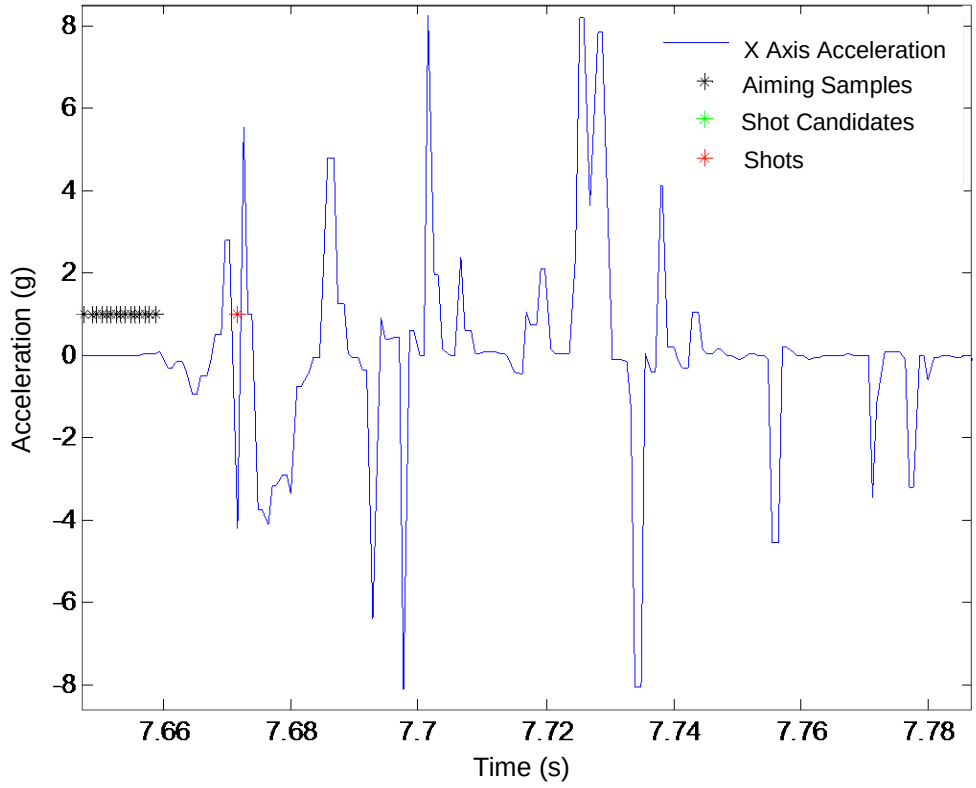

Figure 21. Sample-and-Hold Scheme is Evident in the “Blocky” Nature of Acceleration Data Caused by Requesting Data Above 800 Hz

Taking into account the IMU rate parameter adjustments, the total success rate across all 271 shots taken is 99.63% with false positives 0.37% of the time. The potential for greater accuracy and fewer false positives may be determined through a more thorough optimization analysis of the parameters on larger data sets.

B. REAL-TIME RESULTS

Real-time testing was conducted with the full system operating on a laptop. Two instances of MATLAB were running, one as the firearm node and one as the COC node providing data to Google Earth upon shot detection. Testing was conducted twice, first on a California-legal AR15 and once with the civilian MP15.

1. First Real-Time Test

The first real-time test revealed a system limitation, exposing an invalid assumption. This assumption was that the system's data rate was limited only by the YEI TSS-DL. In fact, the system is also limited by the computing speed of the machine used for data processing. Running the rifle and COC nodes on the same machine compounded this issue, causing the shot-identification algorithm to compete with Google Earth for system resources.

Laboratory development of the code and initial system testing occurred on a Dell XPS desktop running Windows 7 Enterprise on an Intel Core i7-4790 with 16 gigabytes of RAM. This computer did not have issues processing data in real time; thus, the real-time algorithm was designed at the highest speed at which the YEI TSS-DL could provide consistent time steps for the required data. This data rate was 833 Hz, or samples provided at every 1.2 ms.

Field experimentation during the first real-time test utilized a Pavilion dm4-3055 laptop running Windows 7 on an Intel Core i5-2430M processor with 8 gigabytes of RAM. The difference in computing speed between the laptop and the desktop used for development became evident during testing in the field on the California-legal AR15. Attempting to process the data in the 1.2 ms between readings outstripped the computing capacity of the laptop and caused dropped data samples. Sections of missing data, defined

as time steps greater than 1.6 ms, were as long as 923.2 ms and constituted 26.5% to 43.41% of the time in each firing sequence. An example of this lack of data corrupting the algorithm's ability to identify shots is seen in Figure 22.

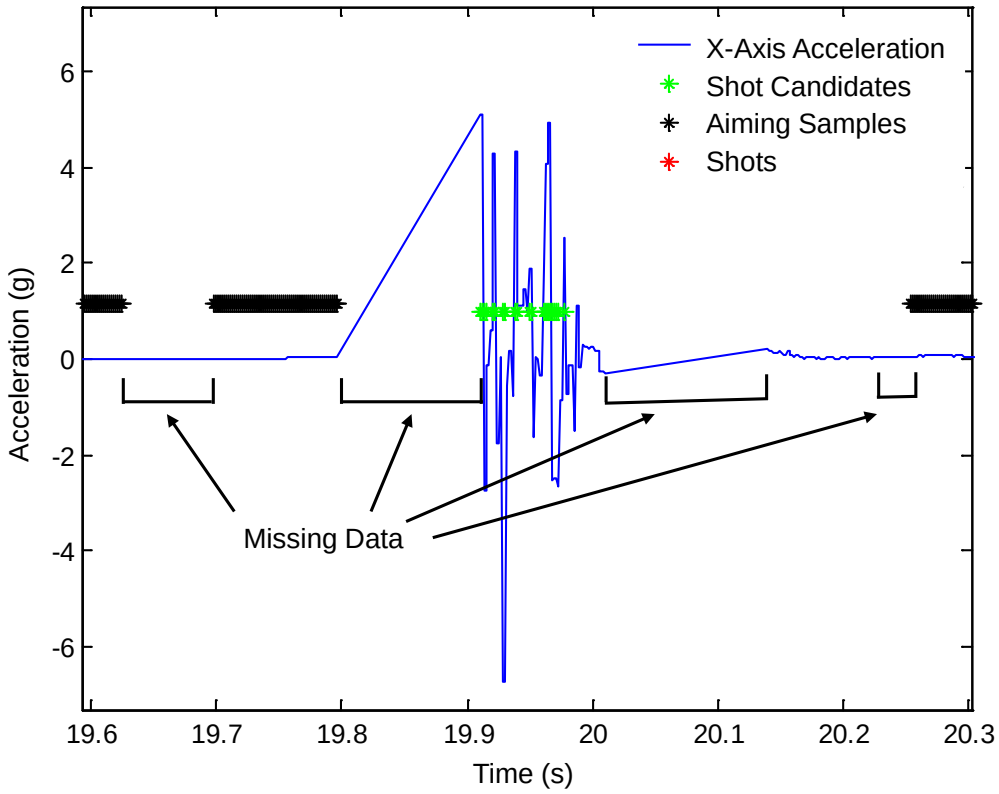

Figure 22. Missing Data Prevents Shot Identification

The 110.0 ms of missing data at the beginning of the shot profile prevents recognition of the hammer fall profile, and breaking the acceleration threshold does not occur within the recent-aiming window. Accordingly, the shot, which broke between 19.8 s and 19.9 s, is not identified. While the algorithm did work on shots where there were sufficient data collected, this issue of dropped data negated a quantitative assessment of the results necessitating a second real-time test.

2. Second Real-Time Test

The second real-time test was conducted with the civilian MP15 using one YEI TSS-DL for both shot identification and orientation. This YEI TSS-DL was set in Kalman

mode, providing data every 4.5 ms to ensure that the data processing machine could keep pace. A new HP Pavilion laptop was used throughout the two days of testing. This laptop used a next generation Intel Core i5-6200U processor on Windows 10 with 12 gigabytes of RAM and ran two instances of MATLAB and Google Earth. Google Earth was run both connected and disconnected from Wi-Fi. The real-time mapping was successful in both cases, validating Google Earth's caching capability. Additionally, three shooters were used during this round of testing.

After the experience of the first real-time test, this second round of testing focused on getting the full system running reliably. This required patience with the orientation functionality and minor changes to the shot-identification code. The data from the first several sequences of shooting contain bad data for orientation as the YEI TSS-DL kept providing orientations that were drifting. Several calibrations and power cycles later, reliable orientation in the direction of the target was attained.

Next, a change was required in the differential power parameter. The 4.5-ms data period was very similar to the 4.4-ms period used in data collection for the a priori algorithm. For this reason, the 87.95 W was deemed a reasonable starting threshold; however, being an additive value over a cycle, the differential power threshold typically is not reached until later in the reloading cycle. In the a priori code, the differential power threshold is assessed after the other parameters, which are found right as the shot breaks, at the beginning of a cycle. Evaluating the differential power parameter in a similar fashion in the real-time code requires changing the code to reference past indexed values of the other parameters vice evaluating them all at the same index point. Additionally, the shot would not be identified until later in the cycle. Instead, a lower value for the differential power threshold was set at 36.0 W. Further analysis is required to accurately place this value. The plot of the acceleration signal from a hammer pair identified with the real-time code with differential power and algorithm markings overlaid is shown in Figure 23.

Another issue in the code that was not identified until after shooting was completed was related to the hammer fall profile. The real-time code used during shooting checked for the absence of the aiming state two data points before the acceleration threshold was broken, along with changes between the shot breaking and the four previous data points,

due to the aim check length being five periods. The results as they were at the time of the test are contained in Table 11 of Appendix C. These results show five missed shots and two false positives resulting in 92.53% of shots correctly identified with 2.99% false positives.

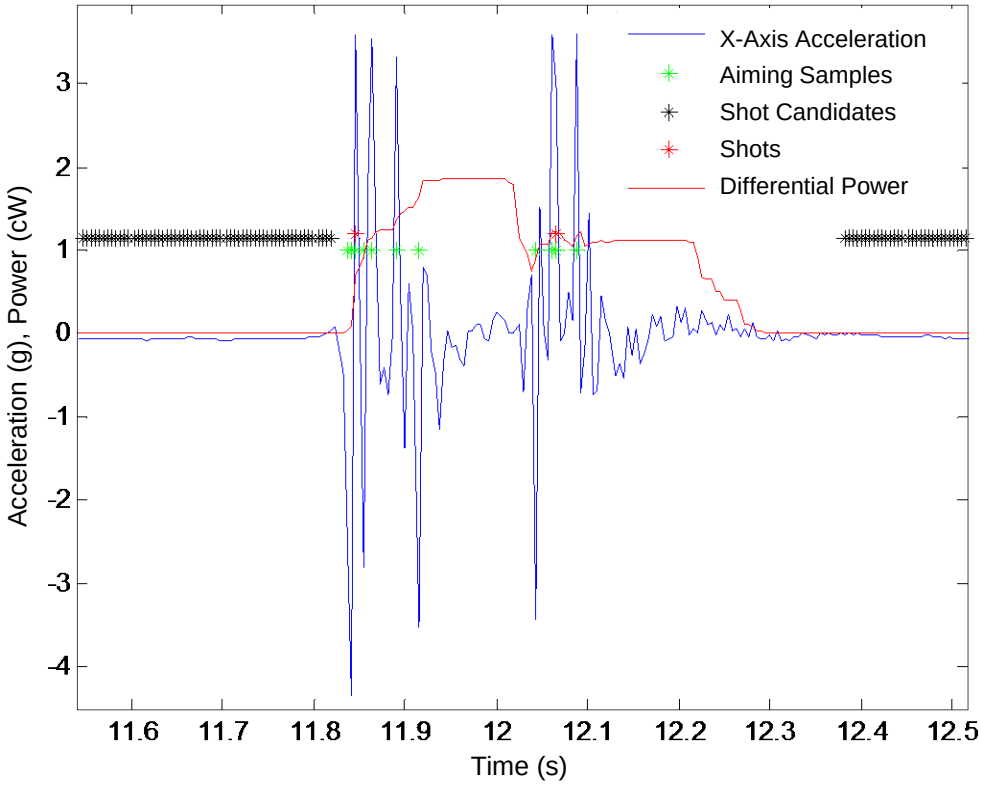

Figure 23. Real-Time Algorithm Identifies Hammer Pair with Differential Power Overlaid

Subsequent code was written, given in Appendix A, which allows re-enacting the real-time code from both the COC-node side and rifle-node side. This is possible as the workspaces from both instances of MATLAB were saved following each sequence of shooting. The re-enact code simply loops through the timestamped accelerations and orientation data one at a time, simulating receiving the data from the YEI TSS-DL. This code allows analysis and code improvement as if it were happening in real time. The results with the hammer profile corrected to check for the absence of the aiming state one data

point before the acceleration threshold is broken are contained in Table 12 of Appendix C. The much-improved result of zero missed shots and two false positives resulting in 100% of shots correctly identified with 2.99% false positives is shown in Table 12.

These two false positives occurred when sending the bolt back to the front of the rifle during initial loading. As seen in Table 13 of Appendix C, these false positives can be removed by increasing the peak count from three to four but at the expense of one missed shot in another sequence of shooting. This tradeoff relates to under sampling and attempting to identify a specific profile without full information about that signal. Ultimately, operating at these sampling rates requires larger data sets to accurately determine the optimum values for every parameter. The parameter values used in the second real-time test are found in Table 5. These values are similar to those used in the a priori algorithm with the difference being attributed to the slightly increased time step and its impact on indexing.

Table 5. Second Real-Time Test Parameter Values

Parameter	Value
aiming_threshold	0.115 g
accel_threshold	1.75 g
aim_length	94.5 ms
aim_check	22.5 ms
cycle_time	166.5 ms
rapid_time	306 ms
recent_aiming	49.5 ms

C. INTEGRATION WITH GPS, ORIENTATION, AND COC MAPPING

System integration was successful in that a proof-of-concept prototype based on COTS products was established. Two separate rifle nodes responding to real-time simulated shot inputs passed data via the NPS network to the COC node, which mapped these shots in Google Earth. Additionally, the second real-time test was successful in

identifying and plotting shot azimuths in real time by passing data between two separate instances of MATLAB.

The errors depicted on the Google Earth overlay impact the ability of the system to increase SA. Examples of this overlay are seen in Figure 18 and Figure 25, which is presented further on in a discussion regarding electromagnetic interference. The GPS error ring radius was heuristically determined to be 50.0 m; however, this was based on the BU353S4 receiver being located on the windowsill of the fifth deck of Spanagel Hall, facing west. Only a limited portion of the sky was visible to the receiver from that location. Additionally, the receiver was behind double-paned glass. Both reasons are potential suspects for the limited performance, as USGlobalSat Incorporated claims the BU-353S4 has an accuracy of less than 2.5 m [28]. Using the receiver during the second real-time test with clear views of the sky provided a positive, qualitative assessment of this claim; however, quantitative validation and improvement of GPS localization is not the focus of this thesis. The 50-m radius was kept as it represents a worst-case scenario. Concurrent work within the Reticle Program [27], [35] is focused on developing a personal inertial navigation system that can be used to refine localization for the rifle nodes. Additionally, transitioning the prototype system necessitates using military GPS with better accuracy. Further discussion of such improvements is found in the next chapter.

