NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ROBOTIC NAVIGATION IN GPS-DENIED ENVIRONMENTS USING THE STRAPDOWN NAVIGATION ALGORITHM WITH ZERO-VELOCITY UPDATES

by

Samuel S. Druen

June 2020

Thesis Advisor: Xiaoping Yun
Second Reader: James Calusdian

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2020	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE ROBOTIC NAVIGATION IN GPS-DENIED ENVIRONMENTS USING THE STRAPDOWN NAVIGATION ALGORITHM WITH ZERO-VELOCITY UPDATES	**5. FUNDING NUMBERS**
6. AUTHOR(S) Samuel S. Druen	
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.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

GPS-denied environments, including indoor, urban canyon, and shipboard settings, present difficulties for autonomous robot navigation. One navigation solution in GPS-denied environments is to incorporate inertial sensors; however, due to sensor noise and calibration error, the accumulation of position error, or drift, causes the position estimate from inertial sensors to fail after a period of time. This thesis aimed to determine the viability of a pedestrian algorithm, which incorporates the zero-velocity update, to address the error and calculate distance traveled by a mobile robot in a GPS-denied environment. This work focused on indoor navigation using various sensors to provide data to the algorithm to calculate estimated distance traveled. Experiments were constructed and performed using a cart, robot, and mounted sensors in three laboratory settings: across the ground with preset distances, on an instrument rail track, and in an optical tracking environment. Tests conducted with the sensors determined that a system traveling above a minimum velocity threshold up to three meters can effectively implement a pedestrian tracking algorithm given known quaternion values. Adding a native means of determining system angles will allow this solution to be applied in more environments.

14. SUBJECT TERMS GPS-denied, indoor navigation, autonomous ground-based robot navigation, pedestrian-based navigation algorithms, zero-velocity update	**15. NUMBER OF PAGES** 101
	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.

ROBOTIC NAVIGATION IN GPS-DENIED ENVIRONMENTS USING THE STRAPDOWN NAVIGATION ALGORITHM WITH ZERO-VELOCITY UPDATES

Samuel S. Druen
Lieutenant, United States Navy
BS, Virginia Military Institute, 2015

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2020

Approved by: Xiaoping Yun
Advisor

James Calusdian
Second Reader

Douglas J. Fouts
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

GPS-denied environments, including indoor, urban canyon, and shipboard settings, present difficulties for autonomous robot navigation. One navigation solution in GPS-denied environments is to incorporate inertial sensors; however, due to sensor noise and calibration error, the accumulation of position error, or drift, causes the position estimate from inertial sensors to fail after a period of time. This thesis aimed to determine the viability of a pedestrian algorithm, which incorporates the zero-velocity update, to address the error and calculate distance traveled by a mobile robot in a GPS-denied environment. This work focused on indoor navigation using various sensors to provide data to the algorithm to calculate estimated distance traveled. Experiments were constructed and performed using a cart, robot, and mounted sensors in three laboratory settings: across the ground with preset distances, on an instrument rail track, and in an optical tracking environment. Tests conducted with the sensors determined that a system traveling above a minimum velocity threshold up to three meters can effectively implement a pedestrian tracking algorithm given known quaternion values. Adding a native means of determining system angles will allow this solution to be applied in more environments.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AI	Artificial Intelligence
DoD	Department of Defense
FQA	Factored Quaternion Algorithm
GUI	Graphical User Interface
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
GPS	Global Positioning System
NED	North-East-Down
ZUPT	Zero Velocity Update

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I wish to thank my Lord and savior for providing me with the chance to fulfill a dream and the support to get me through this journey.

I want to thank my wife, Kristin, for being by my side, supporting me through this entire process and the classes. You inspired me to pursue this journey and have helped me to remain steadfast through it all.

I want to thank my thesis advisors, Dr. Xiaoping Yun and Dr. James Calusdian. Dr. Yun, thank you for your invaluable advice and patience with me throughout this entire thesis writing process. Dr. Calusdian, thank you for your guidance in framing my focus and helping me through with discussions when I lost that focus.

To my church family and friends, thank you for giving me that extra level of support and an outlet when I needed it the most.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Current autonomous systems rely heavily on input from the Global Positioning System (GPS) to build their navigation control law due to its ease of use and relative accuracy when compared to other navigational systems. GPS provides an accurate means of fixing the position of an object within an accepted range of uncertainty. However, an overreliance on GPS can be exploited as a significant weakness during times of conflict when the system is operating in an environment where GPS signals are denied. GPS also has inherent limitations inside of buildings and within urban canyons. Inertial navigation systems (INS) are a current emphasis in autonomous navigation studies in conjunction with integrating accelerometer outputs to dead-reckon the system position. In [1], Yun et al. discuss various commercial solutions available on the market and issues that arise due to sensor errors within onboard systems. In this chapter, the motivation for this thesis, examples of similar solutions currently being studied, and the overall purpose and goal of this research are presented.

A. MOTIVATION

Autonomous systems are beneficial in military applications where risk to personnel is great. The use of an autonomous system relieves the demands on personnel by performing tasks normally done by humans. As the demand from the Department of Defense (DoD) for unmanned and autonomous systems increases, the environments in which these systems operate also broaden. These systems are increasingly required to operate in environments that are not conducive to the use of conventional GPS solutions. When operating in indoor environments, autonomous systems are unable to attain a reliable positional fix because the signal from GPS satellites is not strong enough [2]. Similarly, if an autonomous system is surrounded by metal surfaces in shipboard environment, GPS signals are useless. Some of the related solutions discussed in the next section are limited in these environments due to the presence of smoke, visible obstructions, or the amount of ferrous material surrounding the system. Therefore, a means of calculating position within

these environments must be addressed in order to expand the utility of autonomous systems within environments where GPS is denied or no longer a feasible solution.

B. RELATED SOLUTIONS

Some of the related solutions for this research are navigation algorithms utilizing vision, magnetic field, and inertial data to build a navigation solution. Each example will be discussed in this section.

1. Vision Solution

A solution for autonomous navigation utilizes cameras and visual image processing to create a visual navigation solution. Common implementations use optical sensors attached to the autonomous system and local positioning visual systems. In [3], Sazdivski et al. mount cameras on an autonomous system and utilize artificial intelligence (AI) to augment inertial navigation systems. AI was needed to provide a localization solution without a priori data [3]. In [4], Lategahn et al. utilize a camera mounted to a chassis and present a navigational solution with only one optical sensor. In [5], Zhou et al. present the idea of utilizing optical sensors as navigational anchors to create a local positioning system.

Optical or visual, systems are limited if the environment denies the utilization of optical sensors. If employing a local positioning system like in [5], the location must have sensors installed prior to the robot entering the location. Unless the system is being augmented with AI, systems with onboard optical systems need a means of localizing within their local environment, such as landmarks [4].

2. Magnetic Field

Another option is the use of magnetometer data and the local magnetic fields in order to calculate an angle of the system and thus be able to calculate the associated quaternion [1], [2], [6], and [7]. When navigating with a magnetometer, readings of the surrounding magnetic fields are used to calculate a heading or determine where the sensor is facing [7]. In [1] and [7], Yun et al. utilize the magnetic field reading to augment inertial navigation while a sensor is in angular motion. In [6], Calusdian et al. built upon the

research in [1] and [7] to better create an algorithm to handle the transfer from stationary to angular motion. In [6], the magnetometer readings utilize local declination to calculate angular motion. In [2], Storms et al. utilize the ambient magnetometer readings of a room in order to localize a robotic system. In order to achieve this, Storms et al. measured the magnetic field in a room and, based on the readings, the system matched its location to a magnetic field reading [2].

From [1], [2], [6], and [7], magnetic field navigation serves as a good alternative in conjunction with INS when GPS signals are not available. However, in a shipboard environment or when in close proximity to changing electronic systems, such as electric motors and generators, magnetic reference signals are significantly skewed.

3. Inertial Navigation

Inertial navigation utilizes acceleration measurements in order to calculate the distance traveled by integration. In the absence of GPS, this solution is the predominant method for navigation or is augmented by other sensors like in [1], [2], and [6]–[8]. Much of the focus in this field has been to utilize INS to aid in the tracking of pedestrian movements, as done in [1], [6], [7], and [8], in environments or situations where GPS no longer provides the needed fidelity to accurately track individuals. INS solutions can also be applied to robotic systems operating in GPS-denied environments. INS solutions provide acceleration data which can be integrated to provide positional data of the system. A major complication of inertial navigation, however, is sensor drift [6]. In [6], sensor drift and solutions to update the velocity in order to compensate for such drift are discussed.

C. PURPOSE AND GOAL

The purpose of this thesis is to confirm if a pedestrian tracking algorithm can be implemented on a wheeled robot chassis. The pedestrian tracking algorithm is built on research done in [6] by Calusdian et al. In [6], Calusdian et al. were able to design a control algorithm that was capable of accurately calculating the distance traveled by a pedestrian. To scope this study three goals were set: first, determine the distance a robot can travel before the data must be updated; second, determine the effects of velocity on the ability of

the algorithm to calculate distance; finally, determine if the algorithm can be implemented to an arbitrarily moving body. This study applies concepts discussed in [1] and [7] by Yun et al. and [6] by Calusdian, et al., which look into correcting the induced error in velocity by correcting the velocity to known values. The results are a simplified means for an autonomous system to navigate within the environment.

Chapter II presents the concepts that were utilized to govern this research. Chapter III addresses the hardware, software, and experiment procedures utilized to conduct this research. Chapter IV explores the results of the experiments. Finally, recommendations for future work and conclusions from this research are discussed in Chapter V.

II. BACKGROUND

In this chapter, a basic framework of the guiding concepts behind the thesis will be provided. More specifically, reference frames, quaternions, the Zero Velocity Update (ZUPT) algorithm, and the concept of interpolation encountered in this thesis will be discussed. In each section below, the concepts will be explained as each concept applies to this thesis.

A. REFERENCE FRAME

The reference frame is a coordinate system in which sensor measurements are represented. For this thesis, there are two reference frames, which are called the sensor, or body, reference frame and the navigational frame.

1. Sensor Reference Frame

The data collected from the sensors are represented in the sensor frame, which is a coordinate system attached to the sensor body. Therefore, the sensor collects all data with the sensor body as the center of motion. This also means that data collected is not associated with, and has no connection with, the surrounding environment. The sensor orientates itself with a conventional x, y, and z-axis, as seen in Figure 1.