The qualitative accuracy of GPS and the foundational concept of using this system to resection the location of the enemy were evaluated during the second real-time test. Prior to that test, the GPS receiver was stationary and located inside the laboratory as explained above. Once the full system was running reliably during the second real-time test, the shooter walked laterally with respect to the target, firing to the east at a target located approximately 100.0 m away. Using the USGlobalSat Incorporated claim of 2.5-m radius and the five-degree azimuths errors, discussed in the next paragraph, we obtained the map overlay in Figure 24, with the target point represented by a yellow star. All of the lines and text of Figure 24 have been thickened from the original screenshot for clarity. It should be noted that this sequence of firing took place on the second day of testing without an updated calibration. All firing sequences from that day showed 15 to 18 degrees of firing azimuth offset that that was corrected in the generation of Figure 24. The yellow path drawn in

Google Earth links the outside intersections of the azimuth error lines showing the potential area where the target could be located. Even without the yellow path, the probable location of the target is immediately evident, facilitating the SA of any supporting agency viewing the overlay.

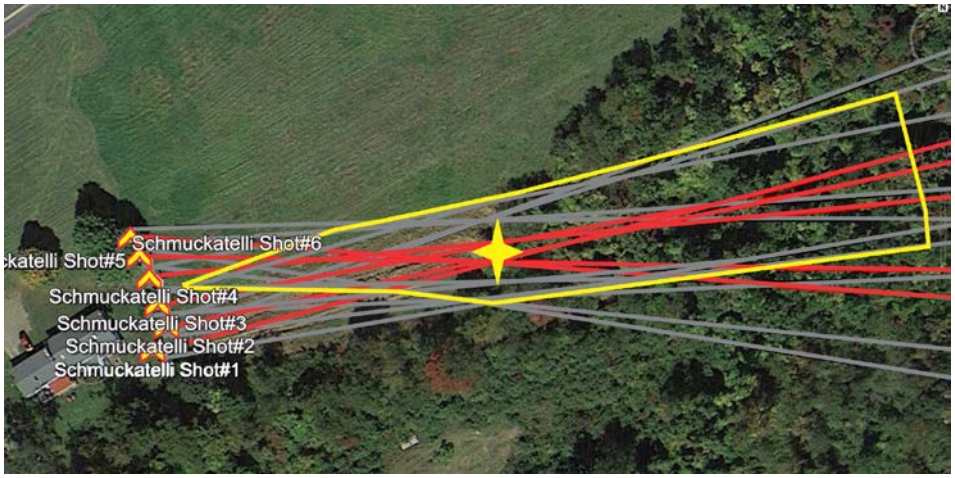

Figure 24. Map of Shots Fired While Walking Laterally to Target

The error azimuths on the overlay were set to five degrees. This is based on the previous work of Khan, who showed that the YEI TSS-DL sensor was capable of providing accuracy to within +/- 4.07 degrees under dynamic conditions, with an additional buffer added [8]. The dynamic error of the sensors was not re-validated, but steady state errors, after the correction factor derived from the declination station described in Appendix D, were repeatable within one to two degrees. This is only an approximation, as we did not undertake the rigorous effort to truly establish ground truth, which Khan attained via the VICON system [8]. Instead, the method of establishing accuracy was to compare the plotted firing azimuths to the desired target points of the declination station.

Additionally, the above azimuth error does not take into account any time-varying, local magnetic fields that can have a substantial impact on the accuracy of the firing azimuth. Again, the constant magnetic field errors were removed by calibration and, in the laboratory testing, with a correction factor derived using a declination station as described in Appendix D. The effect of time-varying fields was demonstrated by taking a simulated

shot with no electromagnetic interference and a second simulated shot with a cell phone immediately adjacent to the sensor. The screenshot contained in Figure 25 illustrates the 6.8-degree difference in the red firing azimuth caused by the electromagnetic interference created by the cell phone. Once again, the line widths have been increased for clarity.

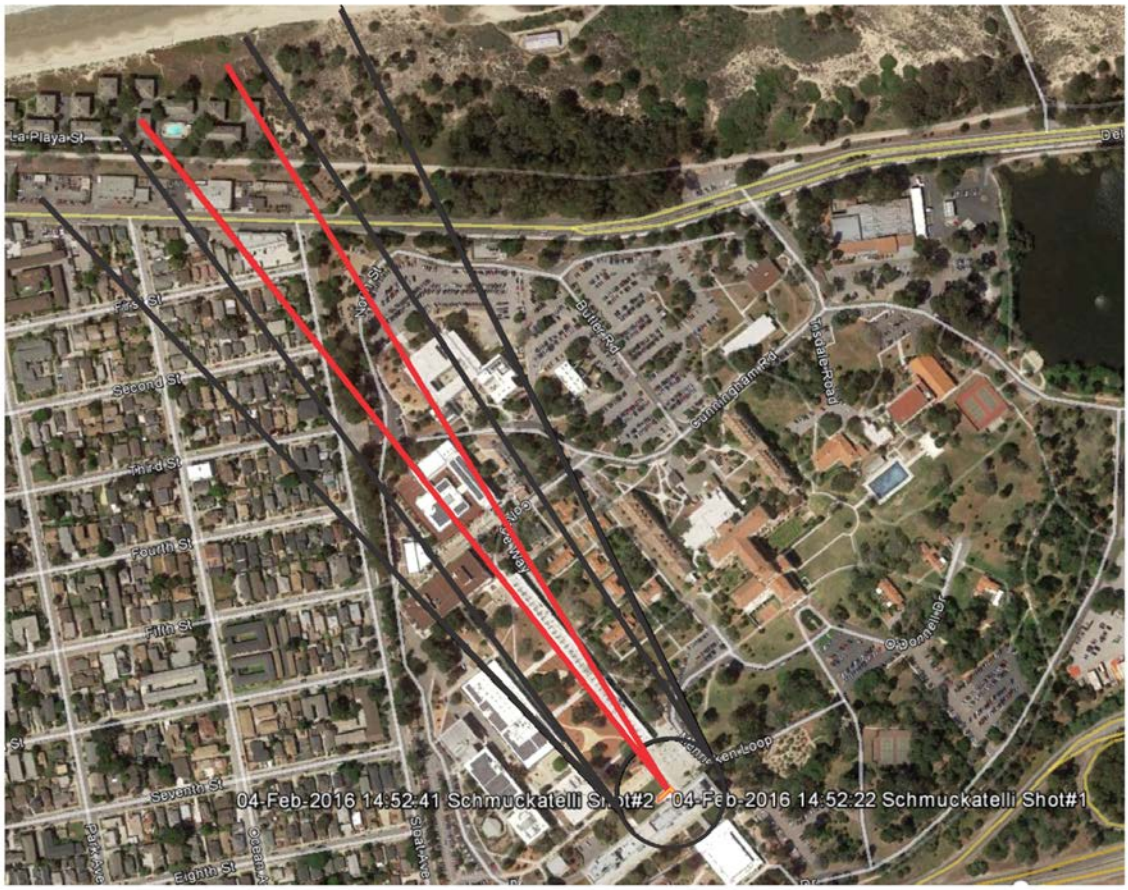

Figure 25. Electromagnetic Interference from a Cell Phone Causes the YEI TSS-DL to Return a 6.8-Degree Firing Azimuth Error

One final issue with regard to orientation estimates is the potential for lag when moving the weapon quickly. Khan recognized this in [8] but was unable to test whether significant lag occurred while manipulating the weapon at realistic speeds attained during target engagement. This was evaluated during the second real-time test. The target was placed at approximately 70.0 m. The shooter started out facing south and pivoted to the east, bringing the rifle to his shoulder and engaging the target; thus, the rifle traversed

approximately 60–90 degrees of azimuth before accurately engaging the target. This action was executed eight times. All eight shots were identified and all were on the appropriate azimuth. Using Google Earth to measure the distance from the outside azimuths at the 50-m ring yielded a distance of 4.38 m. Using the chord length formula with this distance yields an angular distance of 5.02 degrees. This falls on the low side of the four other sequences with reliable orientation data. These sequences' shots were fired without ever moving the rifle off its aim point and have angular differences ranging from 1.06 degrees to 13.16 degrees; thus, orientation lag is not an issue at the speeds necessary to engage a man-sized target at 70.0 m.

Any azimuth error, in conjunction with GPS error, has a significant impact on the applicability of the proposed system to the increase of SA. At the maximum effective range of the M16 of 800.0 m, +/- five degrees of error turns into a straight-line error of 139.45 m, based on the chord length formula. The addition of the 100.0-m GPS error results in a 239.45-m error. While this level of accuracy is clearly not sufficient for targeting, even the general layout of the situation on the battlefield is highly beneficial to the unit leader. This allows him to rapidly orient on the general vicinity of the enemy, decreases the amount of time to orient aircraft during a talk on, and can be used to double check any targeting coordinates prior to sending a call for fire. Additionally, as more members of the unit engage the enemy from different locations, the ability to resection the enemy's location becomes possible, as described previously.

D. EMBEDDED SYSTEM TIMING ISSUES

The Raspberry Pi real-time algorithm was not tested on a firearm. Preliminary testing in the laboratory suggests that the microcontroller is too slow for the data processing requirements for the shot-identification algorithm. While a simulated shot on a modified algorithm produces the desired output, the amount of time for data processing between samples of the full algorithm is approximately 14.0 ms. This data rate is without a shot being fired, which requires execution of more code and is expected to take even longer. This data rate is not sufficient to detect even slow acceleration events such as the hammer

fall. Furthermore, the first round of real-time testing suggests that significant periods of dropped data are to be expected with a Raspberry Pi-based rifle node.

V. CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK

A. CONCLUSIONS

The successful identification of AR15 acceleration shot profiles with relatively inexpensive COTS AHRSs was demonstrated in this thesis. While the COTS AHRS operated at a sub-Nyquist data rate, it is not possible to generalize the methodology used to combat the effects of sub-Nyquist sampling used in this project. This strategy only worked due to the uniqueness of shooting dynamics, most notably, an explosive event preceded by the required stillness for aiming. With this limitation acknowledged, the 99.63% identification in post-processing confirmed the methodology for AR15 rifles. This shot-identification capability was successfully integrated in real time with an orientation capability and a GPS receiver on a rifle node. Two rifle nodes were networked to a COC node, allowing the data to be mapped in Google Earth. This type of system would help increase SA of small-unit leaders and supporting agencies, and its success encourages the search for further untapped data sources to increase combat effectiveness.

In terms of contributions, tools were developed to interact with the YEI TSS-DL in real time in the MATLAB environment and, on a broader level, an application of electronic sensors for weapons' data tracking was examined. The former is novel and opens up numerous opportunities to transition motion tracking research from post processing to real-time applications in a familiar programming environment. The latter contribution is politically sensitive, resulting in a paucity of academic research. The only similar work discovered was that of Loeffler in [24], who post-processed IMU data to identify shots from a handgun. His 98.9% successful identification rate matches closely the 99.63% rate of the a priori algorithm developed in this work. Loeffler's overarching concept of using IMU data for shot identification was extended in this thesis to AR15-variant weapons and to a real-time application.

Outside academia, smart guns are recently in the news [25], but only Yardarm Technologies is pursing weapons-orientation tracking [26]. Yardarm is also the only company pursuing IMU data to identify shots but is not publishing any research, likely for

market-related reasons. Most other research into weapon dynamics during firing is focused on recoil modeling and reduction [12], [14] and not for inclusion in a real-time system; thus, this and other Reticle Project work is unique.

B. FUTURE WORK

There are numerous avenues of future work stemming from this particular Reticle Project thesis that fall under the categories of system improvement and additional applications.

1. System Improvement

The shot-identification algorithm needs further testing, ideally with infantrymen conducting live-fire exercises. This would help refine the parameter values used in the algorithm and may reveal other approaches for solving the problem. This testing would also allow feedback from small-unit leaders on the type and style of information presented in Google Earth.

The widespread testing described will only be feasible once the prototype system has improved to the point where the rifle-node equipment is man-portable on an embedded system. The most significant step in achieving that goal will be the development of an embedded system with the power to run the algorithm or a means of streamlining the algorithm to run efficiently on an embedded system. One potential solution is to integrate and test the YEI TSS-DL with the smartphone being fielded for the NW system. This approach would have the added benefit of the already-developed power system of NW as powering of Reticle Project systems has yet to be studied. Additionally, some work on transitioning to wireless YEI TSS sensors was conducted and should be continued. Wires running from a backpack or butt of a rifle to the upper receiver is impractical. Newer sensors should be investigated with particular attention paid to the output data rate. The more resolution obtained on the shot acceleration profile, the greater the accuracy of the shot-identification algorithm. This benefit must be weighed against the ability of the processor to keep pace with computations.

Other parts of the prototype system also need further development. A significant concern, discussed in Chapters III and IV, is the accuracy of orientation estimates. A means to correct local magnetic field distortions or a completely new method of obtaining orientation is required to minimize azimuth errors. Additionally, only the yaw angle of orientation was dealt with in this thesis, as the data provided is displayed on a two-dimensional surface. In urban environments, threats are located in all three dimensions. Taking the pitch of the rifle into account potentially allows commanders to estimate which floor of a building friendly forces are engaging.

More research into the localization capability of the system is also required. Utilization of military GPS will lower the location error to acceptable levels wherever a GPS signal is present; however, operating in, and the transition to, GPS-denied environments is a large area of study that has already received some attention within the Reticle Project [27], [35]. Integrating a personal navigation system, as in [27] and [35], can increase outdoor location accuracy if fused with GPS data and allow for indoor tracking once the GPS signal is degraded beyond a certain point.

Finally, there is much work to be done on integrating all previous facets of the Reticle Project. Previous applications have dealt with rifle orientation [8], dismounted navigation [27], and body posture detection [35], all of which have less demanding sampling rate requirements than those necessary to obtain full information of rifle accelerations during firing. This timing issue presents a serious challenge. There is also the need to tie all of these applications into the Robotic Operating System based Pub/Sub network of [9]. Overall, there are numerous opportunities of study for improving this prototype system.

2. Additional Applications

There are several applications of networked infantrymen directly related to IMU weapons data. The algorithm discussed in this thesis already counts the numbers of rounds fired per rifle node. The overall system can be used to track the number of rounds expended, giving the small-unit leader concrete knowledge of his unit’s ammunition status. At the individual level, this data can be used to indicate a required magazine change. IMU

weapons data can be used to identify the number of magazine changes, an additional measure of the ammunition status of the unit. Another potential IMU data application relates to the security of the system. The system contains locations of all friendly combatants, making it a target for cyber intrusion. The IMU data can potentially be used as a means for kinematic authentication, where each rifleman uses unique, individually specified movements of the rifle as a password to unlock the system. These are just a few ideas for future IMU data applications.