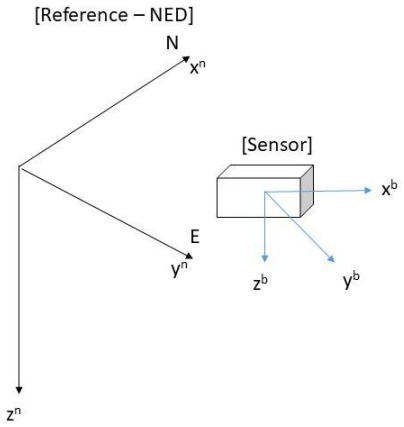

Figure 1. Sensor axis orientation

As presented in Figure 1, the axes are centered on the sensor body and are not referenced with any other point in space. In order to align the coordinate system of a body with the coordinate system associated with the sensor, the sensor must be rigidly attached to the body. This arrangement aligns the sensor body frame with the chassis body frame to allow both frames to be treated as the same. When the data from the sensor is transformed into the navigational frame, the transformation also applies to the chassis.

2. Navigational Reference Frame

For the navigational frame, the North-East-Down (NED) reference frame was selected. This gives an external reference point in order to orient the sensor reference body frame to the Earth. For NED, the x-axis is aligned to magnetic north, the y-axis is aligned with magnetic east, and the z-axis is aligned facing down toward the surface of the Earth. NED is commonly used in aeronautical applications, and it can also provide a common reference frame for several other applications. Using the magnetic poles of the Earth as known reference values, points can be utilized to transform the data from the arbitrary sensor frame to a reference frame that provides a context for navigation.

Since this experiment was limited to a small locality, no more than 10 meters at one time in any direction, the Earth can be assumed flat and conversions for an ellipsoidal or circular Earth are not needed.

B. QUATERNIONS

The method chosen to transform the data was to use quaternions. Similar to the work done by Calusdian et al. in [6], quaternions were utilized in order to lighten the computational load and avoid the use of matrices consisting of sinusoidal functions [6].

In order to calculate the transformation quaternion, the Factored Quaternion Algorithm (FQA) was employed as presented in [7] by Yun, et al. The algorithm takes the acceleration in the body frame, as well as data from the onboard magnetometer, and calculates the associated elevation, roll, and azimuth quaternions [6]. However, since the magnetometer would be utilized in conjunction with a robot chassis, the magnetic field calculations were omitted because of interference that would result from the presence of

the robot computer and the electric motors of the chassis. Therefore, in this particular utilization of the algorithm from [6], a vector of the form

$$a = [a_x^b, a_y^b, a_z^b] \tag{1}$$

was input into the function. The superscripts in (1) annotate that the accelerations are in the body frame prior to transformation into the navigation frame. From the input accelerations, the needed terms for the quaternion can be calculated. From [4], the elevation angle θ is related to the acceleration measurement as follows:

$$\begin{aligned} \sin\theta &= a_x \\ \cos\theta &= \sqrt{1-\sin^2\theta}. \end{aligned} \tag{2}$$

It is important to note that though the terms are notated as sinusoidal functions, all calculations are done via algebraic means, which does speed up the rate that the calculations are done. With the terms in (2) available, the half-angle representations are derived from [6] and calculated as follows:

$$\begin{aligned} \sin\frac{\theta}{2} &= sign(\sin\theta)\sqrt{(1-\cos\theta)/2} \\ \cos\frac{\theta}{2} &= \sqrt{(1+\cos\theta)/2}. \end{aligned} \tag{3}$$

With the half angles calculated, the elevation quaternion rotation operator q_e can be calculated as follows [6]:

$$q_e = \cos\frac{\theta}{2}(1,0,0,0) + \sin\frac{\theta}{2}(0,0,1,0). \tag{4}$$

With the elevation operator calculated, the focus can be turned to the roll quaternion. Similar to [6] and utilizing the terms from (2), the first terms for the roll quaternion can be calculated as follows:

$$\sin\varphi = \frac{-a_y}{\cos\theta}$$
$$\cos\varphi = \frac{-a_z}{\cos\theta}. \tag{5}$$

With these terms, a similar calculation as (3) can be done in order to get the half-angle terms of φ. With the half-angle terms available from (5), the roll quaternion can be calculated, like in [6], as such:

$$q_r = \cos\frac{\varphi}{2}(1,0,0,0) + \sin\frac{\varphi}{2}(0,1,0,0). \tag{6}$$

For the azimuth quaternion, since magnetometer readings were not utilized, the azimuth quaternion was hardcoded to [1, 0, 0, 0]. Even with the magnetometer data removed, the algorithm was able to estimate the quaternion based on the acceleration data in the sensor frame. Adapted from [6], the resulting calculated quaternion has the form

$$q = q_e * q_r. \tag{7}$$

Given the transformation quaternion, we can transform the acceleration from the body frame to the navigational frame through quaternion multiplication. To rotate the acceleration vector into the navigational frame, as done in [6], the vector is transformed using

$$a_n = q * a_b * q^*. \tag{8}$$

For the conjugate of the quaternion notated as q^* and presented in [6], it is calculated as such:

$$q^* = [q_0, -q_1, -q_2, -q_3]. \tag{9}$$

As stated above, the transformation is achieved by quaternion multiplication which is performed differently from typical multiplication [6]. If given two quaternions

$$p = \left[p_0, \vec{p} \right], \;\; q = \left[q_0, \vec{q} \right], \tag{10}$$

the quaternion multiplication is performed, presented in [6], as such:

$$p * q = p_0 q_0 - \vec{p}\vec{q} + p_0 \vec{q} + q_0 \vec{p} + \vec{p} \times \vec{q}. \tag{11}$$

The resulting quaternion is of the same form as the original vectors [6].

C. ZERO VELOCITY UPDATE

Similar to the method utilized in [6], the ZUPT algorithm was employed in this research. The ZUPT is used to update and correct for error within the measured acceleration data. From preliminary work, it was observed that the acceleration of the sensor will be zero both at the beginning, just prior to the onset of sensor motion, and immediately after the end of the motion when the sensor comes to rest. At both times, the velocity of the sensor will effectively be zero and the ZUPT algorithm can be employed as a means to correct for the error in the velocity calculations knowing that at both points the velocity should be zero.

As illustrated by Figure 2, there is a constant acceleration prior to the motion of the sensor. The constant acceleration was due to gravitational acceleration acting upon the sensor along the particular axis.

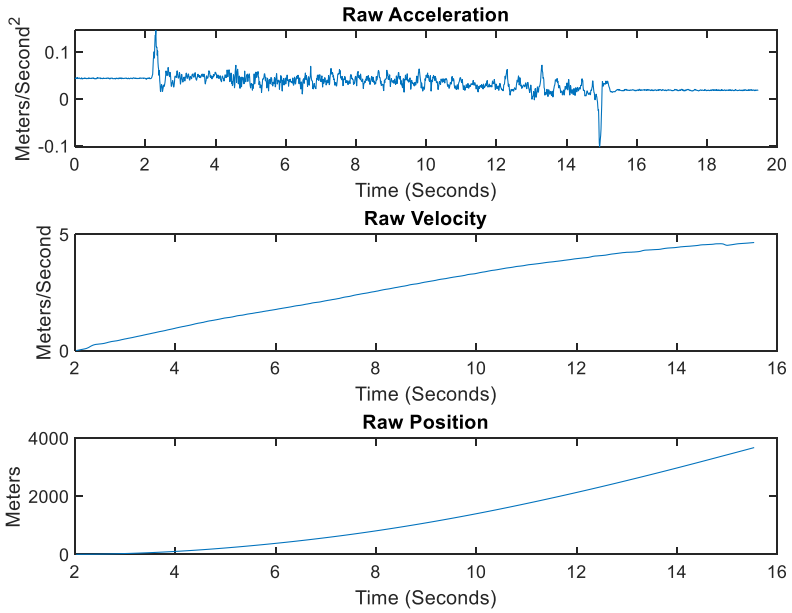

Figure 2. Acceleration data with uncorrected velocity and position

However, when utilizing direct integration of the data we can see in the second plot that the velocity goes on to infinity, bounded only by the fact that the data was integrated over the time of motion. This is further illustrated in the third plot where the calculated position is over 3000 meters for a motion that traveled only one meter.

In order to address this, we take our measured acceleration, which contains both the data α_a and an error term ε [6], and express it as follows:

$$\begin{aligned} \alpha &= \alpha_a(t) + \varepsilon \\ t &= [0:T]. \end{aligned} \tag{12}$$

The error term is what is compounded and propagated through the subsequent calculations. To alleviate the effects of the error, the ZUPT algorithm is used to correct the velocity in order to have a better position calculation.

To correct the propagated error from the acceleration, there must be known values in velocity in order to measure the effects of the error. For the ZUPT, the known values are

the starting and stopping points of motion, which have zero velocity. To correct propagated errors in the velocity, as presented in [6], the following is applied:

$$v_c(t) = v(t) - \frac{v(T)}{T}t, \quad t = [0, T]. \tag{13}$$

In the application, v_c is the corrected velocity, which is calculated by subtracting the error term over the entirety of the motion. For the application, the error term is the final velocity divided by the final time of motion. Correcting the error from the acceleration, from [6], gives us the following representation of velocity:

$$v_a(t) = v_c(t) + \varepsilon(t), \quad t = [0, T]. \tag{14}$$

Since we are still dealing with imperfect sensor data, there is an underlying error within the measurements that cannot be addressed that will still be made evident in the integration for the position. The result of the ZUPT can be seen in Figure 3.

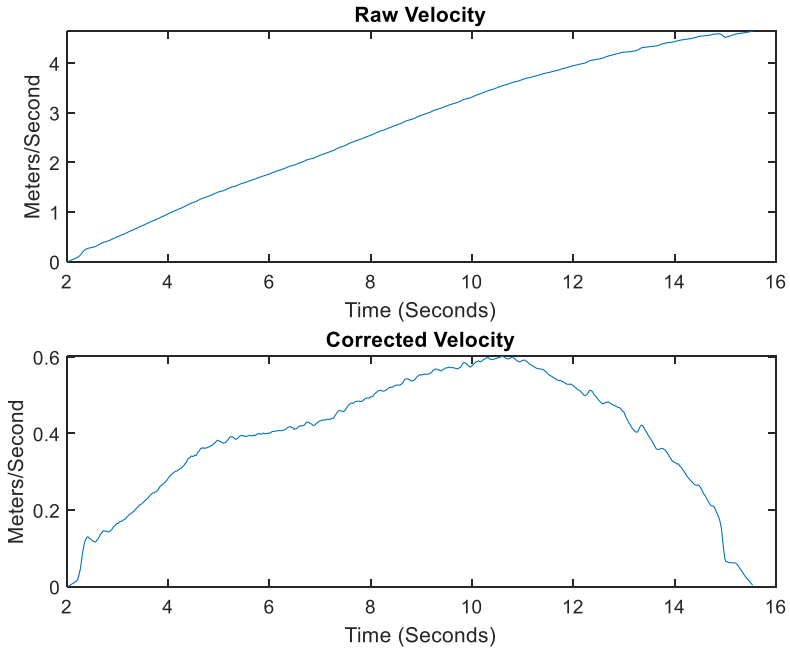

Figure 3. Uncorrected velocity with corrected velocity

From Figure 3, the original velocity increases linearly for a motion that starts and stops. In the second plot of Figure 3, the corrected velocity can be observed; we see that the start and stopping point are zero, and the increase in velocity and subsequent decrease can clearly be seen, as expected.

D. INTERPOLATION

During the course of the investigation, the use of multiple data collection platforms was needed. This creates the issue of a disconnect between the time steps. The disconnect in time steps cause data to no longer correlate directly to one another. In order to re-correlate the data sets, interpolation is necessitated. Interpolation is the calculation of data points between time steps that do not exist from the original set.

There are multiple means for data interpolation. The typical means of interpolation is linear interpolation. Linear interpolation assumes there is a linear relationship between two points, and all possible points exist along a line between those points.

In this chapter, the concepts that provide the framework upon which the procedures of this thesis were built were discussed. The concept of reference frames was described for both the sensor reference frame and the navigational reference frame. The transformation quaternion, FQA, and ZUPT in reference to their implementation within this thesis were discussed. Finally, data interpolation, which will be needed in the software implementation of the algorithm, was also discussed. In the next chapter, the procedures used in order to perform the experiments necessary to understand the effects of motion on a sensor will be discussed.

THIS PAGE INTENTIONALLY LEFT BLANK

III. PROCEDURES

The various hardware and software components used in the conduct of this research will be presented in this chapter. More specifically, the sensors that were utilized to collect the acceleration data in this thesis, the software that each sensor utilized to collect the data, and the different platforms used to perform the trials will be described.

A. SENSORS

In this study, three sensors were used: the Lord Microstrain 3DM-GX3-25, Lord Microstrain 3DM-GX4-25, and the Yost 3-space TSS-DL v 2.0. The two Lord Microstrain sensors represented high fidelity, specialized market options, and the Yost 3-space represented a more economical market option with lower sensor fidelity. These two categories of sensors were chosen in order to observe if sensor quality should be taken into account for a general navigational solution.

1. Lord Microstrain 3DM-GX3-25 and 3DM-GX4-25

The Lord Microstrain family of sensors is a set of industrial-grade sensors designed to provide a user with a range of sensor measurements along three axes of measurement and differing navigational solutions [9]. The “25” designation of the sensors among the other Lord Microstrain sensors delineate them as part of the Attitude Heading Reference System, or AHRS, product line [11]. For this study, the Lord Microstrain products that had GPS capabilities were not utilized in order to best simulate a GPS-denied environment.

Note: Dimensions used in the image are from [9].

Figure 4. Microstrain 3DM-GX3-25

Figure 4 is an example of the Microstrain 3DM-GX3-25 sensor utilized for experiments on the various platforms throughout the experiment.

Note: Dimensions used in the image are from [10].

Figure 5. Microstrain 3DM-GX4-25

Figure 5 is an example of the Microstrain 3DM-GX4-25 utilized for experiments. The Microstrain sensors, presented in Figures 4 and 5, contain nine sensors internally. In each device, there are three accelerometers, three magnetometers, and three angular rate sensors.

2. Yost 3-Space TSS-DL v 2.0

For comparison purposes, the Yost TSS-DL v2.0 3-space inertial measurement unit (IMU) sensor was also utilized because it is an economical sensor type. It was significant to verify that any data collected utilizing the Microstrain sensors could be replicated on a platform that was more readily available. For the data collection, the 3-Space software suite was utilized to collect the data, which provides a graphical user interface (GUI) that can be configured to collect four varying types of data required by the user. An example of the particular sensors used in this research is seen in Figure 6.

Note: Dimensions used in image are from [12].

Figure 6. Yost TSS-DL v2.0

B. SOFTWARE

In this study four software suites were utilized: MATLAB, 3DM Monitor, MIP Monitor, and Motive. Each will be covered in detail in this section.

1. MATLAB

For all of the experiments, the main computing environment used was MATLAB 2019b. MATLAB is a computing environment designed for engineering and scientific academic data processing [13]. MATLAB was employed in order to process the data collected during the trials above and manipulate and display the data in a way that was beneficial and could easily be visualized. The Mathematics and Graphics functionalities were the primary function utilized to process and visualize the data, both of which were built into MATLAB. All of the data collected via the varying collection software listed below was then imported into MATLAB for additional processing and analysis.

2. 3DM Monitor

For the Microstrain sensors, data collection software was provided by Microstrain in the included CD for the sensor. For the 3DM-3GX family of sensors, the corresponding data collection software is known as MIP 3DM Monitor. The monitor program has a GUI that the user can set in order to collect data via the preset data collection settings in the program, and outputs an Excel data file that can be read into MATLAB. For this research, acceleration and angular acceleration were collected. This was a pre-programed feature of the GUI, and therefore accelerometer data could not be exclusively collected.

3. MIP Monitor

Similar to the above section, the 3DM-GX4-25 sensor also came with a data collection software, the MIP Monitor. Again, MIP Monitor is a GUI that the user can set to collect specific data points based on preset data collection software built into the GUI. As before, the software outputs an Excel file that can be read into MATLAB.

4. Motive Tracker

The software suite utilized by Optitrack was Motive tacker, which is a six-degrees-of-freedom tracking software [14]. The system can be utilized to provide data streaming and integration with processing software such as MATLAB [14]. For the experiments, data was streamed into MATLAB where the associated x, y, and z-axis positions and Euler angles were tracked for the sensor during motion.

C. CART

In order to analyze general motion with the sensors, the behavior of the sensor was observed on a wheeled platform and eventually applying the principles to a wheeled ground-based robot. For the first rounds of experiments, a laboratory cart with attached sensors was utilized as the initial platform. The sensor was affixed to the leading edge of the cart in order to better model the true distance traveled by the sensor. With the sensor attached to the cart, the acceleration was measured on a five-meter course, shown in Figure 7.

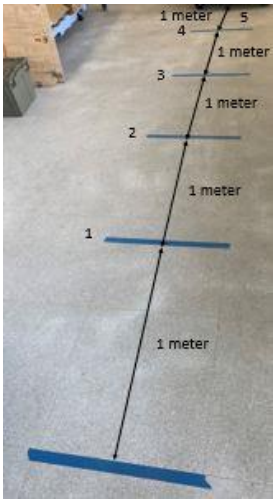

Figure 7. Five-meter track for experiments

Each piece of masking tape was placed at a one-meter interval from zero to five meters. In order to ensure the accuracy of the experiments, the leading edge of the cart was aligned with the front edge of the masking tape at the beginning of each trial. At the end of

each trial after the desired distance had been traversed, a stopping device was placed to ensure that the cart stopped consistently at the appropriate location. With the sensor monitoring systems running, the velocities were varied to observe if the sensor performed better at a particular velocity range. The sensor was also tested on a 10-meter track with distances of one, two, three, five and 10 meters to verify that the values measured on the five-meter track matched the data measured on the 10-meter track. The distances were selected in order to observe at what point the IMU with the ZUPT algorithm could accurately estimate the distance traveled.

To process the data collected, the data was imported into MATLAB. With the data plotted in MATLAB, the start and stop points of the data were selected.

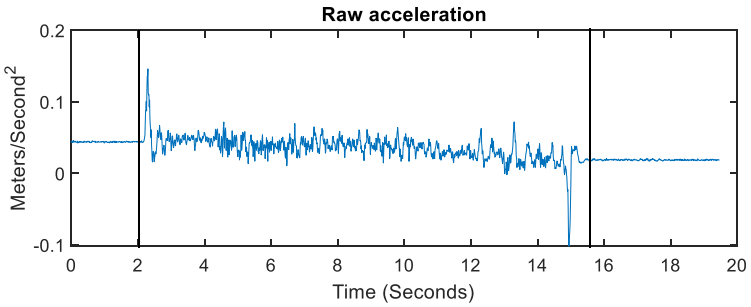

Figure 8. Motion start and stop

An example of acceleration data is shown in Figure 8. The black vertical marker lines show an example of where the start and stop points of the motion were manually selected. After selecting the start and stop points in the data, the trapezoidal integration function *cumtrapz* in MATLAB [15] and the ZUPT were utilized.

D. PIONEER P3-DX ROBOT

The next component of the test was to observe the sensor behavior with a robotic platform. The platform utilized was the Pioneer P3-DX indoor robot chassis. This platform was utilized within the lab for previous work and was selected to maintain commonality with potential future works.

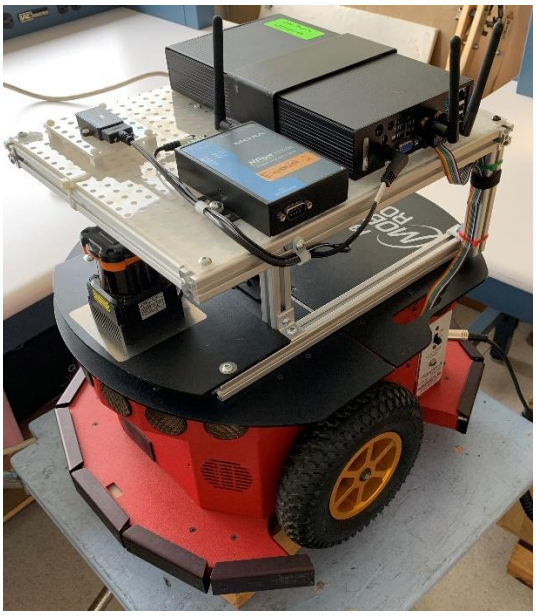

Figure 9. Pioneer P3-DX

The robot in Figure 9 was tested on the original five-meter track, as seen in Figure 7, and the 10-meter track similar to experiments done with the cart. As the robotic platform did not have a control algorithm yet created for these experiments, a tethered controller was utilized to control the robot based on six speed settings seen in Figure 10.

Figure 10. Tethered controller

Six-speed settings were selected that represented speeds that a robot operating autonomously will be able to achieve. This experiment also served to observe the differences between motion of a lab cart and that of the wheeled robot platform. If the motion was similar enough, then further testing could be achieved without the use of a robot platform. The same sensor utilized on the cart was mounted on the robot chassis and performed a similar battery of tests as the cart. The same sensor was used to further compare the performance between the cart and the robot. The same process was used to process the data from the robot tests as was used to process the data from the cart experiment.

E. DATA OPTICS, INC. OPTICS RAIL

In order to do an in-depth study of the sensor to verify if a pedestrian position estimation algorithm can accurately estimate position in a robotic realm, the sensor was attached to a stationary platform. To remove as much noise as possible from the motion, a Data Optics, Inc. optical rail was used. The specific rail is shown in Figure 11.