APPENDIX A. CODE

A. A PRIORI CODE

```
%%Final_A_priori.m     By Captain Kiel Reese
%A priori algorithm for shot identification. This code takes
%accelerometer data from an AR15 style weapon (M16A2 or M4) and
%analyzes for shots fired. Original data collected with YEI 3-Space
%Sensor, sampling every 4.4ms~227 Hz. Data files consist of microsecond
%time stamps followed by all three axes acceleration, with the header
%removed. Data collected at differing frequencies will have to adjust
%parameters accordingly. Output provided is the sample rate of the
%data, number of shots, and four plots consisting of the acceleration
%data with algorithm overlays, all the shot profiles overlaid on one
%another, the differential power vs time, and the number of peaks
%involved in each shot.

%%%%%%%**INITIALIZE DATA**%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clear all
clc

data1 = load('quickreactdoubletap_10yrd.txt'); %ensure text file has
                                               %had its header removed
                                               %and is in the directory

n= length(data1);                              %for indexing purposes
count = linspace(1,max(size(data1)),max(size(data1)));%for plotting
                                                      %purposes

time = data1(:,1);          %time stamps in microseconds
time = time-time(1);        %sets start time to 0 seconds
time_sec = time/1000000;    %puts microseconds to seconds

%%Calculate time step and sample rate
delta_t=zeros(1,length(time_sec));        %initialize delta_t array

for xx = 2:length(time_sec)
    delta_t(xx-1) = time_sec(xx)-time_sec(xx-1);
end

delta_t(length(delta_t)) = delta_t(xx-1); %puts the second to last
                                          %value in for the
                                          %uncalculated last value
time_step = mean(delta_t);    %display avg time step
sample_rate =1/time_step      %display sample rate

%Acceleration data in g
x_gs = data1(:,2);        %along x axis
y_gs = data1(:,3);        %along y axis
z_gs = data1(:,4);        %along z axis
```

```
%%%%%%%%**END INITIALIZATION**%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%**AIMING:  Filtering to determine if aiming is occurring**%%%%%%%

aiming_threshold =.115;  %in g, +/- this value in g is the threshold
                         %deemed to be aiming (envelope for aiming)
aim_length = 21;         %in counts (21~92ms).
aim_check = 5;           %how many counts previous to check that aiming
                         %is within threshold

%Compare every sample with the 5th sample behind it, and ensure they
%are within the aiming envelope

aiming = ones(n,1); %initialize aiming array
for aa = aim_check+1:n
    if abs(x_gs(aa)-x_gs(aa-aim_check))<aiming_threshold
        aiming(aa) = 1;
    else
        aiming(aa) = 0;
    end
end

%Count the number of consecutive points assessed as aiming
aim_count = zeros(n,1);%initialize aim_count array

for bb = 1:n
    if aiming(bb)==0;
        aim_count(bb) = 0;
    else
        if bb == 1  %leave first aim_count at 0. Required due to
                    %indexing.
            aim_count(bb) = 0;
        else
            aim_count(bb) = aim_count(bb-1) + aiming(bb);
        end
    end
end

%Filter out times when the firearm is not aiming for a certain amount
%of time(aim_length). This gets rid of peaks that happen to fall within
%the threshold.

for cc = 1:n
    if aim_count(cc)<aim_length
        aiming(cc) = 0;
    else
        aiming(cc) = 1;
    end
end

%%%%%%%**END OF AIMING**%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%**BEGIN SHOT FINDING: Iterative process involving clearing out
%one cycle and then checking for rapid fire follow up shots...assumes
%only 3 successive rapid fire shots maximum**%%%%%%%%%%%%%%%%%%%%%%%%%%%%

cycle_time = 39;        %value in counts (4.4ms per count) for cycling
                        %of weapon (39~168ms)
accel_threshold = 1.75; %in g, to indicate a possible shot
rapid_time = 70;        %time, in counts, that rapid fire could occur
                        %without getting a second sight picture.
                        %(70~308ms) Measured from shot breaking.
recent_aiming = 11;     %time, in counts, before the shot breaks that
                        %aiming was occurring (11~48.4ms)

%initialize tracking arrays.
shot_candidates = zeros(1,length(x_gs));
shot_aiming =  zeros(1,length(x_gs));
shot_rapid = zeros(1,length(x_gs));

%Go through x_gs and look for accelerations over the threshold. This
%threshold is an absolute value.

for dd = 1:length(x_gs)
    if abs(x_gs(dd))> accel_threshold
        shot_candidates(dd) = 1;
    end
end

%Look for shot candidates that occur within recent_aiming time frame.
%This time takes into consideration the hammer falling and potential of
%undersampling to cause missed peaks right at the shot break.

for ee = 1:length(shot_candidates)
    if shot_candidates(ee)==1 && aiming(ee-recent_aiming) == 1
        shot_aiming(ee) = 1;
    end
end

%Go through aiming shots and get rid of any peaks (possible shots)
%within one cycle time.

for ff = 1:length(shot_aiming)
    if shot_aiming(ff) == 1
        shot_aiming(ff+1:ff+cycle_time) = 0;
    end
end

%Check for hammer fall. Either the last sample before the shot breaks
%is not considered aiming or there is a drastic change in acceleration
%between the last three samples before the shot break. This second
%requirement is needed since aiming checks 5 sample previous.

for ee = 1:length(shot_aiming)
```

```
    if shot_aiming(ee) == 1 && (aiming(ee-1) == 0 || (abs(x_gs(ee-1)-…
       x_gs(ee-2)) > aiming_threshold || abs(x_gs(ee-1)-x_gs(ee-3)) >…
       aiming_threshold))

        shot_aiming(ee) = 1;
    else
        shot_aiming(ee) =0;
    end
end

%Go through aiming_shots again and get rid of any peaks(possible shots)
%within one cycle time

for ff = 1:length(shot_aiming)
    if shot_aiming(ff) == 1
        shot_aiming(ff+1:ff+cycle_time) = 0;
    end
end

%Go through INITIAL shot candidates and see if any are from within
%rapid_time of an aimed shot, but not within one cycle time. Assumption
%is that if you go ~130ms after a shot, you will get another sight
%picture.

for gg = 1:length(shot_candidates)
    if shot_candidates(gg) == 1 && sum(shot_aiming(gg-rapid_time:gg…
    -cycle_time)) == 1

        shot_rapid(gg) = 1;
    end
end

%Clear one cycle time in front of rapid shots

for hh = 1:length(shot_rapid)
    if shot_rapid(hh) == 1
        shot_rapid(hh+1:hh+cycle_time) = 0;
    end
end

%Check original candidates for a second rapid shot. Assumes no more
%than two shots fired after initial shot without regaining sight
%picture

for ii = 1:length(shot_rapid)
    if shot_candidates(ii) == 1 && sum(shot_rapid(ii-rapid_time:ii-
cycle_time)) == 1
        shot_rapid(ii) = 1;
    end
end

%Clear one cycle time in front of second rapid shots
for jj = 1:length(shot_aiming)
    if shot_rapid(jj) == 1
```

```
        shot_rapid(jj+1:jj+cycle_time) = 0;
    end
end

%%%*****Total shot count, both aimed and rapid follow up shots****%%%%
shot_count = shot_aiming + shot_rapid;

%Check against dry fire by ensuring that there is more than three peak
%accelerations during one cycle time. If there was a dry fire, there
%would only be 1–2 significant peaks.

peak_count = zeros(1,length(shot_count));
for kk =1:length(shot_count)
    if shot_count(kk) == 1
        peak_count(kk)=sum(shot_candidates(kk+1:kk+cycle_time));
    end
end

%In a similar vein to the peak counter, this section of code
%calculates the differential power of the signal over one cycle time,
%which gives an indication of how much change in acceleration is
%occurring during that cycle time.

%Develop the difference vector that is each acceleration minus the
%previous acceleration

for ii = 2:length(x_gs)
    difference(ii-1) = x_gs(ii)-x_gs(ii-1);
end

difference = abs(difference); %make the differences all positive
window_time = cycle_time; %decreasing window time may give better
                          %resolution on shot characteristics
window = difference(1:window_time); %initial window array is made up of
                                    %difference
power_record = zeros(1,length(x_gs)); %Preallocate power_record array

%Start at the data point after the length of window time and cycle
%through the difference array. Find the window of difference values,
%add them up and square for power.

for ii = window_time+1:length(difference)
    window(window_time+1) = difference(ii);%add the current difference
                                           %value to the end of the
                                           %window
    window = window(2:window_time+1);%delete the first difference value
                                     %in the window to obtain the new
                                     %window
    window_power = sum(window.^2); %sum the squared difference values
                                   %in the window
    power_record(ii) = window_power; %record this power value
                                     %indicative of the change in
                                     %signal over the previous cycle
end
```

```
%Implement the peak counter and window power logic
for ii = 1:length(shot_count)
    power_condition = [];
      %At each shot count, check the power value of the second half of
      %the cycle time against a threshold (heuristically determined)

    if shot_count(ii) == 1
        power_condition = find(power_record(ii+20:ii+cycle_time…
         -1)>87.95);
        if isempty(power_condition) && peak_count(ii) < 3 %no powers
                                    %exceeded the threshold and the peak
                                    %count was too small
            shot_count(ii) = 0;
        else
            shot_count(ii) = 1; %power level and peak count thresholds
                                %were met
        end
    end
end

%Display Number of events assessed as shots
shot_counter =sum(shot_count)

%%%%%**END OF SHOT FINDING**%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%**BEGIN PLOTTING**%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%Record each individual shot for pattern recognition
column_step = 0;
for kk =1:length(x_gs)
    if shot_count(kk) == 1
        column_step = column_step +1;
        %Grab 6 steps before acceleration threshold is crossed
        x_gs_record(:,column_step) = x_gs(kk-6:kk+cycle_time,:);
        time_record(:,column_step) = time_sec(kk-6:kk+cycle_time,:);
        %Pull time step back to zero
        time_record(:,column_step) = time_record(:,1)-time_record(1,1);
    end
end

%Plot X axis accelerations
figure(1)
plot(time_sec, x_gs)
hold on

%Plot aiming times and shot times over acceleration plot
aim_times = time_sec.*aiming;%gets rid of non aiming points
plot(aim_times,aiming,'k*') %plots aiming times, will leave an
                            %aim mark at 0,0
shot_count_trans = transpose(shot_count);
shot_count_time = shot_count_trans.*time_sec;
plot(transpose(shot_candidates).*time_sec, shot_candidates,'g*')
```

```
plot(shot_count_time, shot_count,'r*')%plots shot times, will leave a
                                      %shot mark at 0,0
ylabel('Acceleration (g)')
xlabel('Time (s)')
title('Accelerations vs Time')
legend('X Axis Acceleration', 'Aiming Samples', 'Shot Candidates',…
'Shots')

%plots all individual shots on top of one another in a separate figure
for ll = 1:shot_counter
    figure(2)
    plot(time_record(:,ll),x_gs_record(:,ll))
    hold on
    title('Individual Shots')
    xlabel('Time (s)')
    ylabel('Acceleration (g)')
end

%Plot the differential power vs time
figure(3)
plot(time_sec', power_record,'r')
title('Differential Power vs Time')
xlabel('Time (s)')
ylabel('Differential Power (W)')

%Plot the total peak count of shots
peak_count_final = peak_count + 1; %This includes the first peak that
                                   %breaks the threshold
figure(4)
plot(time_sec,peak_count_final,'-*')
grid on
title('Number of Peaks in Each Shot vs Time')
xlabel('Time(s)')
ylabel('Number of Peaks')

%%%%%%%%**END PLOTTING/END ALL**%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

B. REAL-TIME MATLAB CODE

1. Rifle Node

```
%%Rifle_node_one_TSS_WITH_diffpower_Fully_Commented  By Capt Kiel Reese
%This script runs the Rifle Node and records data every 15000 data
%points for analytic purposes. It connects to the COC via TCPIP and
%sets up one serial YEI sensor to run the shot finding algorithm and
%orientation finding. It sets up a GPS sensor on another serial port,
%and then runs indefinitely. Assumes all first shots will be aimed but
%will find up to two additional rapid, follow-up shots without
%obtaining another sight picture(taken without aiming within ~130ms of
%aimed shot or first rapid shot). Will still require another Matlab
%instance open to act as the COC for mapping purposes. Designed for avg
%of 4.5ms between readings of the YEI sensor (in Kalman Mode).
```

```
close all
clear all
clc

%% Set up Comm to COC
%set up tcpip to send lat,lon, and firing az to COC
%computer 172.20.65.120 (computer across the lab) or 'localhost'(Matlab
%instance running on same computer) to act as COC

%set to 'localhost' if using Matlab instance on same computer
t2 = tcpip('localhost', 10000, 'NetworkRole', 'client');
t2.TimeOut = 1; %based on GPS timing
fopen(t2);
disp('Connected to COC')

%% Setup YEI Sensor Conducting Shot Finding
%define required commands for the YEI Sensor
TSS_START_BYTE = uint8(247);
TSS_RESPONSE_HEADER_START_BYTE = uint8(249);
TSS_START_STREAMING = uint8(85);
TSS_STOP_STREAMING = uint8(86);
TSS_TARE_CURRENT_ORIENTATION = uint8(96);
TSS_GET_UNTARED_EULERS = uint8(7);
TSS_NULL = uint8(255);

%define serial comm through specified COM port of YEI sensor at a
%specified baudrate(115200)
s1 = serial('COM4','BaudRate', 115200,'ByteOrder','bigEndian');
fopen(s1);   %opens the COM port

%seperate m file that tells device to get corrected accelerations and
%Euler angles. Also sets streaming timing, if the sensor is told to
%stream.
setupstreaming_header_Accel_and_taredEuler

%With parameterless wired commands the command byte will be the same
%as the checksum
write_bytes = [];
write_bytes = [write_bytes, TSS_RESPONSE_HEADER_START_BYTE,
TSS_START_STREAMING, TSS_START_STREAMING]; %tells the sensor to stream

fwrite(s1, write_bytes,'uint8');
disp('TSS_SHOT_FINDING_START_STREAMING')

%% Setup YEI Sensor Finding Orientation-for use in two sensor
%configuration
% %Define command to read orientation from orientation YEI sensor
% write_bytes = [];
% write_bytes = [write_bytes, TSS_START_BYTE];
% write_bytes = [write_bytes, TSS_GET_UNTARED_EULERS]; %this is set up
                %to have the sensor return one instance of orientation,
                %will be used %immediately upon shot detection
% write_bytes_check_sum = write_bytes(2:length(write_bytes));
% check_sum = uint8(mod((sum(write_bytes_check_sum)), 256)); %puts
```