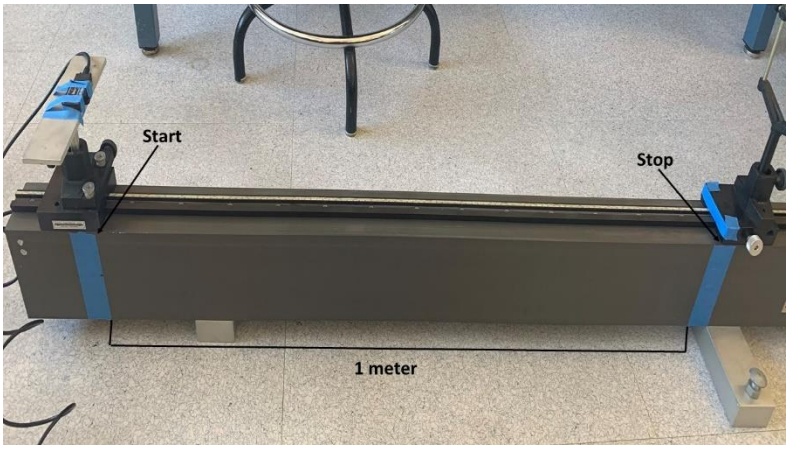

Figure 11. Optical rail

The blue stripping is a one-meter measurement. The leading edge of one strip and the tailing edge of the second strip correspond to the start and stop position. The sensor was affixed to a shelf that was inserted into an optical tool sled seen in Figure 12.

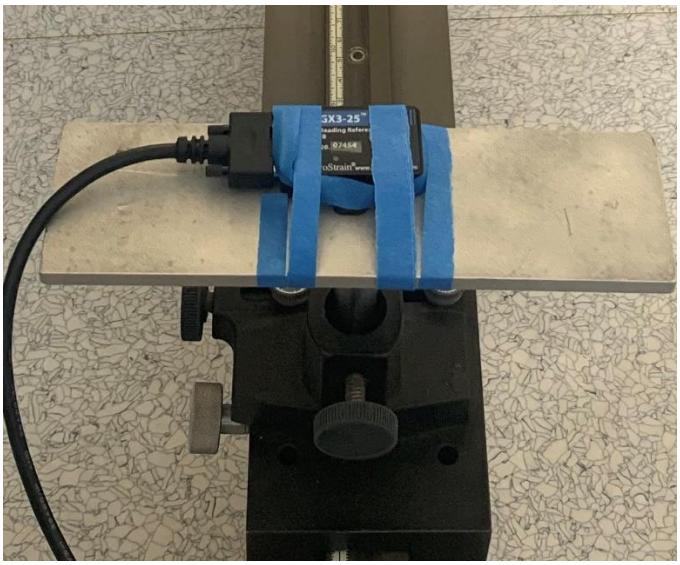

Figure 12. Optical tool sled

The rail was lubricated in order to smooth out and remove any noise from the motion of the sled. The optical rail was confirmed to be horizontal using a carpenter level available in the laboratory. The motion was measured at five approximate velocities in order to observe the effect of velocity in a controlled testing environment. The first rounds of testing were conducted with the sensor on the shelf shown, in Figure 12, with the y-axis corresponding to the axis of motion. The sensor was to remain flat and the only variable to be measured was the forward acceleration in the axis of motion.

Following the single axis of motion, testing began with the sensor attached to a shelf that had an angle indication in order to excite an additional factor in the motion. The sensor was tested inclined at a fixed angle starting from plus and minus four degrees to plus and minus 90 degrees in order to observe if the FQA could accurately calculate the fixed angle prior to the beginning of motion; the angle was not changed during motion for this battery of experiments. The sensor was mounted with either the x or y-axis as the axis of motion, and the motion was varied between three approximate velocities. The variable angle sled is shown in Figure 13.

Figure 13. Shelf attached to angle indicator

In order to gauge at what angle the sensor was being placed, the sled also had an angular reference face seen in Figure 14.

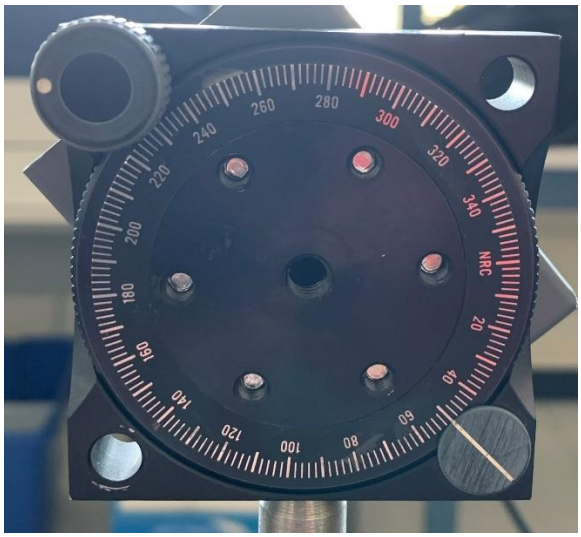

Figure 14. Angular reference face

The mounting plate was not centered with the zero point of the angular face because the shelf does not rest at zero; instead, it rests at ten degrees. Also, since the shelf is off-centered, there was a slight angle added in different axes.

Before testing was moved into an optical tracking environment, the sensor was placed at a measured angle of approximately 40 degrees and varied the angle through the sensor motion on the rail. As before, the velocity was varied during the motion between three approximate velocities; the angular position was also changed.

For processing the optical rail data, the FQA was implemented in MATLAB in order to address the addition of an arbitrary angle to the sensor. A point was selected in the data prior to the start of motion in order for the FQA to calculate a steady-state set of quaternions to transform the data from the sensor frame to the navigational frame. For the variable angle tests, the FQA was utilized on every data point along the motion of the sensor. Similar to the experiments before, the start and stop points of the motion were selected once the data had been transformed and again trapezoidal integration was utilized within MATLAB.

F. OPTITRACK PRIME OPTICAL TRACKING SUITE

In order to handle arbitrary motion within a space, the OptiTrack system was utilized in order to calculate true quaternions in space. The optical sensors provide an optical solution to calculate the sensor angle, as shown in Figure 15.

Figure 15. OptiTrack optical sensors

The sensor was affixed to a wooden block with reflective orbs that the system utilized to calculate the sensor position in space via optical cameras. An example of this sled can be seen in Figure 16.

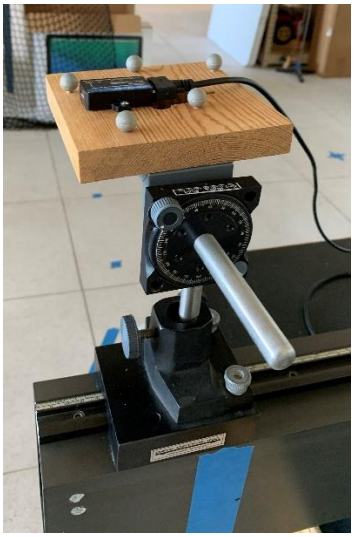

Figure 16. OptiTrack sled

Initially, the sensor remained on the Optical rail to verify that the system performed as expected. Adding in an additional data collection method also meant integrating two sets of data collected on two different machines. For the OptiTrack data, the system ran through a desktop computer, and the IMU data was collected via a laptop computer. The time stamp

for the IMU could be zeroed and started at true zero via the Microstrain data collecting software. The OptiTrack data, however, clocked off of the desktop computer, and due to system sampling rates, could sometimes create redundant data. In order to address this, each trial began with a small abrupt motion that placed the sensor at the needed starting point to impart an impulse in the data. This could easily be seen in both the IMU data and the OptiTrack data. The point at which the impulse feature was found in both data sets was made the new zero reference for both data sets. Then the OptiTrack data could be filtered to remove any redundant data and the IMU data could be fit to match the corresponding data points. Utilizing the interpolation function interp1 within MATLAB [16], the values between the data points in the IMU data were interpolated to corresponding values from the OptiTrack data. Once the data provided by the OptiTrack system was verified and could be utilized as a known true value, tests with variable angles could be done again on the sled. Again, the data covered three approximate velocities, varied over multiple runs.

Following the variable angle test, testing moved into estimation of arbitrary motion. For this test, the sensor was removed from the rail and simply placed on a table. Then the sensor and wooden platform were picked up and arbitrarily moved through space approximately one meter. As before, tape was placed on the table at one meter in order to give a starting and stopping reference point. To examine the effects of arbitrary motion, three types of motion were used: an arcing motion, a wobbling motion, and a rocking motion. An example of each motion is presented in Figure 17.

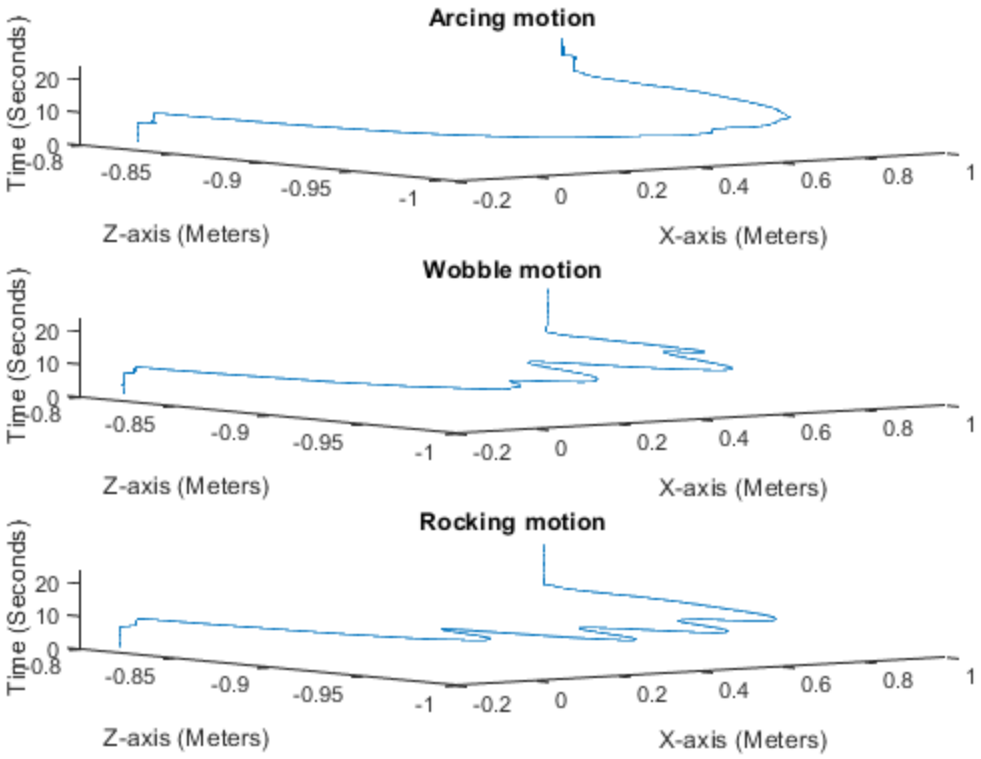

Figure 17. Plots of arbitrary motion

For the arcing motion, the sensor was picked up and moved in an arc from the start to the stop location on the table. For the wobble motion, the sensor was picked up and given a corkscrew-like arc motion from start to stop. For the rocking motion, the sensor was given a rocking action around the x-axis as the sensor was moved in an arc from the start to stop location.

The hardware and software used in this thesis were discussed in this chapter. As well, the procedures utilized in performing each experiment were outlined in this chapter. In the next chapter the results of the experiments conducted will be analyzed and discussed.

IV. RESULTS

In this chapter, the results from each series of experiments with the cart, robot, rail system, and OptiTrack system are presented. The procedures that were utilized to perform each experiment were outlined in Chapter III. All of the results for each experiment are grouped by the respective experiment.

A. CART EXPERIMENT

Work in [6] by Calusdian et al. and [7] by Yun et al. was done on pedestrian motion, however, there is a difference in motion between pedestrian and wheeled movement. The significant difference between a pedestrian step motion and a wheeled chassis motion is observed in Figure 18.

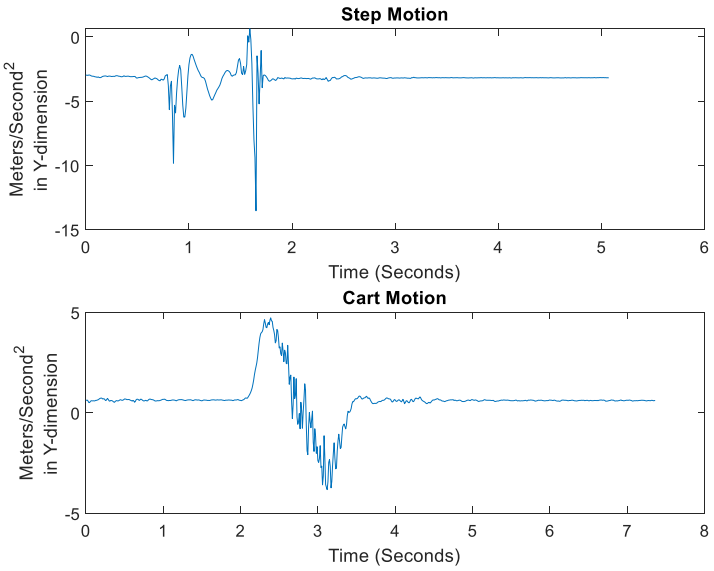

Figure 18. Step motion versus cart motion

The top plot in Figure 18 is an example of the stepping motion, and the bottom is an example of the wheeled motion, both of which start and stop at zero. However, the waveform of the step motion varies significantly than that of the smoother motion of the

cart. The experiment was conducted on the five-meter course within the laboratory environment. The results of the cart experiments are shown in Table 1 with the calculated distance with and without the ZUPT.

Table 1. Calculated versus traveled distance

Distance traveled	Calculated distance with ZUPT	Calculated distance without ZUPT
1 meter	1.0380 meters	1.6222 meters
2 meters	2.0295 meters	3.6027 meters
3 meters	3.0091 meters	7.4483 meters
5 meters	4.9878 meters	16.4793 meters

Columns two and three of Table 1 represent the same data points in order to keep continuity in the results presented. Table 1 is a summary of the results that were closest to the distance traveled by the sensor. Of the 85 trails conducted, trials that were performed at a higher velocity had the lowest error both with and without the ZUPT. In order to determine if this was a coincidence, the motion was plotted to observe what each waveform looked like. An example of the acceleration data from a lower velocity trial and that of a higher velocity trial can be seen in Figure 19.

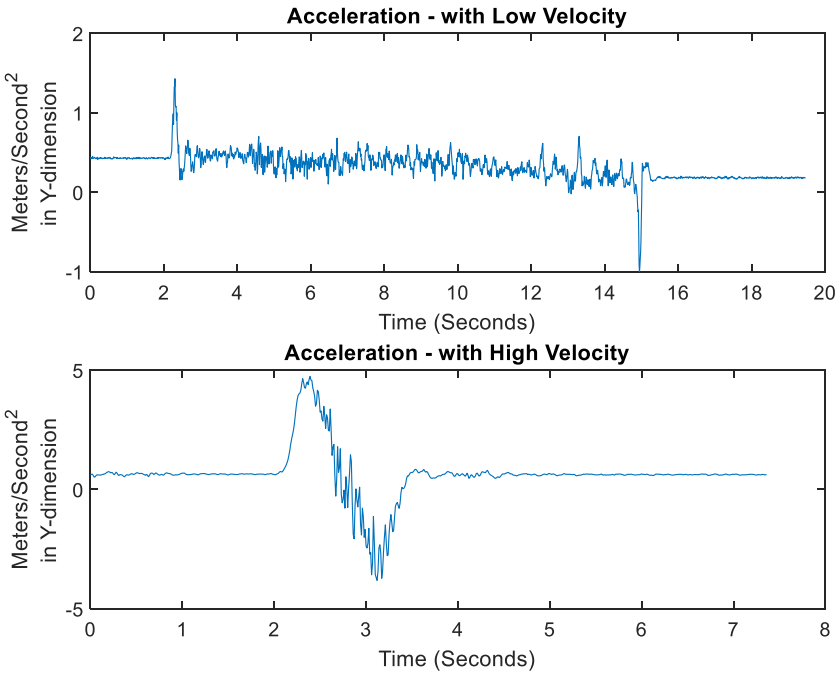

Figure 19. Acceleration plot of low and high-velocity trials

In Figure 19, the data in the lower velocity trial remains close to the point of rest and requires a more substantial amount of time to travel the same amount of distance. The lower acceleration of the slower trial allowed for sensor noise to have a greater effect on the data. For the higher velocity data in Figure 19, the shape of the data is much more distinct. The distance of the peak of motion from the sensor noise minimized the effects of noise in the sample. In order to observe the variability of the data overall, the standard deviation is presented in Table 2.

Table 2. Standard deviations versus distance traveled

Distance traveled	Standard deviation with ZUPT	Standard deviation without ZUPT
1 meter	.1096 meters	9.2912 meters
2 meters	.2085 meters	21.2783 meters
3 meters	.4196 meters	32.7379 meters
5 meters	1.2294 meters	98.0281 meters

The standard deviation covers the entire dataset and is not grouped by velocities. This creates a wider deviation size as the distance increases. From the data in Table 1, three meters was the distance that was calculated by the algorithm accurately. The difference between the distance traveled and the distance calculated was at most 0.03 meters. In conjunction with the standard deviation, as the distance increased, so does the variation between samples. As the distance increased, the varying of the velocity had a greater effect.

This portion of the study helped to analyze and understand the effect the ZUPT had on data collected by a sensor on a wheeled platform. The tests provided data that was used to verify that the algorithm was performing as expected for similar platforms. However, the true analysis began with data collection on a robot chassis.

B. ROBOT EXPERIMENT

For the robot chassis, three major questions needed to be addressed. First, does the distance traveled before an update is performed affect the ability of the algorithm to calculate the distance traveled? Second, does the velocity of the robot affect the ability of the algorithms to calculate the distance travel? And finally, can the pedestrian algorithms presented in [5], [4] and [7], be utilized with a robotic system?

1. Effect of Distance

To answer the first question, tests similar to those done with the cart for distances of one, two, three, four, five, and ten meters were performed. From the cart data, a distance

of interest was three meters. However, the question of what distance a robotic system travels before a reference update needs to be done has to be addressed.

The robot was moved from one to five meters to ensure that the effects over the distance were clearly observed. A trial at 10-meters was also conducted to observe if a significant jump in distance created any benefit compared to the other distances. Displayed in Figure 20 is a summary of the results from the different distances.

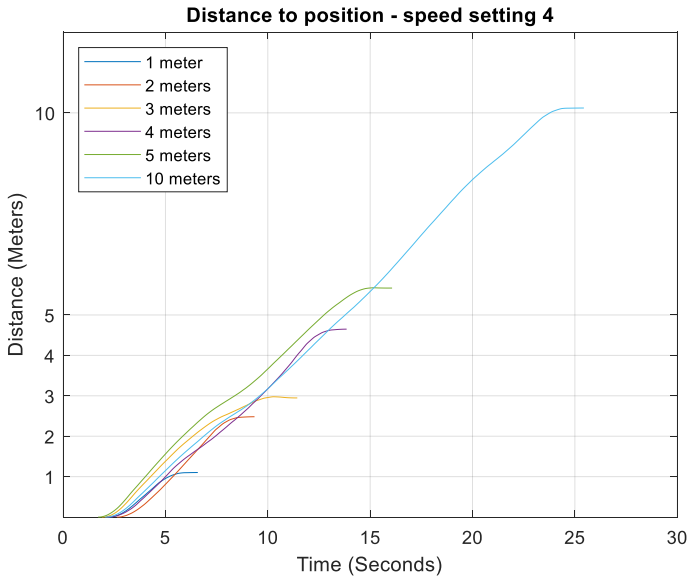

Figure 20. Calculated distances traveled

Three meters is shown to be the furthest distance traveled that was consistently calculated with the robot and ZUPT of the data in Figure 20, which corroborates what was observed in the cart trial. The total data from the distance trials is summarized in Table 3.

Table 3. Distance and calculated distance with standard deviation

Distance Traveled	Calculated Distance Without ZUPT	Calculated Distance With ZUPT	Standard Deviation
1 meter	3.7473 meters	1.1022 meters	.8390 meters
2 meters	7.8690 meters	2.4823 meters	2.5034 meters
3 meters	17.3953 meters	2.9476 meters	1.6306 meters
4 meters	16.688 meters	4.6492 meters	2.9063 meters
5 meters	37.5132 meters	5.6655 meters	10.7142 meters
10 meters	91.8326 meters	10.1201 meters	12.8846 meters

Aside from the one-meter trials, the three-meter trials had the lowest deviation within the total dataset and was the closest overall for all the other distance trials, as shown in Table 3. Again, similar to the cart data, the standard deviation was over the entire dataset of at least 19 trials each. Data was not differentiated between the different speed settings and therefore presented a wider range of results, especially at the farther distances.

2. Effect of Velocity

For the effects of velocity, the system was tested with six varied speed settings for the robot chassis controlled with a tethered controller. For the purpose of the trials, the exact speed at each setting was not calculated, but the settings were marked in order to ensure that the same velocity was achieved for each test. From the cart experiments there had been indications that at higher speeds the algorithm was able to better calculate the overall distance traveled.

From the trials it became clear that at lower velocities integration with only the ZUPT created an issue with calculating the distance traveled, as was similarly observed in the cart experiments. Therefore, towards the end of the trails with the robot system, the

lowest velocity setting was removed as it was not providing any beneficial data towards the overall experiment. The results from the speed trials are displayed in Figure 21.