```
                                                    %checksum into bits
% write_bytes = [write_bytes, check_sum]; %add checksum byte

% %define serial comm through specified COM port at a specified
  %baudrate
% s2 = serial('COM10','BaudRate', 115200,'ByteOrder','bigEndian');
% fopen(s2);   %opens the COM port
% disp('TSS_ORIENTATION_FINDING_OPEN')

%% Setup GPS Serial
s3 = serial('COM3','BaudRate', 4800);

disp('GPS Set')
pause(1)

%% Variables and Tracking Arrays for Shot Finding
sample_count = 1;
n = 15001; %is greater than the 15000; will limit the amount of data
aiming_threshold =.115;
aiming_difference = zeros(1,n-2);
aiming_initial = zeros(1,n-2);
aiming = zeros(1,n-2);
shot_aiming = zeros(1,n-2);
summer = zeros(1,n-2);
shot = zeros(1,n-2);
shot_candidates = zeros(1,n-2);
peaks = zeros(1,n-2);
difference = zeros(1,n-2);
window_power = zeros(1,n-2);
pass_shot = 0;
reset_count = 1;

%Variables for implementation of Rapid Fire
aimed_shot = zeros(1,n-2);
rapid_shot1 = zeros(1,n-2);
rapid_shot2 = zeros(1,n-2);

%% Variables for GPS
GM_Monterey = 13.38; %GM is East. Will not be used if sensor tared
                     %at true north
shot_count = 0;
pass_to_COC=[];

%% Indefinite Loop Until Ended with Cntrl C in the Workspace

while sample_count < n %sample count gets reset before getting kicked
                       %out of the loop

    %first if statement limits data size and keeps index greater than a
    %rapid time. Appends extra rows to the tracking arrays for each
    %reset.
    if sample_count>= 15000
        disp('Reset')
        reset_count = reset_count + 1;
```

```
        sample_count = 69;
        aiming_difference = [aiming_difference;zeros(1,n-2)];
        aiming_initial = [aiming_initial; zeros(1,n-2)];
        aiming = [aiming;zeros(1,n-2)];
        shot_aiming = [shot_aiming; zeros(1,n-2)];
        summer = [summer; zeros(1,n-2)];
        shot = [shot; zeros(1,n-2)];
        shot_candidates = [shot_candidates; zeros(1,n-2)];
        peaks = [peaks; zeros(1,n-2)];
        difference = [difference; zeros(1,n-2)];
        window_power = [window_power; zeros(1,n-2)];
        aimed_shot = [aimed_shot; zeros(1,n-2)];
        rapid_shot1 = [rapid_shot1; zeros(1,n-2)];
        rapid_shot2 = [rapid_shot2; zeros(1,n-2)];
    end

%% Read the bytes returned from the serial of Shot Finding YEI
    if sample_count == 1 %required to throw out extraneous header data
                         %received from instruction set
        disp('STARRRRRT')
        data_str = fread(s1,2,'uint32');
        data_str1 = fread(s1,3,'single');
        data_str2 = fread(s1,3,'single');
    else %now that extraneous header data on first return from sensor
         %is gone, actually obtain timestamp, accelerometer readings,
         %and orientation data
        data_str = fread(s1,1,'uint32'); %reads the time stamp,
        data_str1 = fread(s1,3,'single'); %3 axis accelerometer
                                          %readings
        data_str2 = fread(s1,3,'single'); %3 Euler Angles

        %obtain each set of data
        x_accel(reset_count,sample_count) = data_str1(1); %Check sensor
                         %with TSS Sensor Suite to ensure correct axes.
        %Pulls Yaw angle
        orientation_data(reset_count, sample_count) = data_str2(2);

        timestamp(reset_count,sample_count) = data_str; %saving time
                                          %stamp for plotting purposes.

 %% Build Aiming Criteria

        if sample_count<68 %this if statement required because all
%thresholds are based off of previous data. Therefore, the farthest
%reach-back necessary (of the first iteration) needs to be taken into
%account.
            aiming_difference(reset_count, sample_count) = 1;
            aiming_initial(reset_count,sample_count) = 1;
        else
            aiming_difference(reset_count, sample_count)=…
x_accel(reset_count, sample_count)-x_accel(reset_count, sample_count-
5); %check against a previous sample (can be adjusted, originally was
~20ms)
```

```
            %check for aiming difference staying within the threshold
            %for aiming
            if abs(aiming_difference(reset_count, sample_count)) >…
                                                        aiming_threshold
                aiming_initial(reset_count, sample_count) = 0;
            else
                aiming_initial(reset_count, sample_count) = 1;
            end

            %if aiming for ~90ms consecutively, consider the rifle as
            %aiming
            summer(reset_count, sample_count) =…
        sum(aiming_initial(reset_count, sample_count-21:sample_count));
            if summer(reset_count, sample_count) == 22
                aiming(reset_count, sample_count)= 1;
            else
                aiming(reset_count, sample_count) = 0;
            end

            %find when accelerations break the shot threshold. More
            %sensitive to shots along gravity vector (shooting up or
            $down)
            if abs(x_accel(reset_count, sample_count)) > 1.75
                peaks(reset_count, sample_count) = 1;
            end

            %check for breaking the shot threshold when recently aiming
            %(~45ms). This is only “recently” aiming because the
            %acceleration from the hammer fall will break the aiming
            %threshold acceleration immediately prior to the first peak
            %acceleration
            if peaks(reset_count, sample_count) == 1 &&…
                                aiming(reset_count, sample_count-11)==1
                shot_aiming(reset_count, sample_count) = 1;
            end

            %look for hammer fall, last point before peak was not
            %classified as aiming or there was a significant
            %acceleration change within the last 4 samples (to recent
            $to affect aiming classification)
            if shot_aiming(reset_count, sample_count) == 1 &&…
sum(shot_aiming(reset_count, sample_count-37:sample_count)) == 1 &&…
(aiming(reset_count, sample_count-1) == 0 || (abs(x_accel(reset_count,…
sample_count-3)-x_accel(reset_count, sample_count-4)) > …
aiming_threshold || abs(x_accel(reset_count, sample_count-2)…
-x_accel(reset_count, sample_count-4)) > aiming_threshold) ||…
abs(x_accel(reset_count, sample_count-1)-x_accel(reset_count,…
sample_count-4)) > aiming_threshold))
                shot_candidates(reset_count, sample_count) = 1;
            end

            %build differential power criteria
```

```
            difference(reset_count, sample_count) =…
abs(x_accel(reset_count, sample_count)-x_accel(reset_count,…
sample_count-1));
            window_power(reset_count, sample_count)=…
sum(difference(reset_count, sample_count-37:sample_count).^2);

            %looks for at least 3 peak accelerations within
            %approximately the first half of a cycle time with a recent
            %hammer fall, with recent aiming. Gets rid of misfires
            %where hammer falls, but there's only one or two peaks of
            %acceleration
            if peaks(reset_count, sample_count) ==1 &&…
sum(shot_candidates(reset_count, sample_count-37:sample_count)) == 1 …
&& sum(peaks(reset_count, sample_count-15:sample_count)) >= 3…
 &&  window_power(reset_count, sample_count)>36…
 && sum(aimed_shot(reset_count, sample_count-37:sample_count)) == 0
                aimed_shot(reset_count, sample_count) = 1;
            end

            %look for first rapid shot
            if peaks(reset_count, sample_count) == 1 &&…
sum(peaks(reset_count, sample_count-12:sample_count))>=3 &&…
sum(aimed_shot(reset_count, sample_count-67:sample_count))==1 &&…
sum(aimed_shot(reset_count, sample_count-37:sample_count)) == 0 &&…
sum(rapid_shot1(reset_count, sample_count-37:sample_count)) == 0 &&…
window_power(reset_count, sample_count)>36
                rapid_shot1(reset_count, sample_count) = 1;
            end

            %look for second rapid shot
            if peaks(reset_count, sample_count) == 1 &&…
sum(peaks(reset_count, sample_count-12:sample_count))>=3 &&…
sum(rapid_shot1(reset_count, sample_count-67:sample_count))==1 &&…
sum(rapid_shot1(reset_count, sample_count-37:sample_count)) == 0 &&…
sum(rapid_shot2(reset_count, sample_count-37:sample_count)) == 0 &&…
window_power(reset_count, sample_count)>36
                rapid_shot2(reset_count, sample_count) = 1;
            end

            %report any shots
            if aimed_shot(reset_count, sample_count) ==1 ||…
rapid_shot1(reset_count, sample_count) ==1 ||…
rapid_shot2(reset_count, sample_count) == 1
                shot(reset_count, sample_count) = 1;

%% From here down, shot actually occurred
                clc
                pass_shot = pass_shot+1
                shot_count = shot_count+1;
                %use orientation data from last aiming data point to
                %calulate firing azimuth. Cycles back through last 12
                %samples. This number could be increased to match
                %recent aiming value
                last_aiming_index = sample_count;
```

```
                while last_aiming_index > sample_count - 12
                    last_aiming_index = last_aiming_index -1;
                    if aiming(reset_count, last_aiming_index) == 1
                        break
                    end
                end

                %obtain yaw at the last aiming index
                yaw_angle = orientation_data(reset_count,…
                                                    last_aiming_index);
                yaw_deg = yaw_angle*180/pi %convert to degrees

                orient_off_north = yaw_deg;

                if orient_off_north <= 0 %east is negative
                    orient_off_north = orient_off_north*-1
                                                            %+GM_Monterey;
                else %positive value is west
                    orient_off_north = 360-orient_off_north
                                                            %+GM_Monterey;
                end

                firing_azimuth = orient_off_north

%% Get GPS Coordinates from GPS to Pass to COC
%-------------------------------------------------------------------%
$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47
%
% Where:
%      GGA          Global Positioning System Fix Data
%      123519       Fix taken at 12:35:19 UTC
%      4807.038,N   Latitude 48 deg 07.038' N
%      01131.000,E  Longitude 11 deg 31.000' E
%      1            Fix quality: 0 = invalid
%                                1 = GPS fix (SPS)
%                                2 = DGPS fix
%                                3 = PPS fix
%                   4 = Real Time Kinematic
%                   5 = Float RTK
%                                6 = estimated (dead reckoning) (2.3
feature)
%                   7 = Manual input mode
%                   8 = Simulation mode
%      08           Number of satellites being tracked
%      0.9          Horizontal dilution of position
%      545.4,M      Altitude, Meters, above mean sea level
%      46.9,M       Height of geoid (mean sea level) above WGS84
%                       ellipsoid
%      (empty field) time in seconds since last DGPS update
%      (empty field) DGPS station ID number
%      *47          the checksum data, always begins with *
%
%Source: http://www.gpsinformation.org/dale/nmea.htm
%-------------------------------------------------------------------%
```

```
%read the GPS NMEA String
GPGGA_flag = 0;
fopen(s3);   %opens the COM port
%pull the next GPGGA strubg
while GPGGA_flag == 0
    A=fscanf(s3); %read serial port of GPS
    GPGGA_flag = strcmp(A(1:6),'$GPGGA'); %pick off the
                                          %GPGGA String
end
fclose(s3) %required to close each time so new GPS
           %coordinates get pulled for shot at new
           %location

%pull lat and lon from GPGGA string and put into
%useable form
a = mat2str(A); %Convert to string
a_delim = strsplit(a,','); %delimit with comma
lat_lon_cell = a_delim(3:6);%in cell structure
lat_lon_str= cell2mat(lat_lon_cell); %grab string from
                                     %cells
%grab each required string
lat_str = lat_lon_str(1:9);
lat_dir = lat_lon_str(10);
lon_str = lat_lon_str(11:20);
lon_dir = lat_lon_str(21);
%convert lat and lon to float
lat=str2num(lat_str);
lon = str2num(lon_str);
lat = lat/100;
lon = lon/100;
%get into correct decimal form
lat_int = floor(lat);
lat_dec = lat-lat_int;
lon_int = floor(lon);
lon_dec = lon-lon_int;

lat_dec_deg = lat_dec*100;
lon_dec_deg = lon_dec*100;

lat_true_decimal = lat_dec_deg/60;
lon_true_decimal = lon_dec_deg/60;

lat = lat_int+lat_true_decimal;
lon = lon_int+lon_true_decimal;
%deal with directions
if lat_dir == 'S'
    lat = -1*lat;
end

if lon_dir == 'W'
    lon = -1*lon;
end

%Final Coordinates
```

```
                centerLatitude = lat
                centerLongitude = lon

                %Desk in SP-521 for analytic purposes
                %centerLatitude = 36.595033;
                %centerLongitude = -121.874774;

%% Pass to COC and end all loops
                pass_to_COC = [pass_to_COC, centerLatitude,…
centerLongitude, firing_azimuth,shot_count];
                fwrite(t2,pass_to_COC,'double'); %sends over TCPIP to
                                                 %COC computer
                pass_to_COC = []; %empty the matrix to ready for the
                                  %next shot

            end
        end
    end
    sample_count = sample_count+1; %Increment
end

%plot each set of 15000 points of acceleration vs time, with shot
%finding alorithm points overlayed. Will need to copy and paste from
%here down into command window, as ctrl c will have ended the program
time_and_plot_real_time_record_4_5ms_correct_labeling

%use these to close all opened serial and network ports
fclose(t2)
fclose(s1)
fclose(s2)
fclose(s3)
```

2. YEI TSS-DL Setup

```
%%setupstreaming_header_Accel_and_taredEuler.m  By Captain Kiel Reese
%It sets up the YEI TSS-DL to stream corrected accelerations and tared
%Euler angles continuously and as soon as it is commanded to do so in
%the rifle node program, as fast as possible. All quotation comments
%were taken from YEI example Python codes that have since been removed
%from the YEI website.

%%Establish required YEI commands for use in building communication
%packets. All values are taken from the YEI TSS-DL user's manual.