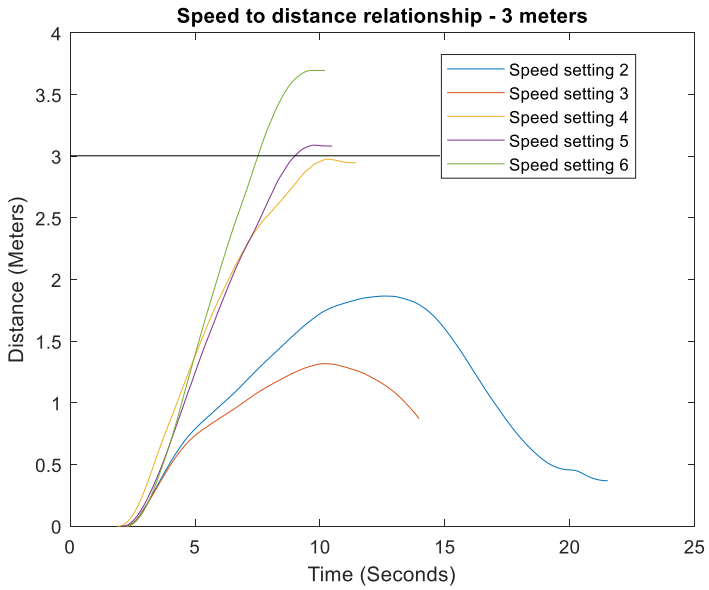

Figure 21. Speed settings and corresponding distance

The black horizontal marker line in Figure 21 is the expected distance to be calculated and summarized with the best datasets from the various runs. From the plot, the optimal speed settings are settings three and four for the robot chassis.

From the cart experiment, the optimum velocity needs to be the highest possible to provide the most accurate distance calculations. However, as presented in Figure 21, the highest velocity was not the most accurate trial. As the velocity increased, the chassis had more settling as it slowed. An example of the settling is observed in Figure 22.

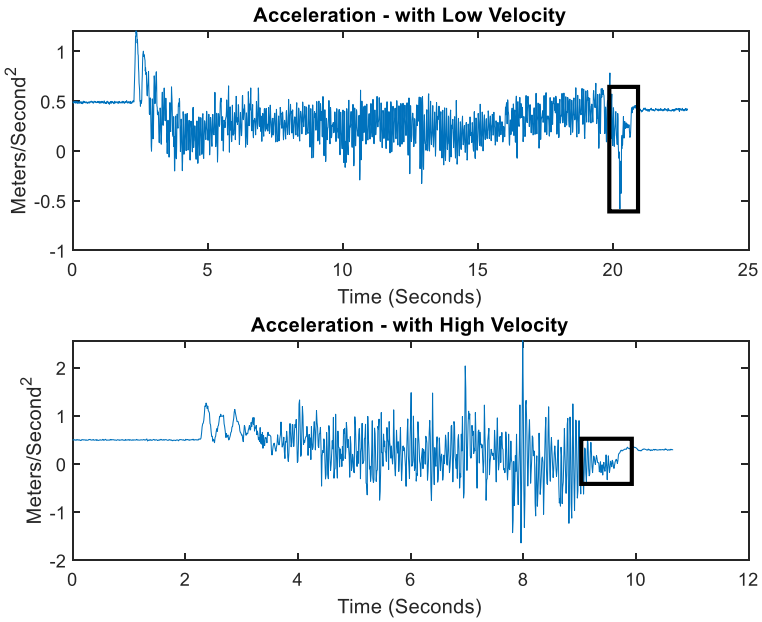

Figure 22. Low-velocity and high-velocity trials

The boxes in Figure 22 are examples of the settling in the mechanical system as the robot stopped its motion. This region added further error on top of the already present drift and sensor noise. The higher in velocity the robot went the larger this region was, causing a peak in velocity performance.

3. Viability of Pedestrian Algorithm

The trials described above demonstrate that a pedestrian footstep algorithm can be utilized to track the distance traveled by a robot. From the distance trials, it was observed that a robot can travel three meters before an update is needed. Three meters presents a maximum distance that a system can travel without the aid of external data sources. As shown in the velocity trials, the robot needs to be operating within a velocity range that is achievable by the robot, and above a minimum speed threshold, but not operate at such a velocity that it creates an uncontrollable condition. From the experiments above and the results from [5], [6], and [7], the described algorithms from this and previous works can be applied to an autonomous robotic system. To investigate this assumption further, a more

in-depth study was performed on the sensor ability to calculate the distance traveled on a controlled rail system as described in Chapter III section E.

C. RAIL EXPERIMENT

For the rail, the ability of the algorithm to calculate the distance traveled with three types of motion was observed. To address the anticipated motion of a robotic platform, the sensor was tested with flat linear motion, linear motion at a set angle, and finally linear motion with a varied angle.

1. Flat Linear Motion

To verify the data presented, the sensor was placed on a flat tool sled. As tested before, the velocity of the sensor was varied on the sled in order to get a full range of motion. Other sensors were tested to confirm that results were consistent across multiple platforms. The results are summarized in Table 4.

Table 4. Calculated distances versus various sensors

Position calculated via double integration (without ZUPT)	Position calculated via double integration(with ZUPT)
GX3-25	
-1.7469 meters	0.9992 meters
-0.1263 meters	1.0101 meters
0.7983 meters	1.0052 meters
0.944 meters	0.9997 meters
0.9666 meters	1.0075 meters
0.9754 meters	1.0004 meters
0.9889 meters	0.9992 meters
GX4-25	
-17.0895 meters	2.9726 meters
-2.4972 meters	1.1265 meters

Position calculated via double integration (without ZUPT)	Position calculated via double integration(with ZUPT)
-0.0471 meters	1.0614 meters
0.2477 meters	1.082 meters
1.0358 meters	1.0481 meters
Yost 3-Space	
9.5459 meters	1.0819 meters
3.4979 meters	1.1248 meters
1.6166 meters	0.9752 meters
1.325 meters	1.0033 meters
1.2119 meters	0.9779 meters
1.1547 meters	1.0048 meters

As presented in Table 4, all three sensors performed within a close range of each other. Interestingly, both high-end sensors and the commercial sensor all performed with the same relative results. There was an issue with the data for the Microstrain GX4-25 sensor. There was no timestamp from the data collection software, and an arbitrary time stamp had to be created based on the sampling rate of the sensor. Because the data had no time reference, the GX4-25 was not used for follow on trials. As presented in Table 4, all of the sensors performed adequately during the one-meter trials. The data from the flat linear motion trial closely matched results from the robot and cart trials. This validated the optical rail as a testing platform for further experiments. The next phase of testing was the addition of an angle other than zero, and the addition of the FQA to the algorithm.

2. Linear Motion with a Set Angle

The next step in the analysis was to set a non-zero angle through the full duration of motion. This also integrated the use of the FQA to calculate the angle at which the sensor is placed. A summary of the angles tested is shown in Table 5.

Table 5. Reference angle and calculated angle

Reference Angle	Calculated Angle
±4°	-4.686°, 4.383°
±6°	-6.932°, 6.71°
±10°	-11.193°, 9.163°
±20°	-20.754°, 20.494°
±30°	-29.57°, 31.064°
±40°	-40.015°, 41.226°
±50°	-50.975°, 50.679°
±60°	-59.461°, 60.705°
±70°	-69.687°, 69.537°
±80°	-79.605°, 80.749°
±90°	-90.743°, 89.391°

The results of the angles calculated by the FQA are presented in Table 5. There was some variability, however. The angle the sled was set was more of a reference point than an exact measurement. Even with each of the degree changes, the algorithm was still able to accurately calculate the distance traveled. The results are summarized in Table 6.

Table 6. Reference angle with calculated distance and standard deviation

Angle	Calculated distance	Standard deviation
±4°	1.0050 meters	.4095 meters
±6°	1.0000 meter	.3337 meters
±10°	1.0023 meters	.2112 meters
±20°	1.0032 meters	.2516 meters
±30°	1.0043 meters	.2737 meters
±40°	1.0003 meters	.2730 meters

Angle	Calculated distance	Standard deviation
±50°	1.0004 meters	.1936 meters
±60°	1.0052 meters	.2836 meters
±70°	1.0098 meters	2.5039 meters
±80°	0.9990 meters	.1360 meters
±90°	0.9991 meters	.3639 meters

From Table 6, the data was all tightly spread and well within expected results. The only outlier was in the 70-degree trials; one trial resulted in a distance calculation of 16 meters. The data was run multiple times, and the outlier could only be attributed to a false reading, without the outlier, the standard deviation was 0.2214 meters similar to the other results. With the data corroborating previously observed results, trials on the effects of variable angles were conducted.

3. Linear Motion with a Variable Angle

The previous trials verified that with the FQA, a sensor, and ZUPT, a system can accurately calculate the distance traveled. However, a robotic system is not consistently moving completely flat or at a fixed angle. The motion of the chassis introduces variation in the angles read by a sensor. Therefore, a varied angle was introduced during the motion of the sensor with the FQA calculating the transformation quaternion continuously. The results are summarized in Table 7.

Table 7. Reference angles and calculated distance and standard deviation

Starting Angle	Stop Angle	Calculated Distance	Standard Deviation
[40°, 0, 0]	[30°, 0, 0]	2.8298 meters	55.85521 meters
[0, -40°, 0]	[0, -32°, 0]	3.036 meters	30.50226 meters
[30°, 0, 0]	[20°, 0, 0]	4.8796 meters	4.8796 meters

The calculated distances and the reference distances in Table 7 differed significantly when compared to previous trials. The results also had a significant deviation between trials as presented in Table 7. The results in Table 7 were verified by running each test a second time; all results behaved similarly. Angle variability has a significant effect on the ability of the algorithm to calculate traveled distance in the absence of an accurate quaternion.

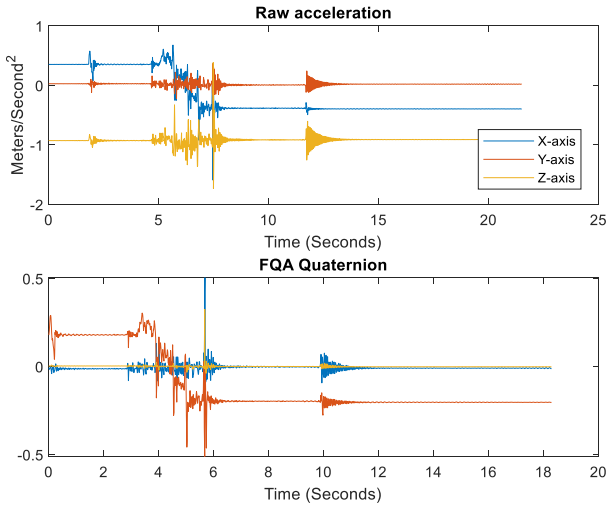

Figure 23. Acceleration data and corresponding quaternion

An example of the accelerometer data and the corresponding quaternion calculated, during these runs is shown in Figure 23. The FQA cannot accurately calculate the quaternion and simply mirrors the motion. The accelerometer data is not properly transformed when processing due to the inaccuracies in the quaternion. This was a verification of results observed in [5], [6], and [7], and why implementation of a complementary filter was utilized in [6] by Calusdian et al. However, the complementary filter in [6] utilized magnetometer readings from the sensor, which was not being used in this work. The sensor utilized was on a robot with a computer and electric motors. Data from this sensor did not provide an accurate enough angular reading as was done in [1], and [6]. The effect of a computer on a magnetometer is illustrated in Figure 24.

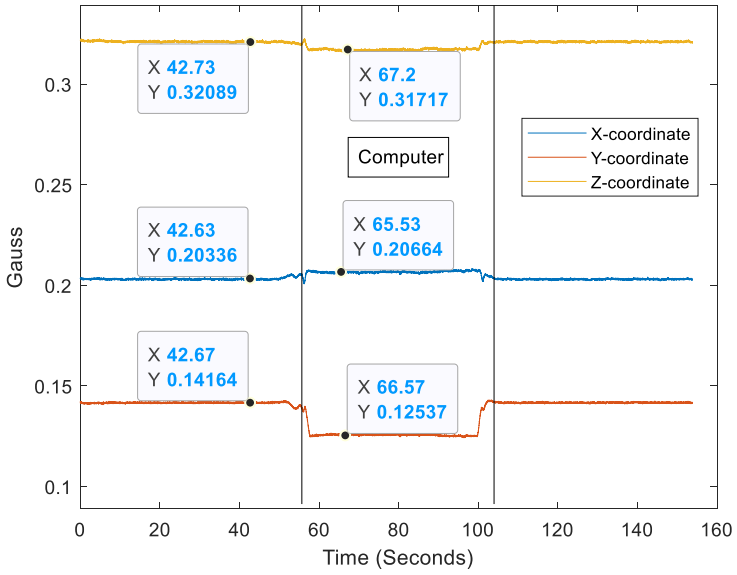

Figure 24. Effects of computer on magnetometer

Although the values appear to be minimal, the effects illustrated were when a laptop computer came within two feet of the sensor. As the computer was brought closer the effects were amplified.

In order to investigate further, the experiments were moved within an optical tracking environment to have the quaternions calculated through the tracking software. If the distance can still be accurately calculated with a known quaternion, then quality quaternion values are needed, and pedestrian tracking algorithms can be utilized for robotic systems.

D. OPTITRACK EXPERIMENT

The OptiTrack system, available in the laboratory, provides a way to calculate an accurate quaternion for any point of motion for the sensor. However, the OptiTrack system had to be verified to ensure that it produced expected values. As done previously, the system was tested with set angles, progressing to variable angles, and once satisfied that the system was performing accurately, to arbitrary motion.

1. Linear Motion with a Set Angle

Similar to the set angle trials ran in Section C of this chapter, a sled was set to an angle and verified that the observed angle from the optical sensor was a reasonable value. The results of the angles were not collected because they were a reference. The ability of the algorithm to calculate distance was already verified in the linear motion trials. The system was able to observe the angle accurately, and the results of the trials are summarized in Table 8.

Table 8. Calculated distances with standard deviation.

Calculated Distance With ZUPT	Calculated Distance Without ZUPT	Standard Deviation
1.1674 meters	79.4381 meters	0.2455 meters

Presented in Table 8, the distance the algorithm was able to accurately calculate was similar to the results observed in the optical rail experiments. The variation between the data was acceptable to go forward with the variable angle trials.

2. Linear Motion with a Variable Angle

Similar to the trials done with the set angle, the initial angle that the sensor was set to was verified. This was compared to what was observed by the optical system. Trials were completed with the angle varied during the motion. The results are summarized in Table 9.

Table 9. Calculated distances and standard deviation.

Calculated Distance With ZUPT	Calculated Distance Without ZUPT	Standard Deviation
1.0292 meters	28.3039 meters	0.6551 meters

The data proved that with an accurate quaternion the algorithm was able to accurately calculate the distance traveled by the sensor, presented in Table 9. In order to verify that it was an issue with the quaternion, the two quaternions were plotted.

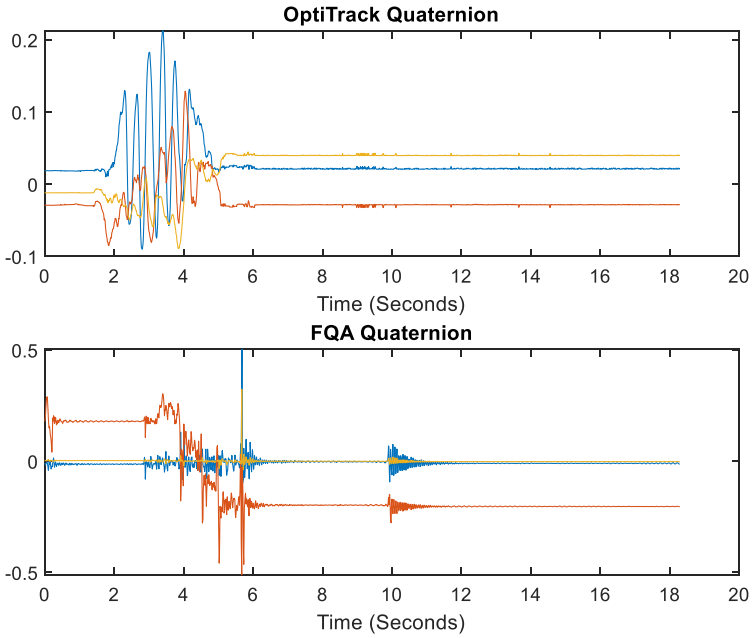

Figure 25. OptiTrack quaternion versus FQA quaternion

The quaternion calculated from the OptiTrack data and the quaternion calculated by the FQA function are shown in Figure 25 for comparison. There is a significant difference between the two quaternions. However, if the quaternion is still incorrect, the acceleration in the sensor frame will not be properly transformed into the navigational frame.

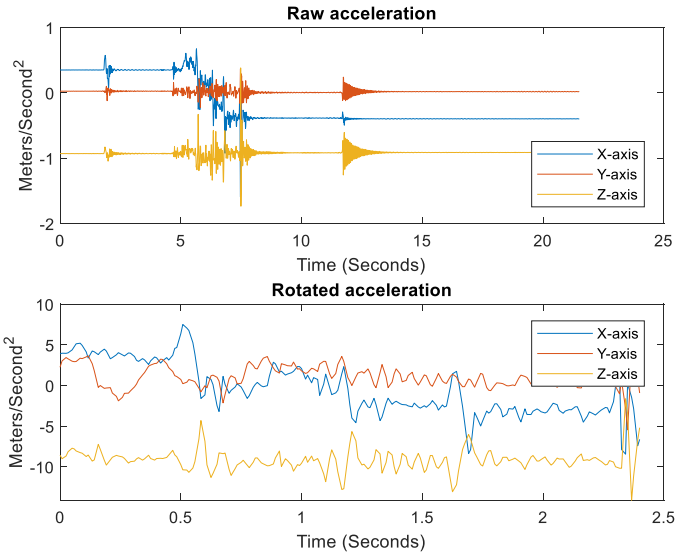

Figure 26. Raw and transformed acceleration

The top plot in Figure 26 is the raw acceleration data, and the bottom plot in Figure 26 is the acceleration data rotated into the navigational frame. The transformed acceleration does follow the expected behavior of the system. Since the x-axis was the main axis of the motion, acceleration along the y-axis was around 0 m/s^2 as expected. The z-axis corresponded to the axis of gravity and therefore was expected to be at approximately -9.8 m/s^2. This observation further clarified why the calculated distances in Table 7 varied so much compared to other results. Once the algorithm was observed to accurately calculate distance with a variable angle, arbitrary motion was tested.

3. Arbitrary Motion

For the arbitrary motion, three main types of motion—arc, wobble, and rocking motion—were tested in order to model some motion that would be expected by a robotic system. For each motion the results are summarized in Table 10.

Table 10. Sensor motion with calculated distance and standard deviation.

Motion	Calculated Distance	Standard Deviation
Arc	1.0574 m	.2841 meters
Wobble	1.0395 m	.6009 meters
Rocking	.9939 m	.1318 meters

The distances calculated with a known quaternion were able to be calculated in all three types of motion, as presented in Table 10. The standard deviation of the wobbling motion is higher than the other two motions because the first trials were at a lower velocity. The lower velocity caused the calculated distances to be higher than the distance traveled. This was similar to observations from the earlier cart and robot experiments. The algorithm was not able to accurately calculate the distance traveled when the linear velocity was too low. This hypothesis was verified by simplifying the motion to an arcing motion but maintaining a higher speed. In order to add additional variation, a rocking motion was introduced as an additional motion to the testing. All three motions had no effect on the ability of the algorithm to calculate the distance traveled as long as the velocity was high enough. As long as a means to accurately calculate the quaternion is available and the linear velocity of the system is high enough to overcome sensor noise, a pedestrian algorithm can be utilized to calculate distance traveled.

V. CONCLUSION

As demands from the DoD for unmanned and autonomous systems increase, the environments in which these systems operate also broaden. Indoor, shipboard, and GPS-denied environments make navigation for these systems difficult. This thesis investigated whether pedestrian tracking algorithms presented in [6] could be utilized to calculate the distance traveled by a robot with an IMU. A common issue with the use of inertial sensors for navigation has been drift error [6]. To address the drift error, this thesis explored use of the ZUPT. The ZUPT is utilized when the velocity is known to be zero in order to correct propagated error from acceleration data. Utilizing the ZUPT, the factored quaternion algorithm, and the OptiTrack visual tracking system, all sensor motion was observed and the distance traveled was calculated accurately.

A. ASSESSMENT OF GOALS

The purpose of this thesis was to confirm if a pedestrian tracking algorithm can be implemented on a wheeled robot chassis. Three goals were achieved by this thesis: first, the distance a robot can travel before an update needed to be performed was determined; second, the effect velocity has on the ability to calculate distance traveled was determined; and finally, the algorithm was proven effective at calculating distance traveled through arbitrary motion given a known quaternion.

To achieve the maximum distance a robot can travel, this study tested a robot chassis on a laboratory track with an attached sensor. From the laboratory tests, it was determined that the maximum distance achieved was three meters. Once the robot began traveling further than three meters, the accuracy of the distance traveled began to degrade.

To achieve the second goal, determining the effect of velocity, the study included similar tests with a robot and with an attached IMU sensor. From the robot experiments, a speed setting of three or four, as presented in Chapter III Section D, was needed to accurately calculate distance traveled. For individual sensors, testing determined that the

sensor needs to travel at as high a velocity as possible to accurately calculate distance traveled.