TSS_START_BYTE = uint8(247);
TSS_SET_WIRED_RESPONSE_HEADER_BITFIELD = uint8(221);
GAP_BYTE = uint8(0);

TSS_GET_CORRECTED_RAW_ACCEL_DATA = uint8(39);
TSS_GET_TARED_EULER = uint8(1);
TSS_GET_UNTARED_EULER = uint8(7);
```

```
TSS_NULL = uint8(255); %”No command, used to fill the empty slots in
                       %”set stream slots””

TSS_SET_STREAMING_SLOTS = uint8(80);
TSS_SET_STREAMING_TIMING = uint8(82);

%response header options-only will need microsecond time stamp
TSS_RH_SUCCESS_FAILURE = uint8(1);
TSS_RH_TIMESTAMP = uint8(2);
TSS_RH_COMMAND_ECHO = uint8(4);
TSS_RH_CHECKSUM = uint8(8);
TSS_RH_LOGICAL_ID = uint8(16);
TSS_RH_SERIAL_NUMBER = uint8(32);
TSS_RH_DATA_LENGTH = uint8(64);

disp('TSS_SET_UP_RESPONSE_HEADER')

%Build the packet that will set the YEI TSS-DLs header, which will be
%pre-pended to all returned packets
write_bytes = [];
write_bytes = [write_bytes, TSS_START_BYTE];
write_bytes = [write_bytes, TSS_SET_WIRED_RESPONSE_HEADER_BITFIELD];
write_bytes = [write_bytes, GAP_BYTE, GAP_BYTE, GAP_BYTE,
TSS_RH_TIMESTAMP];

write_bytes_check_sum = write_bytes(2:length(write_bytes));
check_sum = uint8(mod((sum(write_bytes_check_sum)), 256)); %puts
                                                    %checksum into bits
write_bytes = [write_bytes, check_sum]; %add checksum byte

fwrite(s1, write_bytes,'uint8'); %writes this opening set of bits to
                                 %the designated serial port. Serial
                                 %port must be designated and opened in
                                 %rifle program
disp('TSS_SET_STREAMING SLOTS')

%Build the packet that tells the YEI TSS-DL what to stream
%”There are 8 streaming slots available for use, and each one can hold
%one of the streamable commands. Unused slots should be filled with
%0xff so that they will output nothing”
write_bytes = [];
write_bytes = [write_bytes, TSS_START_BYTE];
write_bytes = [write_bytes, TSS_SET_STREAMING_SLOTS];
write_bytes = [write_bytes, TSS_GET_CORRECTED_RAW_ACCEL_DATA]; %”stream
                                                               %slot0”
write_bytes = [write_bytes, TSS_GET_TARED_EULER]; % “stream slot1”
write_bytes = [write_bytes, TSS_NULL]; % “stream slot2”
write_bytes = [write_bytes, TSS_NULL]; % “stream slot3”
write_bytes = [write_bytes, TSS_NULL]; % “stream slot4”
write_bytes = [write_bytes, TSS_NULL]; % “stream slot5”
write_bytes = [write_bytes, TSS_NULL]; % “stream slot6”
write_bytes = [write_bytes, TSS_NULL]; % “stream slot7”

write_bytes_check_sum = write_bytes(2:length(write_bytes));
```

```
check_sum = uint8(mod((sum(write_bytes_check_sum)), 256)); %puts
                                                    %checksum into bits
write_bytes = [write_bytes, check_sum]; %add checksum byte

fwrite(s1, write_bytes,'uint8'); %writes this opening set of bits to
                                 %the designated serial port. Serial
                                 %port must be designated and opened in
                                 %the rifle node program

disp('TSS_SET_STREAMING_TIMING')

%"Interval determines how often the streaming session will output data
%from the requested commands. An interval of 0 will output data at the
%max filter rate."

 interval = uint8(0);% microseconds, needs to be 4 bytes long
 interval_byte3 = uint8(17);
 interval_byte4 = uint8(48);

%Duration determines how long the streaming session will run for
%"A duration of 0xffffffff will have the streaming session run till the
%stop stream command is called"

duration = uint8(255);%0xffffffff % microseconds 4 bytes long

%"Delay determines how long the sensor should wait after a start
%command is issued to actually begin streaming"

delay = uint8(0); % microseconds, 4 bytes long

write_bytes = [];
write_bytes = [write_bytes, TSS_START_BYTE];
write_bytes = [write_bytes, TSS_SET_STREAMING_TIMING];
write_bytes = [write_bytes, interval, interval, interval, interval];
write_bytes = [write_bytes, duration, duration, duration, duration];
write_bytes = [write_bytes, delay, delay, delay,delay];

%put into bigEndian format

write_bytes = swapbytes(write_bytes);
write_bytes_check_sum = write_bytes(2:length(write_bytes));
check_sum = uint8(mod((sum(write_bytes_check_sum)), 256)); %puts
                                                    %checksum into bits

write_bytes = [write_bytes, check_sum]; %add checksum byte

fwrite(s1, write_bytes,'uint8'); %writes this opening set of bits to
                                 %the designated serial port. Serial
                                 %port must be designated and
                                 %opened in main program
```

3. Real-Time Plotting

```
%time_and_plot_real_time_record_4_5ms_correct_labeling
%By Capt Kiel Reese
%This program is run after shooting to plot each reset worth of data
%with variables overlayed for analytic purposes

for ii = 1:reset_count %plots each reset of 15000, or ~each minute of
                       %data
    if ii == 1 %the first reset_count of data has issues with the first
               %data sets, due to header issue

       %obtain the actual time to plot against
       time = timestamp(ii,:);
       timeoffset = time-time(2);%timestamps are a relative value, need
                                 %to subtract out the first valid value
       true_time = timeoffset/1000000; %puts into seconds
       truetime = true_time(2:length(true_time)); %sizes it correctly
       for jj =2:length(truetime)
           time_step(jj-1) = truetime(jj)-truetime(jj-1); %to check for
                                                  %consistent time steps
       end

       %plot acceleration data with shot-finding algorithm overlays
       figure(ii)
       plot(truetime,x_accel(ii, 2:length(x_accel)))
       hold on
       plot(truetime.*peaks(ii, 2:length(x_accel)), peaks(ii,…
                                          2:length(x_accel)),'g*')

       %next lines was used in analysis but removed for consistency
       %naming conventions
       %shot_aiming_plot = shot_aiming(ii,:)+0.05;
       %plot(truetime.*shot_aiming(ii, 2:length(x_accel)),…
                        shot_aiming_plot(ii, 2:length(x_accel)),'b*')
       %shot_candidates_plot = shot_candidates(ii,:)+.1;
       %plot(truetime.*shot_candidates(ii, 2:length(x_accel)),…
                   shot_candidates_plot(ii, 2:length(x_accel)),'m*')
       aiming_plot = aiming(ii,:)+.15;
       plot(truetime.*aiming(ii, 2:length(x_accel)), aiming_plot(ii,…
                                          2:length(x_accel)),'k*')
       shot_plot = shot(ii,:)+.2;
       plot(truetime.*shot(ii, 2:length(x_accel)), shot_plot(ii,…
                                          2:length(x_accel)),'r*')
       legend('X Axis Acceleration', 'Shot Candidates','Aiming …
Samples','Shots')
       hold on
       %next line used to overlay differential power
       %plot(truetime, window_power(ii, 2:length(x_accel))/100,'r')

    else %all remaining sets of reset_count data, start after the 62nd
         %data point obtain the actual time to plot against
       time = timestamp(ii,:);
       timeoffset = time-time(70);
```

```
        true_time = timeoffset/1000000;
        truetime = true_time(70:length(true_time));
        for kk =2:length(truetime)
            time_step(kk-1) = truetime(kk)-truetime(kk-1);
            if truetime(kk) < 0
                truetime(kk) = truetime(length(find(truetime > 0))+1);
            end
        end
        %creates a new figure for every reset with shot-finding
        %algorithm overlays
        figure(ii)
        plot(truetime,x_accel(ii, 70:length(x_accel)))
        hold on
        plot(truetime.*peaks(ii, 70:length(x_accel)), peaks(ii,…
                                                70:length(x_accel)),'g*')
        %next lines was used in analysis but removed for consistency in
        %naming conventions
        %shot_aiming_plot = shot_aiming(ii,:)+0.05;
        %plot(truetime.*shot_aiming(ii, 70:length(x_accel)),…
                            shot_aiming_plot(70:length(x_accel)),'b*')
        %shot_candidates_plot = shot_candidates(ii,:)+.1;
        %plot(truetime.*shot_candidates(ii, 70:length(x_accel)),…
                            shot_candidates_plot(70:length(x_accel)),'m*')
        aiming_plot = aiming(ii,:)+.15;
        plot(truetime.*aiming(ii, 70:length(x_accel)),…
                                aiming_plot(70:length(x_accel)),'k*')
        shot_plot = shot(ii,:)+.2;
        plot(truetime.*shot(ii, 70:length(x_accel)),…
                                shot_plot(70:length(x_accel)),'r*')
        legend('X Axis Acceleration', 'Shot Candidates','Aiming…
Samples','Shots')
        hold on
        %next line used to overlay differential power
        %plot(truetime, window_power(ii, 70:length(x_accel))/100,'r')
    end
end
```

4. COC Node

```
%%google_earth_from_server_send.m  By Captain Kiel Reese
%This program runs in the COC. It receives data from the rifle nodes
%over TCPIP and plots the GPS coordinates of the rifle and azimuth of
%fire in Google Earth.

close all
clear all
clc
disp('GoogleEarth_from_server_send')

%Set up TCPIP to receive lat, lon, firing az, and shot count from any
%other machine

t2 = tcpip('0.0.0.0', 10000, 'NetworkRole', 'server');
t2.TimeOut = 1;
fopen(t2);
```

```
disp('Connected to Firearm')
pause(1)

%Run continuously looking for data in the TCPIP port will generate a
%warning in the command window if no shot occurs, this is fine

shot_data_record = zeros(1,10); %preallocate the recording array

while 1
    COC_receives = fread(t2,4,'double'); %read the TCPIP port
    no_shot = isempty(COC_receives); %if timeout with no shot, keep
                                     %checking

    if no_shot == 0 %if no_shot is 0, then a shot occurred and there
                    %was data in the TCPIP port
        clc %cleans up the warning signs for a fresh screen

        %Assign variables to data received from the rifle node
        centerLatitude = COC_receives(1);
        centerLongitude = COC_receives(2);
        firing_azimuth = COC_receives(3);
        shot_count = COC_receives(4);

        ggl_earth_map %uses the above variables and Matlab Mapping Tool
                      %Box to generate required coordinates
        ggl_earth_kmlwrite %uses generated coordinates to produce .kml
                           %files and opens in GoogleEarth

        %record all data
        shot_data_record(shot_count,1:4) = [COC_receives(4),…
        COC_receives(1:3)'];
        shot_data_record(shot_count,5:10) = time_num;

    end

end
```

5. Map Overlay Calculations

```
%%ggl_earth_map.m  By Captain Kiel Reese
%Works with ggl_earth_kmlwrite.m. This file uses Matlab Mapping Toolbox
%functions to calculate required coordinates to build kml files for
%Google Earth.

%Latitude and Longitude received from sensor and provided from
%ggl_earth_from_server_send.m

%Lab location for testing purposes
    %centerLatitude = 36.5950330000001;
    %centerLongitude = -1.218747740000e+02;
    %firing_azimuth = 0;
    %shot_count = 1;
```

```
%radius is maximum effective range for an area target of 5.56mm NATO
%ammunition
radius = 800;
az = [];
e = wgs84Ellipsoid; %scircle1 needs the geoid to calculate off of
[lat, lon] = scircle1(centerLatitude, centerLongitude, radius, az,…
                      e);%unused 800m radius

%% Calculates the firing azimuth and error lines from center out to
%maximum effective range of 5.56mm

dist=nm2deg(800/1852); %convert from meters to nautical miles, then
                        %nm2deg puts into degrees of arc length

%% Define all errors for coordinate calculations
right_az = firing_azimuth+90; %in degrees
left_az = firing_azimuth-90; %in degrees
GPS_error = 50; %in meters. 50 meters chosen heuristically
GPS_error_nm = nm2deg(GPS_error/1852);
pos_error = 5; %in degrees
neg_error = -5; %in degrees

%Left and right error azimuths
pos_error_deg = firing_azimuth+pos_error;
neg_error_deg = firing_azimuth+neg_error;

%% Calculate coordinates for kml development

%coordinates of the firing azimuth at 800m max effective range
[lattarget,lontarget] = reckon(centerLatitude, centerLongitude, dist,…
                        firing_azimuth);

%Generate coordinates on the GPS error ring orthogonal to the firing
%azimuth.
[lat_right_error,lon_right_error] = reckon(centerLatitude,…
                              centerLongitude, GPS_error_nm, right_az);
[lat_left_error,lon_left_error] = reckon(centerLatitude,…
                              centerLongitude, GPS_error_nm, left_az);

%Generate coordinates on the max effective range ring for error
%generated by azimuth error and GPS error ring

[lattarget_right,lontarget_right] = reckon(lat_right_error,…
                              lon_right_error, dist, pos_error_deg);
[lattarget_left,lontarget_left] = reckon(lat_left_error,…
                              lon_left_error, dist, neg_error_deg);

%Generate full sets of coordinates showing the track of 3 lines between
%6 points(1. Rifle location along azimuth, 2. Right outer edge of GPS
%error ring along right azimuth error line, 3. Left outer edge of GPS
error ring along left azimuth error line).
```

```
[lattargetplot,longtargetplot] = track2('gc', centerLatitude,…
                    centerLongitude, lattarget,lontarget);
[latposerrorplot, longposerrorplot] = track2('gc', lat_right_error,…
                    lon_right_error, lattarget_right,lontarget_right);
[latnegerrorplot,longnegerrorplot] = track2('gc', lat_left_error,…
                    lon_left_error, lattarget_left,lontarget_left);

%GPS Error Circle

[lat_GPS_error, lon_GPS_error] = scircle1(centerLatitude,…
                    centerLongitude, GPS_error, right_az, e);
```

6. Create KML File and Map in Google Earth

```
%%ggl_earth_kmlwrite.m  By Captain Kiel Reese
%Works with ggl_earth_map.m and writes kml files that are then
%displayed in Google Earth.

%Insert Marines name and rank for display

Marines_Name = 'Schmuckatelli';
Marines_Rank = 'pfc';

%Shot_count will be received from rifle code

sn = num2str(shot_count); %need shot number as a string for labeling
                          %purposes
ext = '.kml';
pic_ext = '.png';

%% Create filenames to display in Google Earth

filename1 = 'Location';
filename2 = 'shot_end_point';
filename3 = 'shot_azimuth';
filename4 = 'pos_error_az';
filename5 = 'neg_error_az';
filename6 = 'GPS_error_ring';

%Shot number ensures that files are different. Avoids overwriting in
%Google Earth. In Google Earth, operator can deselect any overlay not
%desired
filename1 = [filename1,'_',Marines_Name,'_#', sn, ext]; %first shot
                  %-->filename1 becomes 'Location_Schmuckatelli_#1.kml'
filename2 = [filename2,'_',Marines_Name,'_#', sn, ext];
filename3 = [filename3,'_',Marines_Name,'_#', sn, ext];
filename4 = [filename4,'_',Marines_Name,'_#', sn, ext];
filename5 = [filename5,'_',Marines_Name,'_#', sn, ext];
filename6 = [filename6,'_',Marines_Name,'_#', sn, ext];

%% Create name with time stamp for GPS location of rifle
```

```
time = datestr(clock);
name1 = time;
name1 = [name1,' ',Marines_Name,' Shot#', sn]%displays Marine name and
                                              %shot number, along with
                                              %time stamp

%displays rank icon. File path will need changed depending on where
picture files are found
icon = fullfile('\\special\kareese$', 'Thesis',
'Integration_Attempts','GPS_Looped_on_pass_shot','Ggl_Earth_Scipts');
pic_file = [Marines_Rank, pic_ext]; %example: pfc.png. All rank
                                    %insignia must be in this file,
                                    %named correctly

iconFilename = fullfile(icon, pic_file);

%% Create KML Files for Marine's location at time of shot, point at end
of target azimuth, error azimuths, and GPS error Circles

kmlwrite(filename1, centerLatitude, centerLongitude, 'Name',…
         name1,'Icon', iconFilename);
% name2 = 'Shot End Point';
% kmlwrite(filename2, lattarget, lontarget, 'Name', name2);
kmlwriteline(filename3, lattargetplot, longtargetplot, 'Color', 'red',…
            'Width', 2)
kmlwriteline(filename4,latposerrorplot,longposerrorplot,…
             'Color','black', 'Width', 2)
kmlwriteline(filename5, latnegerrorplot,longnegerrorplot,…
            'Color','black', 'Width', 2)
kmlwriteline(filename6, lat_GPS_error, lon_GPS_error, 'Color','black',…
            'Width', 2)

time_num = clock;

%% Open all the files in Google Earth. Pauses are included to give
Google Earth time.
winopen(filename1)
pause(.5)
% winopen(filename2)
% pause(.5)
winopen(filename3)
pause(.5)
winopen(filename4)
pause(.5)
winopen(filename5)
pause(.5)
winopen(filename6)

%Delete the files in Matlab directory to avoid overwhelming. Shot
%number change will keep the indiviual files distinguishable in Google
%Earth

pause(1)
delete(filename1)
```

```
pause(.2)
% delete(filename2)
% pause(.2)
delete(filename3)
pause(.2)
delete(filename4)
pause(.2)
delete(filename5)
pause(.2)
delete(filename6)
```

7. Real-Time Simulation Code, Rifle Node

```
%%real_time_simulator_WITH_diffpower_working.m   By Capt Kiel Reese
%This script takes the Rifle Node data collected at the second PA
%shooting and clears out all the data with the exception of
%accelerometer, orientation and timestamp data(that which was sent by
%the sensor). It then loops through this data to re-enact the data
%processing as if it was happening in real time. Runs the same
%algorithm as the rifle node program. All the GPS data and firing
%azimuth was recorded on the COC/server side. Designed for avg of 4.5ms
%between readings of the YEI sensor (in Kalman Mode)

close all
clear all
clc

%Define names of files
A = 'dad_7_shots_rifle';
B = 'day2_first_5_rifle';
C = 'day2_shoot_diffspots_rifle';
D = 'double_taps_rifle';
E = 'filmed_two_shots_rifle';
F = 'first_4_shots_rifle';
G = 'meggy_6_shots_rifle';
H = 'rack_rifle';
I ='second_set_5_rifle';
J = 'send_bolt_home_rifle';
K = 'swing_shoot_rifle';
%L = 'test_gun_side';
M = 'third_set_6_rifle';
N = 'walk_n_shoot_2_rifle';

%Load the desired workspace variables and clear out all variables not
%provided by the YEI TSS-DL
load(D)
clearvars -except x_accel timestamp orientation_data

%% Variables for shot finding
sample_count = 1;
n = 15001; %is greater than the 15000
aiming_threshold =.115;
aiming_difference = zeros(1,n-2);
aiming_initial = zeros(1,n-2);
```

```
aiming = zeros(1,n-2);
shot_aiming = zeros(1,n-2);
summer = zeros(1,n-2);
shot = zeros(1,n-2);
shot_candidates = zeros(1,n-2);
peaks = zeros(1,n-2);
difference = zeros(1,n-2);
window_power = zeros(1,n-2);
pass_shot = 0;
reset_count = 1;

%Variables for implementation of Rapid Fire
aimed_shot = zeros(1,n-2);
rapid_shot1 = zeros(1,n-2);
rapid_shot2 = zeros(1,n-2);

%% Variables for GPS
GM_Monterey = 13.38; %GM is East.
shot_count = 0;
pass_to_COC=[];

%% Loop variables. break_flag and reset_number needed to break while
loop after all data has been looped through
ii =1;
dim_accel = size(x_accel);
reset_number = dim_accel(1);
break_flag = 0;

while ii < length(x_accel)+2 && break_flag == 0
    sample_count = ii;

    %first if statement limits data size and keeps index greater than a
    %cycle time + rapid time. Required for the first reset, so that
    %data is available to start checking against in the index
    if sample_count>= length(x_accel)
       %Jumps to next row of recorded variables and builds the next row
        %for all tracking arrays
        disp('Reset')
        reset_count = reset_count + 1;
        ii=69;
        sample_count = 69;
        aiming_difference = [aiming_difference;zeros(1,n-2)];
        aiming_initial = [aiming_initial; zeros(1,n-2)];
        aiming = [aiming;zeros(1,n-2)];
        shot_aiming = [shot_aiming; zeros(1,n-2)];
        summer = [summer; zeros(1,n-2)];
        shot = [shot; zeros(1,n-2)];
        shot_candidates = [shot_candidates; zeros(1,n-2)];
        peaks = [peaks; zeros(1,n-2)];
        difference = [difference; zeros(1,n-2)];
        window_power = [window_power; zeros(1,n-2)];
        aimed_shot = [aimed_shot; zeros(1,n-2)];
        rapid_shot1 = [rapid_shot1; zeros(1,n-2)];
        rapid_shot2 = [rapid_shot2; zeros(1,n-2)];
```

```
    end
%%The same logic from the real-time rifle node code is below. This
%allows changes to be evaluated on the real-time data.

 %% Build aiming criteria
        if sample_count<68 %this if statement required because all
%thresholds are based off of previous data. Therefore, the farthest
%reach-back necessary (of the first iteration) needs to be taken into
%account.
            aiming_difference(reset_count, sample_count) = 1;
            aiming_initial(reset_count,sample_count) = 1;
        else
            aiming_difference(reset_count, sample_count) =…
x_accel(reset_count, sample_count)-x_accel(reset_count, sample_count…
-5); %check against a previous sample (~40ms before, can be adjusted.
                                        %Originally was ~20ms)

            %Check for aiming difference staying within the threshold
            %for aiming
            if abs(aiming_difference(reset_count, sample_count)) >…
                                                    aiming_threshold
                aiming_initial(reset_count, sample_count) = 0;
            else
                aiming_initial(reset_count, sample_count) = 1;
            end

            %if aiming for ~90ms consecutively, consider the rifle as
            %aiming
            summer(reset_count, sample_count) =…
       sum(aiming_initial(reset_count, sample_count-21:sample_count));
            if summer(reset_count, sample_count) == 22
                aiming(reset_count, sample_count)= 1;
            else
                aiming(reset_count, sample_count) = 0;
            end

            %Find when accelerations break the shot threshold. More
            %sensitive to shots along gravity vector (shooting up or
            %down)
            if abs(x_accel(reset_count, sample_count)) > 1.75
                peaks(reset_count, sample_count) = 1;
            end

            %check for breaking the shot threshold when recently aiming
            %(~48ms).
            %This is only “recently” aiming because the acceleration
            %from the hammer fall will break the aiming threshold
            %acceleration immediately prior to the first peak
            %acceleration
            if peaks(reset_count, sample_count) == 1 &&…
                                aiming(reset_count, sample_count-11)==1
                shot_aiming(reset_count, sample_count) = 1;
            end
```

```
            %look for hammer fall, last point before peak was not
            %classified as aiming or there was a significant
            %acceleration
            %change within the last 4 samples (too recent to affect
            %aiming classification)
            if shot_aiming(reset_count, sample_count) == 1 &&…
sum(shot_aiming(reset_count, sample_count-37:sample_count)) == 1 &&…
(aiming(reset_count, sample_count-1) == 0 || (abs(x_accel(reset_count,…
sample_count-3)-x_accel(reset_count, sample_count-4)) > …
aiming_threshold || abs(x_accel(reset_count, sample_count-2)…
-x_accel(reset_count, sample_count-4)) > aiming_threshold ||…
abs(x_accel(reset_count, sample_count-3)-x_accel(reset_count,…
sample_count-4)) > aiming_threshold))
                shot_candidates(reset_count, sample_count) = 1;
            end

            %build differential power criteria
            difference(reset_count, sample_count) =…
abs(x_accel(reset_count, sample_count)-x_accel(reset_count,…
                                                sample_count-1));
            window_power(reset_count, sample_count)= …
sum(difference(reset_count, sample_count-37:sample_count).^2);

            %looks for at least 3 peak accelerations within a cycle
            %time(could change to half a cycle time) with a recent
            %hammer fall, with recent aiming. Gets rid of misfires
            %where hammer falls, but there is only one or two peaks of
            %acceleration
            if peaks(reset_count, sample_count) ==1 &&…
sum(shot_candidates(reset_count, sample_count-37:sample_count)) == 1…
&& sum(peaks(reset_count, sample_count-15:sample_count)) >= 3…
&&  window_power(reset_count, sample_count)>36 && …
sum(aimed_shot(reset_count, sample_count-37:sample_count))== 0
                aimed_shot(reset_count, sample_count) = 1;
            end

            %look for first rapid shot
            if peaks(reset_count, sample_count) == 1 &&…
sum(peaks(reset_count, sample_count-12:sample_count))>=3 &&…
sum(aimed_shot(reset_count, sample_count-67:sample_count))==1 &&…
sum(aimed_shot(reset_count, sample_count-37:sample_count)) == 0 &&…
sum(rapid_shot1(reset_count, sample_count-37:sample_count)) == 0 &&…
window_power(reset_count, sample_count)>36
                rapid_shot1(reset_count, sample_count) = 1;
            end

            %look for second rapid shot
            if peaks(reset_count, sample_count) == 1 &&…
sum(peaks(reset_count, sample_count-12:sample_count))>=3 &&…
sum(rapid_shot1(reset_count, sample_count-67:sample_count))==1 &&…
sum(rapid_shot1(reset_count, sample_count-37:sample_count)) == 0 &&…
sum(rapid_shot2(reset_count, sample_count-37:sample_count)) == 0 &&…
window_power(reset_count, sample_count)>36
                rapid_shot2(reset_count, sample_count) = 1;
            end
```

```
            %report any shots
            if aimed_shot(reset_count, sample_count) ==1 ||…
rapid_shot1(reset_count, sample_count) ==1 || rapid_shot2(reset_count,…
sample_count) == 1
                shot(reset_count, sample_count) = 1;

%From here down, shot actually occurred
%                clc
%                pass_shot = pass_shot + 1
                shot_count = shot_count+1


            end
        end

    sample_count = sample_count+1; %increment
    ii = ii +1; %increment

    %ensures proper number of resets. Breaks the loop once all data has
    %been cycled through
    if reset_count == reset_number
        break_flag = 1;
    end
end

%plot each set of 15000 points of acceleration vs time, with shot
%finding alorithm points overlayed. Will need to copy and paste from
%here down into command window, as ctrl c will have ended the program

time_and_plot_real_time_record_4_5ms_correct_labeling
```

8. Real-Time Simulation Code, COC Node

```
%% reenact_from_server_side.m  By Capt Kiel Reese
%This code takes the COC side saved workplace variables from the
%second PA shooting and re-enacts the mapping in Google Earth as if the
%shooting was happening in real time. The only difference is that the
%time of the shots was not recorded, so the markers in Google Earth are
%not timestamped.

% close all
clear all
%Define the names of the files
A = 'dad_7_shots_coc';
B = 'day2_first_5_COC';
C = 'day2_shoot_diffspots_coc';
D = 'double_taps_coc';
E = 'filmed_two_shots_coc';
F = 'first_4_shots_coc';
G = 'meggy_6_shots_COC';
H = 'rack_coc';
I ='second_set_5_coc';
J = 'send_bolt_home_COC';
```

```
K = 'swing_shoot_coc';
%L = 'test_gun_side';
M = 'third_set_6_coc';
N = 'walk_n_shoot_2_coc';
load(A)

%% Loop Through the Recorded Data of Each Shot and Map in Google Earth
for ii = 1:size(shot_data_record,1)
centerLatitude = shot_data_record(ii,2);
centerLongitude = shot_data_record(ii,3);
shot_count = shot_data_record(ii,1)
firing_azimuth = shot_data_record(ii,4)%-17
time_num = shot_data_record(ii,5:10);

%same code as in the real-time code
ggl_earth_map%_small_error

Marines_Name = 'Schmuckatelli';
Marines_Rank = 'pfc';

%shot_count will be recieved from rifle code
sn = num2str(shot_count); %need shot number as a string for labeling
                          %purposes
ext = '.kml';
pic_ext = '.png';

%% Create filenames to display in Google Earth

filename1 = 'Location';
filename2 = 'shot_end_point';
filename3 = 'shot_azimuth';
filename4 = 'pos_error_az';
filename5 = 'neg_error_az';
filename6 = 'GPS_error_ring';

%Shot number ensures that files are different. Avoids overwriting in
%Google Earth. In Google Earth, operator can deselect any overlay not
%desired.
filename1 = [filename1,'_',Marines_Name,'_#', sn, ext]; %first shot
%example is ---->filename1 becomes 'Location_Schmuckatelli_#1.kml'
filename2 = [filename2,'_',Marines_Name,'_#', sn, ext];
filename3 = [filename3,'_',Marines_Name,'_#', sn, ext];
filename4 = [filename4,'_',Marines_Name,'_#', sn, ext];
filename5 = [filename5,'_',Marines_Name,'_#', sn, ext];
filename6 = [filename6,'_',Marines_Name,'_#', sn, ext];

%% Create name with time stamp for GPS Location of rifle
time = num2str(time_num);
name1 = [Marines_Name,' Shot#', sn];%displays Marine name and shot
                                    %number, along with time stamp

%displays rank icon. Will have to change the path name to folder
%containing the icon
```

```
icon = fullfile('\\special\kareese$', 'Thesis', 'PA June 2016 …
Realtime');
%icon = fullfile('C:\Users\hannahbear\Documents\NPS\Thesis\Code');
pic_file = [Marines_Rank, pic_ext]; %example: pfc.png. All rank
                  %insignia must be in this file, named correctly
iconFilename = fullfile(icon, pic_file);

%% Create KML Files for Marine's location at time of shot, point at end
%of target azimuth, error azimuths, and GPS error Circles
kmlwrite(filename1, centerLatitude, centerLongitude, 'Name',…
name1,'Icon', iconFilename);
% name2 = 'Shot End Point';
% kmlwrite(filename2, lattarget, lontarget, 'Name', name2);
kmlwriteline(filename3, lattargetplot, longtargetplot, 'Color', 'red',…
'Width', 2)
kmlwriteline(filename4,latposerrorplot,longposerrorplot, …
'Color','black', 'Width', 2)
kmlwriteline(filename5, latnegerrorplot,longnegerrorplot,…
'Color','black', 'Width', 2)
kmlwriteline(filename6, lat_GPS_error, lon_GPS_error, 'Color','black',…
'Width', 2)

%% Open all the files in Google Earth
winopen(filename1)
pause(.5)
% winopen(filename2)
% pause(.5)
winopen(filename3)
pause(.5)
winopen(filename4)
pause(.5)
winopen(filename5)
pause(.5)
winopen(filename6)

%% Delete the files in Matlab directory to avoid overwhelming. Shot
number change will keep the indiviual files distinguishable in Google
Earth
pause(1)
delete(filename1)
pause(.2)
% delete(filename2)
% pause(.2)
delete(filename3)
pause(.2)
delete(filename4)
pause(.2)
delete(filename5)
pause(.2)
delete(filename6)
pause(2)
end
```

9. Python Code

```
##This program runs a Rifle Node on a Raspberry Pi. Requires a YEI TSS
#for shot identification on ttyACM1, a YEI TSS for orientation on
#ttyACM0, and a #GPS receiver on ttyUSB0. Upon identifying a shot,
#prints GPS coordinates and firing azimuth to the command line

#Import required modules
import serial
import struct
import time
import numpy
import math
#YEI TSS Command Variables
TSS_GET_TARED_EULER = 0x1
TSS_START_BYTE = 0xf7
TSS_GET_CORRECTED_RAW_ACCELERATION = 0x27
TSS_NULL = 0xff

#GPS_retrieve() opens a serial port to the GPS, and then parses out
#coordinates from the NMEA 0183 GPGGA sentence.
def GPS_retrieve():
    GGA_flag = 0
    GPS_serial = serial.Serial('/dev/ttyUSB0', 4800)#establish serial
                                                    #port
    while GGA_flag == 0:
        data_str = GPS_serial.readline() #read each line of GPS NMEA
                                         #Strings coming in
        data_string = ''.join(chr(i) for i in data_str) #convert from
                                #bytes to string, this can give issues
        if data_string[0:6] =='$GPGGA': #parse out only GPGGA to give
                                        #fix
            GGA_flag = 1 #causes while loop to end
            GPS_serial.close()#close the serial port
            #Convert Lat and Lon into numbers
            #Latitude
            lat_deg = float(data_string[18:20])
            lat_min = float(data_string[20:27])/60
            lat = lat_deg + lat_min
            lat_dir = data_string[28:29]
            #Longitude
            long_deg = float(data_string[30:33])
            long_min = float(data_string[33:40])/60
            lon = long_deg + long_min
            long_dir = data_string[41:42]
            #obtain directions
            if lat_dir == 'S':
                lat = -1*lat
            if long_dir =='W':
                lon = -1*lon

    #return coordinates
    return [lat,lon]
```

```
#Get_Orientation() opens a serial port to orientation YEI TSS, reads
#the yaw angle and converts to the actual firing azimuth.
def Get_Orientation():
    dataflag = 0
    #GM_Monterey = 13.38333333 #Grid-Magnetic angle in degrees for
    #Monterey, CA. Only need if YEI TSS was not tared to True North

    #establish and open the serial port to the orientation YEI TSS
    orientation_serial = serial.Serial('/dev/ttyACM0',timeout=0.2,…
                                       writeTimeout=0.2, baudrate=115200)
    data_struct = struct.Struct('>fff') #expected data structure
                                        #returned from YEI TSS
    #establish command to YEI to returned untared Euler Angles
    write_bytes=bytearray((TSS_START_BYTE,
                           TSS_GET_TARED_EULER,
                           TSS_GET_TARED_EULER))
    while dataflag == 0:
        orientation_serial.write(write_bytes) #sent command to the YEI
                                              #TSS
        data_str = orientation_serial.read(data_struct.size) #read the
                                                #YEI TSS returned data
        #ensures that the correct data type is returned. If not, write
        #and read until correct
        if len(data_str) != data_struct.size:
            dataflag = 0
        else:
            dataflag = 1
    data = data_struct.unpack(data_str) #put data in a useable form
    orientation_serial.close() #close the serial port
    yaw_angle = data[1] #yaw angle is the second data provided
    yaw_deg = yaw_angle*180/numpy.pi #convert to degrees
    orient_off_north = yaw_deg #+GM_Monterey#add GM angle

    #Correct for any values over 360 due to GM Angle addition and for
    #East being negative
    if orient_off_north > 360:
        orient_off_north = orient_off_north - 360
    if orient_off_north<=0:
        orient_off_north = 360-(-1*orient_off_north)

    #Calculate and add correction factor. Only valid for Spanagel Lab
    #if orient_off_north >220 and orient_off_north < 340.2:
    #argument = (orient_off_north - 133.2)*numpy.pi/180
    #correction = (14.5*numpy.absolute(math.cos(argument)))+5.6
    #orient_off_north = orient_off_north+correction


    firing_azimuth = orient_off_north

    #return calculated firing azimuth
    return firing_azimuth

##/////////////////MAIN/////////////////////////////////////
#establish and open serial port to shot identifying YEI TSS
```

```
serial_port = serial.Serial('/dev/ttyACM1', timeout=0.2,
writeTimeout=0.2, baudrate=115200)
data_struct = struct.Struct('>fff')#expected data structure
#YEI Command for Acceleration in Gs
write_bytes=bytearray((TSS_START_BYTE,
                       TSS_GET_CORRECTED_RAW_ACCELERATION,
                       TSS_GET_CORRECTED_RAW_ACCELERATION))

#Establish required variables and tracking arrays
elapsed_array = []
accel_array = numpy.ones(3)

index = 0
sample_count = 0
n = 50001
aiming_threshold = .115
aiming_difference = []
aiming_initial = []
summer = []
aiming = numpy.zeros(72)
peaks = numpy.zeros(72)
shot_aiming = numpy.zeros(72)
shot_candidates = numpy.zeros(72)
aimed_shot = numpy.zeros(72)
rapid_shot1 = numpy.zeros(72)
rapid_shot2 = numpy.zeros(72)
shot = numpy.zeros(72)

print('Start')

try:
    while sample_count < n:
        t=time.time() #for analyzing how long each iteration takes
        serial_port.write(write_bytes) #command the YEI TSS
        data_str = serial_port.read(data_struct.size) #Read from the
                                                      #YEI TSS

        #ensures returned data is the correct format
        if len(data_str) != data_struct.size:
            continue
        data = data_struct.unpack(data_str) #put data in a useable form
        accel_array = numpy.vstack([accel_array, data]) #stack the new
                                                 #data beneath old data

        #begin shot identifying code...the same as the Matlab version
        if sample_count < 72:
            aiming_difference = numpy.append(aiming_difference, 1)
            aiming_initial = numpy.append(aiming_initial,1)
            summer = numpy.append(summer, 0)
        else:
            aiming_difference=numpy.append(aiming_difference,…
            accel_array[sample_count,1]-accel_array[sample_count-12,1])
            if numpy.absolute(aiming_difference[sample_count]) >…
                                                      aiming_threshold:
                aiming_initial = numpy.append(aiming_initial, 0)
```

```
            else:
                aiming_initial = numpy.append(aiming_initial,1)

            summer = numpy.append(summer,…
             numpy.sum(aiming_initial[sample_count-13:sample_count+1]))
            if summer[sample_count] == 14:
                aiming = numpy.append(aiming, 1)
            else:
                aiming = numpy.append(aiming, 0)

            if numpy.absolute(accel_array[sample_count,0]) > 1.75:
                peaks = numpy.append(peaks, 1)
            else:
                peaks = numpy.append(peaks, 0)

            if peaks[sample_count] == 1 and aiming[sample_count-13]==1:
                shot_aiming = numpy.append(shot_aiming,1)
            else:
                shot_aiming = numpy.append(shot_aiming, 0)

            if shot_aiming[sample_count] == 1 and…
numpy.sum(shot_aiming[sample_count-45:sample_count+1]) == 1:# and …
(aiming[sample_count-1] == 0 or…
numpy.absolute(accel_array[sample_count-2,0]-accel_array[sample_count-…
3,0]) > aiming_threshold):

                shot_candidates = numpy.append(shot_candidates, 1)
            else:
                shot_candidates = numpy.append(shot_candidates, 0)

            if peaks[sample_count] == 1 and …
numpy.sum(shot_candidates[sample_count-45:sample_count+1]) == 1 and …
numpy.sum(peaks[sample_count-18:sample_count +1]) == 3:

                aimed_shot = numpy.append(aimed_shot,1)
            else:
                aimed_shot = numpy.append(aimed_shot,0)

            if peaks[sample_count] == 1 and
numpy.sum(peaks[sample_count-13:sample_count+1]) >=3 and …
numpy.sum(aimed_shot[sample_count-72:sample_count+1]) == 1 and …
numpy.sum(aimed_shot[sample_count-45:sample_count+1])== 0 and …
numpy.sum(rapid_shot1[sample_count-45:sample_count+1]) == 0:

                rapid_shot1 = numpy.append(rapid_shot1,1)
            else:
                rapid_shot1 = numpy.append(rapid_shot1,0)

            if peaks[sample_count] == 1 and
numpy.sum(peaks[sample_count-13:sample_count+1]) >=3 and …
numpy.sum(rapid_shot1[sample_count-72:sample_count+1]) == 1 and …
numpy.sum(rapid_shot1[sample_count-45:sample_count+1])== 0 and …
numpy.sum(rapid_shot2[sample_count-45:sample_count+1]) == 0:

                rapid_shot2 = numpy.append(rapid_shot2,1)
```

```
            else:
                rapid_shot2 = numpy.append(rapid_shot2,0)

            if aimed_shot[sample_count] == 1 or …
                    rapid_shot1[sample_count] == 1 or …
                    rapid_shot2[sample_count] == 1:

                shot = numpy.append(shot, 1)
                [lat,lon] = GPS_retrieve()
                firing_az = Get_Orientation()
                print(lat, lon, firing_az)
            else:
                shot = numpy.append(shot,0)

        #print(accel_array[sample_count,1])
        index = index + 1
        sample_count = sample_count + 1

#keep arrays at size 100 to keep memory usage and processing time down
        if index >100:
           sample_count = 100
           accel_array = accel_array[1:]
           aiming_difference = aiming_difference[1:]
           aiming_initial = aiming_initial[1:]
           summer = summer[1:]
           aiming = aiming[1:]
           peaks = peaks[1:]
           shot_aiming = shot_aiming[1:]
           shot_candidates = shot_candidates[1:]
           aimed_shot = aimed_shot[1:]
           rapid_shot1 = rapid_shot1[1:]
           rapid_shot2 = rapid_shot2[1:]
           shot = shot[1:]

        #calculate and record time steps. Elapsed array is the only
        #array that grows continuously
        elapsed = time.time()-t
        elapsed_array = numpy.append(elapsed_array, elapsed)

#if cntl c is pressed, exit and close the serial port
except(KeyboardInterrupt):
    serial_port.flushInput()

serial_port.close()
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. A PRIORI ALGORITHM RESULTS

The a priori results are contained in this appendix. These results include testing across three days at three different data sampling rates. The total results are also presented.

1. RANGE DAY 1 DATA

Table 6. Results of Range Day One

Name of Data File	# Shots Actual	# Shots Found	False Positives	Missed Shots
firstkneeling_segura	10	10	0	0
firstprone_segura	8	8	0	0
firstshot_prone_segura	1	1	0	0
firststanding_segura	10	10	0	0
rapidmagchange_segura	10	10	0	0
secondkneeling_calusdian	10	10	0	0
secondprone_calusdian	9	9	0	0
secondshot_prone_segura	1	1	0	0
secondstanding_calusdian	11	11	0	0
thirdkneeling_reese	10	10	0	0
thirdprone_reese	10	10	0	0
thirdstanding_reese	10	10	0	0
First Range Totals	100	100	0	0

C. RANGE DAY 2 DATA

Table 7. Results of Range Day Two

Name of Data File	# Shots Actual	# Shots Found	False Positives	Missed Shots
combatglide25_10	10	10	0	0
magchanges	0	0	0	0
prone_childers	10	10	0	0
quickreactdoubletap_10yrd	20	20	0	0
rack_30	0	0	0	0
rack_dryfire	0	0	0	0
runtosingleshot	10	10	0	0
taprackbang1	5	5	0	0
taprackbang2	5	5	0	0
walktosingleshot	10	10	0	0
Second Range Totals	70	70	0	0

D. IMU MODE, 0.79 MS DATA

Table 8. Results of Data Collected in IMU Mode

Name of Data File	# Shots Actual	# Shots Found	False Positives	Missed Shots	3g/7peaks (FP/MS)
combatglide_imu	5	5	0	0	0/0
kneeling_imu	10	11	1	0	0/0
magchange_imu	10	10	1	1	0/1
quickreactdoubletap_imu	10	10	0	0	0/0
runtosingleshot_imu	5	5	0	0	0/0
standing_imu	10	10	0	0	0/0
walktosingleshot_imu	5	5	0	0	0/0
PA IMU Totals	55	56	2	1	0/1
% Correct	0.981818182		Adjusted Parameters	0.981818182	
% Missed	0.018181818			0.018181818	
% False Positives	0.036363636			0	

E. KALMAN, 2.6 MS DATA

Table 9. Results of Data Collected with 2.6 ms Time Steps

Name of Data File	# Shots Actual	# Shots Found	False Positives	Missed Shots
kneeling_realtime_kalman	10	10	0	0
kneeling_realtime_kalman2	10	10	0	0
kneeling_realtime_kalman3	10	10	0	0
magchange_realtime	10	10	0	0
rack5_realtime	0	0	0	0
rackandjam_realtime	1	1	0	0
standingtokneeling_realtime	5	6	1	0
PA Kalman Totals	46	47	1	0
% Correct	1			
% Missed	0			
% False Positives	0.02173913			

F. TOTAL RESULTS

Table 10. Total A Priori Results

	# Shots Actual	# Shots Found	False Positives	Missed Shots
Complete Totals	271	273	3	1
% Correct, with adjusted	0.996309963		adjusted = 1	
% Missed, with adjusted	0.003690037			
% False Positives, with adjusted	0.003690037			

Note: As discussed in Chapter IV, the adjusted values are obtained with the parameter values changed to account for higher sampling rates of the data obtained in IMU mode and Kalman mode at a time step of 2.6 ms.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. REAL-TIME TESTING RESULTS

The results from the second real-time test with the code in varying configurations are contained in this appendix.

A. WITH CODE ON DAY OF TESTING

Table 11. Second Real-Time Test Results with Code Configuration on the Day of Testing

Name of Data File	# Shots Actual	# Shots Found	False Positives	Missed Shots
dad_7_shots_rifle.mat	7	6	0	1
day2_first_5_rifle.mat	5	6	1	0
day2_shoot_diffspots_rifle.mat	5	5	0	0
double_taps_rifle.mat	13	11	0	2
filmed_two_shots_rifle.mat	2	2	0	0
first_4_shots_rifle.mat	4	4	0	0
meggy_6_shots_rifle.mat	6	6	0	0
rack_rifle.mat	0	0	0	0
second_set_5_rifle.mat	5	5	0	0
send_bolt_home_rifle.mat	0	1	1	0
swing_shoot_rifle.mat	8	7	0	1
third_set_6_rifle.mat	6	5	0	1
walk_n_shoot_2_rifle.mat	6	6	0	0
Real Time Totals	67	64	2	5
% Correct	**0.925373134**			
% Missed	**0.074626866**			
% False Positives	**0.029850746**			

B. WITH CORRECTED HAMMER FALL PROFILE

Table 12. Second Real-Time Test Results with Corrected Hammer Fall Profile

Name of Data File	# Shots Actual	# Shots Found	False Positives	Missed Shots
dad_7_shots_rifle.mat	7	7	0	0
day2_first_5_rifle.mat	5	6	1	0
day2_shoot_diffspots_rifle.mat	5	5	0	0
double_taps_rifle.mat	13	13	0	0
filmed_two_shots_rifle.mat	2	2	0	0
first_4_shots_rifle.mat	4	4	0	0
meggy_6_shots_rifle.mat	6	6	0	0
rack_rifle.mat	0	0	0	0
second_set_5_rifle.mat	5	5	0	0
send_bolt_home_rifle.mat	0	1	1	0
swing_shoot_rifle.mat	8	8	0	0
third_set_6_rifle.mat	6	6	0	0
walk_n_shoot_2_rifle.mat	6	6	0	0
Real Time Totals	67	69	2	0
% Correct	**1**			
% Missed	**0**			
% False Positives	**0.029850746**			

C. WITH FOUR PEAKS VICE THREE AND CORRECTED HAMMER FALL PROFILE

Table 13. Second Real-Time Test Results with Parameter Adjustments

Name of Data File	# Shots Actual	# Shots Found	False Positives	Missed Shots
dad_7_shots_rifle.mat	7	7	0	0
day2_first_5_rifle.mat	5	5	0	0
day2_shoot_diffspots_rifle.mat	5	5	0	0
double_taps_rifle.mat	13	13	0	0
filmed_two_shots_rifle.mat	2	2	0	0
first_4_shots_rifle.mat	4	3	0	1
meggy_6_shots_rifle.mat	6	6	0	0
rack_rifle.mat	0	0	0	0
second_set_5_rifle.mat	5	5	0	0
send_bolt_home_rifle.mat	0	0	0	0
swing_shoot_rifle.mat	8	8	0	0
third_set_6_rifle.mat	6	6	0	0
walk_n_shoot_2_rifle.mat	6	6	0	0
Real Time Totals	67	66	0	1
% Correct	**0.985074627**			
% Missed	**0.014925373**			
% False Positives	**0**			

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. FURTHER ORIENTATION DISCUSSION

Testing revealed that even when calibrated in the position used for testing in Spanagel Hall, an error of up to seven degrees was still apparent. This error is caused by local distortions in the magnetic field and were orientation-dependent but not time-dependent. This error was not seen in the second real-time test conducted in a rural setting; however, this was not specifically tested due to the lack of recognizable terrain features needed to establish base truth in the area of the second real-time test.

These distortions necessitated a further correction factor, which was calculated based on experimental data using a hasty declination station. This declination station utilized Google Maps to find coordinates of both the test station location and recognizable terrain features in the area. These features were corners of buildings and measured points on the walls of the laboratory. These measured points in the laboratory are based on the assumption that the walls of the laboratory are truly square. The first point listed is based on the assumption that Root Hall is truly perpendicular to Spanagel Hall. The full list of points is found in Table 14 with the locations shown in the imagery of Figure 26. In Figure 26, the test rifle is located at the yellow star while the red numerals correlate to points in the table.

An online calculator [36] was used to determine the azimuth from the test station to each point. These are the same calculations used in MATLAB for mapping purposes. This hasty declination station was used as the ground truth for comparing orientation readings when the rifle was sighted in on these points. A correction factor was calculated based on the differences with these azimuths to obtain repeatable accuracy on the order of one to two degrees; however, this correction factor is only valid for the test station location.

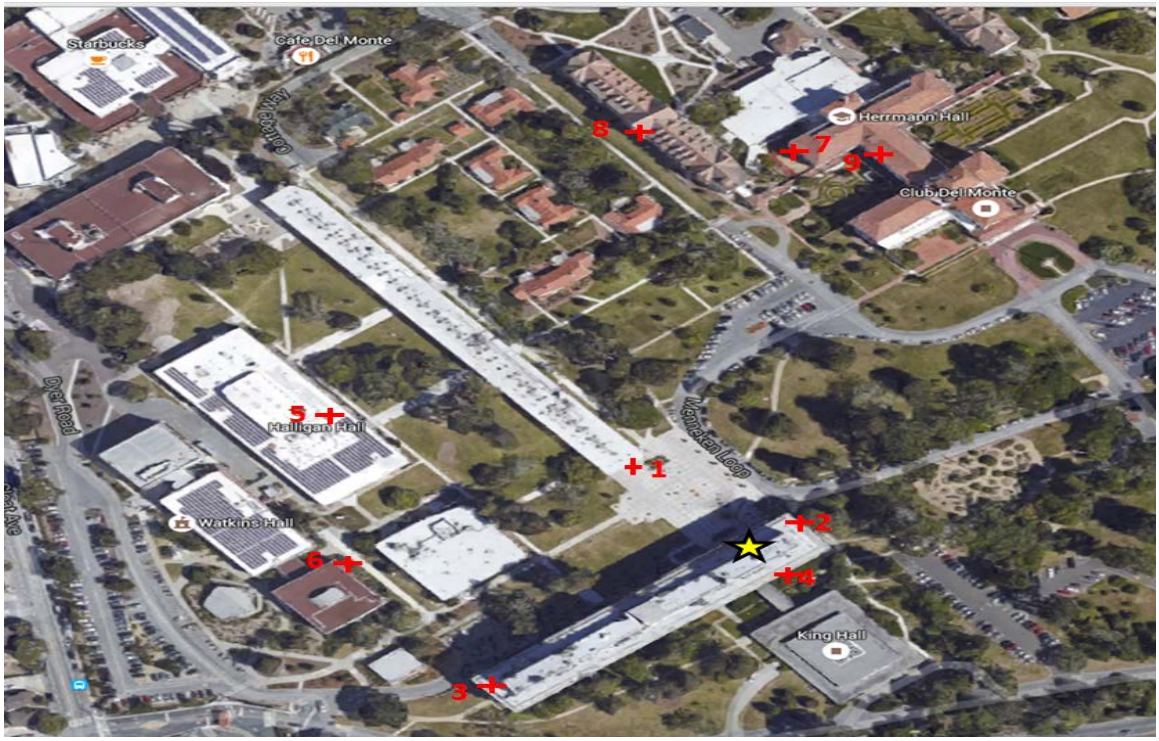

Figure 26. Declination Station Imagery

Table 14. List of Points Used for Declination Station

My Location: 36.595033, -121.874774		
Sensor 5		
Point	**Azimuth**	**Description**
1	318.8	Perpendicular to Root Hall
2	48.8	Perpendicular to Right Wall
3	228.8	Perpendicular to left wall
4	138.8	Perpendicular to wall directly behind
5	295.1	Closest corner of raised portion of Halligan Hall
6	266.8	Corner of brown building with dome shape
7	4.4	Corner of Hermann Hall
8	350.7	Middle Peak of building next to Herman
9	11.9	Left side of box on Herman Hall

LIST OF REFERENCES

[1] *Warfighting*, Marine Corps Doctrinal Publication 1, USMC, 1997, pp. 3–13.

[2] R. J. Dunn III, "Blue Force tracking: The Afghanistan and Iraq experience and its implications for the U.S. Army," Northrop Grumman Mission Syst., Reston, VA, FX2250903, 2003.

[3] N. J. Alexander, "Analysis of Marine Corps efforts in the pursuit of the joint Blue Force situational awareness capability," M.S. thesis, Management, Naval Postgraduate School, Monterey, CA, 2013.

[4] J. L. Rosen and J. W. Walsh, "The Nett Warrior System: A case study for the acquisition of soldier systems," Naval Postgraduate School, Monterey, CA, Tech. Rep. NPS-AM-11-187, 2011.

[5] DOT&E FY 15 Army Programs: Nett Warrior. (n.d.). DOT&E. [Online]. Available: http://www.dote.osd.mil/pub/reports/FY2015/pdf/army/2015nettwarrior.pdf. Accessed Mar. 23, 2016.

[6] *Marine Rifle Squad*, Marine Corps Warfighting Publication 3–11.2 w/ Ch1, USMC, 2002, Ch. 2, pp. 17-25.

[7] A. Newcomb. (2015, June 19). *Nett Warrior: Fort Campbell soldiers train with subject matter experts* [Online]. Available: http://www.army.mil/article/150852/Nett_Warrior__Fort_Campbell_Soldiers_train_with_subject_matter_experts/

[8] C. K. Khan, "Geometry-of-fire tracking algorithm for direct-fire weapon systems," M.S. thesis, Dept. Elect. & Comput. Eng., Naval Postgraduate School, Monterey, CA, 2015.

[9] J. C. Driesslein, "Scalable Mobile Ad Hoc Network (MANET) to enhance situational awareness in distributed small unit operations," M.S. thesis, Dept. Elect. & Comput. Eng., Naval Postgraduate School, Monterey, CA, 2015.

[10] *Rifle Marksmanship*, Marine Corps Reference Publication 3–01A, USMC, 2012, Ch. 3, pp. 20-22.

[11] S. Kuo. (2015, June 1). *Preview—Guide to super-light AR-15 bolt carrier groups* [Online]. Available: http://www.recoilweb.com/preview-guide-to-super-light-ar-15-bolt-carrier-groups-65941.html

[12] F. Morelli, J. M. Neugebauer, M. E. LaFiandra, P. Burcham, and C. T. Gordon, "Recoil measurement, mitigation techniques, and effects on small arms weapon design and marksmanship performance," *IEEE Trans. on Human-Mach. Syst*., vol. 44, no. 3, pp. 422–428, Jun. 2014.

[13] *Operator's Manual for Rifle, 5.56mm, M16A2 W/E-M16A3-M4 W/E*, TM 9–1005-319-10, U.S. Dept. of the Army, Air Force, and Navy, 2010, Ch. 1, Sect. 2, pp. 1-2.

[14] B. P. Burns, "Recoil considerations for shoulder-fired weapons," Army Res. Lab., Aberdeen Proving Grounds, MD, Rep. ARL-CR-692, May 2012.

[15] *Technical Manual for Rifle, 5.56mm, M16A2 W/E, Carbine, 5.56mm, M4, Carbine, 5.56mm, M4A1*, TM 9–1005-319-23&P, U.S. Dept. of the Army and Air Force, 1991, Sect. 1–10.

[16] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, "Sampling," in *Signals and Systems*, 2nd ed. Cambridge, MA: Prentice-Hall, 1996, pp. 514–521.

[17] M. Andrejašič, "MEMS accelerometers," Seminar, Dept. of Phys, Univ. of Ljubljana, Ljubljana, Slovenia, 2008.

[18] MMA8451Q, 3-Axis, 14-bit/8-bit, Digital Accelerometer. (2015, Jun.). Freescale Semiconductor. [Online]. Available: http://cache.nxp.com/files/sensors/doc/data_sheet/MMA8451Q.pdf. Accessed Apr. 5, 2016.

[19] *USB Accelerometer Model X16-1D.* (2014, Dec.). Gulf Coast Data Concepts. [Online]. Available: http://www.gcdataconcepts.com/GCDC_X16-1D_User_Manual.pdf. Accessed Mar. 22, 2016.

[20] *3-Space Sensor User's Manual.* (2014, Nov. 14). YEI Technology. [Online]. Available: http://www.yeitechnology.com/sites/default/files/YEI_TSS_Users_Manual_3.0_r1_4Nov2014.pdf. Accessed Mar. 22, 2016.

[21] LORD Datasheet 3DM-GX4-25 Attitude Heading Reference System (AHRS). (2014). LORD Corp. [Online]. Available: http://files.microstrain.com/3DM-GX4-25_Datasheet_(8400-0060).pdf. Accessed Mar. 22, 2016.

[22] MTi 10-series. (2015). Xsens North America Inc. [Online]. Available: https://www.xsens.com//download/pdf/documentation/mti-10/mti-10-series.pdf. Accessed Mar. 22, 2016.

[23] IMU-10. (2016). Sparton Corporation. [Online]. Available: https://www.spartonnavex.com/product/imu-10/. Accessed Mar. 22, 2016.

[24] C. E. Loeffler. (2014, Sept. 3). Detecting gunshots using wearable accelerometers. *Plos One* [Online]. 9(9). Available: http://journals.plos.org/plosone/article/asset?id=10.1371%2Fjournal.pone.0106664.PDF

[25] R. Parloff. (2015, Apr. 22). *Smart guns: They're ready. Are we?* [Online]. Available: http://fortune.com/2015/04/22/smart-guns-theyre-ready-are-we/

[26] Startup Grind Monterey Bay. (2015, Apr. 17). "Startup Grind Monterey Bay presents Bob Stewart of Yardarm Technologies." [YouTube video]. Available: https://www.youtube.com/watch?v=uuR6qFscdII. Accessed Mar. 22, 2016.

[27] C. Johnson, "A personal inertial navigation system based on multiple, distributed nine degrees-of-freedom inertial measurement units," M.S. thesis, Dept. Elect. & Comput. Eng., Naval Postgraduate School, Monterey, CA, 2016.

[28] BU-353S4 GPS Receiver. (2009). USGlobalSat Incorporated. [Online]. Available: http://usglobalsat.com/store/download/688/bu353s4_ds.pdf. Accessed Apr. 5, 2016.

[29] D. DePriest. (n.d.). *NMEA data* [Online]. Available: http://www.gpsinformation.org/dale/nmea.htm Accessed Apr. 5, 2016.

[30] Raspberry Pi Hardware. (n.d.). Raspberry Pi Foundation. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/. Accessed May 24, 2016.

[31] *Marine Corps Combat Marksmanship Programs*. Marine Corps Order 3574.2K, Commandant of the Marine Corps, Washington, DC, 2007, pp iii-vi.

[32] *Rifle Marksmanship*, Marine Corps Reference Publication 3–01A, USMC, 2012, Ch.1, pp. 1.

[33] *Rifle Marksmanship*, Marine Corps Reference Publication 3–01A, USMC, 2012, Ch. 10, pp. 3.

[34] *Rifle Marksmanship*, Marine Corps Reference Publication 3–01A, USMC, 2012, Ch. 2, pp. 1-4.

[35] A. E. Foushee, "Using posture estimation to enhance personal inertial tracking," M.S. thesis, Dept. Elect. & Comput. Eng., Naval Postgraduate School, Monterey, CA, 2016.

[36] D. Cross. (n.d.) *Azimuth/Distance calculator* [Online]. Available: http://cosinekitty.com/compass.html Accessed Apr. 5, 2016.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California