Finally, to accurately calculate distance traveled through arbitrary motion, an accurate transformation quaternion was calculated with data from the OptiTrack system. With the data transformed into the navigation frame, the acceleration data was used to calculate the distance traveled by the sensor. In an earlier experiment, the distance was not calculated due to the inability of the FQA to calculate an accurate quaternion during angular motion.

B. LIMITATIONS

In the course of this research, there were two major limitations to the implementation of the pedestrian algorithm. First, the system has to travel at a high enough velocity to overcome the effects of sensor noise. Second, an accurate transformation quaternion must be supplied to ensure the algorithm can properly transform the accelerometer data.

The first limitation hinders the system from being able to accurately calculate distance. If the system is traveling at a velocity too low to overcome the effects of sensor noise, the results begin to degrade. In all experiments, if the velocity was too low, the ability for the algorithm to calculate the distance traveled became inconsistent.

The second limitation again hinders the system from being able to accurately calculate the distance traveled. Without a correct transformation quaternion, the acceleration data cannot be transformed into the navigational reference frame. If the data is not in the navigational frame, the results calculated have no physical reference for navigational solutions. The FQA was not able to calculate an accurate quaternion during angular motion, which resulted in the need to include the OptiTrack system in the experiments.

C. RECOMMENDATIONS FOR FUTURE WORK

The following recommendations for future work are based on observations made during the testing of the sensors in each experiment summarized above. The recommendations also coincide with aspects of the research that were not a part of the overall scope of testing for this research.

One opportunity for future work would be to work on alternative means of calculating angles or quaternions accurately native to the system. The OptiTrack trials verified that, provided with accurate angular readings, the transformation matrix can be calculated. With a known transformation matrix, the algorithm can accurately calculate traveled distance. For future work, different algorithms other than the FQA, could be utilized to calculate the quaternions of the system. A survey of past algorithms is covered in [6].

Another avenue of research would be to implement the use of additional sensors to observe the angles achieved by the system in order to calculate the quaternions. Utilizing other onboard sensors with similar implementations like the ZUPT to correct errors could be explored. The onboard gyroscope was not employed because the sensor measures the angular velocity and would require integration in order to give the angle of the system. The need for integration creates an opportunity for similar drifts to be introduced to the data and would likely need additional sensors to correct. Additional market solutions could be explored to provide additional information in order to calculate the angles or quaternions associated with the robotic platform.

Another opportunity would be to implement these algorithms completely onto an autonomous system. Due to the constraints of this study, the algorithms were not implemented into a control algorithm for an autonomous system. A common chassis was explored in the robot trials that is easily configurable to accept the algorithm. Control algorithms were created for the same, or similar, robot system in [17]–[20], and could be augmented with results from this research, in order to provide the system with a better navigational picture in which to operate. In this case, the opportunity to also explore a

means of localization in conjunction with this work would be beneficial to the designing of an autonomous system that can operate in both indoor and outdoor environments.

APPENDIX A. DATA PROCESSING SCRIPT

```
clear all, close all
% Code generated by MATLAB for importing data
import_time2
% Code generated by MATLAB for importing data
import_AccY2

%Acceleration and time at the hand selected data range the robot is
moving
A = AccY(622:932)*9.8;  %*9.8 to convert from g-force to m/s^2
T = Time(622:932);

%initialize a counter to help do zero velocity update
tick = 0;

%processing via numerical integration
% Raw velocity
V = cumtrapz(T, A);

for i = 1:length(V)

    t(i) = tick;
    tick = tick +1;


    % Corrected Velocity
    % Adapted from Code Written by James Calusdian

    %Va = Vc - ((Vc(final time)/final time)*t), t = [0, finaltime]
    VC(i,1) = V(i) - ((V(length(V))/length(V))*t(i));

end

% Raw position
X = cumtrapz(T,V);

%Corrected Position
XC =cumtrapz(T, VC');

figure(1)
subplot(211)
plot(AccY)
title('Raw acceleration')
% annotation('line',[.5376 .5376], [.919 .5809])
% annotation('line',[0.65 0.7339], [.9262 .5857])

subplot(212)
plot(A)
title('windowed acceleration')
```

```
figure(2)
subplot(211)
plot(V)
title('Raw Velocity')
subplot(212)
plot(VC)
title('corrected velocity')

figure(3)
subplot(211)
plot(X)
title('Raw position')
display(X(length(X)))

subplot(212)
plot(XC)
title('Corrected Position')
display(XC(length(XC)))
r = snr(AccY)
```

APPENDIX B. FQA PROCESSING SCRIPT

```
clear all
close all
% Code generated by MATLAB for importing data
import_a_2
import_time3

A_y= AccY*9.8;
A_x= AccX*9.8;
A_z= AccZ*9.8;
T_r = Time;

%Take a sample when the sensor is not moving
a = [AccX(100); AccY(100); AccZ(100)];

%ignore magnetometer data
m = [0; 0; 0];

%steady state Quarternion
q = fqa_hardcode(a, m);
Angles = Euler(q);
Angles = Angles*(180/pi);


%conjugate steady state Quarternion
q_c = q .* [1; -1;-1;-1];

%range of data i.e. range where sensor is moving
r = [855:1280];

%Moment of motion and time stamp
AccX = AccX(r)*9.8;
AccY = AccY(r)*9.8;
AccZ = AccZ(r)*9.8;
T = Time(r);

%Array of Zeros
O = zeros(length(AccX),1);

%Acceleration Quarternion in the sensor body fram
A_b = [O, AccX, AccY, AccZ];
A_b = A_b';

%          | 0  |
%          |AccX|
%A_b/A_n =|AccY|
%          |AccZ|

%transformation from sensor body to navigational frame
for n = 1:length(O)
```

```
    %A_n = q * A_b * q_c
    %      |____________|
    %           |
    % Quarternion multiplication

    %Acceleration Quarternion in navigational frame
    %         q * A_b
%     a_n_tmp(:,n) = q_mult2(q, A_b(:,n));
    %         A_b * q_c
    %A_n(:,n) = q_mult2(q_c, a_n_tmp(:,n));
    A_n(:,n) = rotate_v_by_q(A_b(:,n), q);
end

A_n = A_n';

%initialize a counter to do zero velocity update
tick = 0;

%processing via numerical integration
% Raw velocity
V_x = cumtrapz(T, A_n(:,2));
V_y = cumtrapz(T, A_n(:,3));
V_z = cumtrapz(T, A_n(:,4));


for i = 1:length(O)

    t(i) = tick;

    %Corrected Velocity
    %Adapted from Code Written by James Calusdian

    %Va = Vc - ((Vc(final time)/final time)*t), t = [0, finaltime]
    VC_x(i,1) = V_x(i) - ((V_x(length(V_x))/length(V_x))*t(i));
    VC_y(i,1) = V_y(i) - ((V_y(length(V_y))/length(V_y))*t(i));
    VC_z(i,1) = V_z(i) - ((V_z(length(V_z))/length(V_z))*t(i));

    tick = tick +1;
end
%Raw Position
X_x =cumtrapz(T, V_x);
X_y =cumtrapz(T, V_y);
X_z =cumtrapz(T, V_z);

%Corrected Position
XC_x =cumtrapz(T, VC_x);
XC_y =cumtrapz(T, VC_y);
XC_z =cumtrapz(T, VC_z);

for m = 1:length(O)
    VC(m) = sqrt((VC_x(m)^2)+(VC_y(m)^2)+(VC_z(m)^2));
    V(m) = sqrt((V_x(m)^2)+(V_y(m)^2)+(V_z(m)^2));
```

```
    XC(m) = sqrt((XC_x(m)^2)+(XC_y(m)^2)+(XC_z(m)^2));
    X(m) = sqrt((X_x(m)^2)+(X_y(m)^2)+(X_z(m)^2));
end

%Plot of accelerations
figure(1)
subplot(211)
plot(T_r, A_y, T_r, A_x, T_r, A_z)
title('Raw acceleration')
legend('Y-axis', 'X-axis', 'Z-axis', 'Location', 'best');
% annotation('line',[.5376 .5376], [.919 .5809])
% annotation('line',[0.65 0.7339], [.9262 .5857])

subplot(212)
plot(T, AccY, T, AccX, T, AccZ)
title('Windowed acceleration')
legend('Y-axis', 'X-axis', 'Z-axis', 'Location', 'best');

%Plot of velocities
figure(2)
subplot(211)
plot(T, V_y, T, V_x, T, V_z, T, V);
title('Raw Velocity')
legend('Y-axis', 'X-axis', 'Z-axis', 'Combined', 'Location', 'best');
subplot(212)
plot(T, VC_y, T, VC_x, T, VC_z, T, VC);
title('Corrected Velocity')
legend('Y-axis', 'X-axis', 'Z-axis', 'Combined', 'Location', 'best');

%Plot of positions
figure(3)
subplot(211)
plot(X)
title('Raw Position')
subplot(212)
plot(XC)
title('Corrected Position')
display(XC(length(XC)))
display(X(length(X)))
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. FQA PROCESSING WITH MOTION SCRIPT

```
clear all
close all
% Code generated by MATLAB for importing data
import_a_2
import_time3

A_y= AccY*9.8;
A_x= AccX*9.8;
A_z= AccZ*9.8;
T_r = Time;

%ignore magnetometer data
m = [0; 0; 0];

%full calculated quarternion
for k = 1:length(A_x)

    %sample accelerometer
    am =[A_x(k); A_y(k); A_z(k)];

    qm(:, k) = fqa_hardcode(am, m);

    Angle = Euler(qm(:,k));
    AngleM(:,k) = Angle.*(180/pi);

end

%range of data i.e. range where sensor is moving
r = [509:712];

%Moment of motion and time stamp
AccX = AccX(r)*9.8;
AccY = AccY(r)*9.8;
AccZ = AccZ(r)*9.8;
T = Time(r);
q = qm(:, r);

%Array of Zeros
O = zeros(length(AccX),1);

%Acceleration Quarternion in the sensor body fram
A_b = [O'; AccX'; AccY'; AccZ'];

%          | 0  |
%          |AccX|
%A_b/A_n =|AccY|
%          |AccZ|

%transformation from sensor body to navigational frame
```

```
for n = 1:length(r)

    %A_n = q * A_b * q_c
    %      |_____________|
    %            |
    % Quarternion multiplication

    %Acceleration Quarternion in navigational frame
    %conjugate Quarternion
%     q_c = q(:,n) .* [1; -1;-1;-1];
%
%     %         q * A_b
%     a_n_tmp(:,n) = q_mult2(q(:,n), A_b(:,n));
%
%     %         A_b * q_c
%     A_n(:,n) = q_mult2(q_c, a_n_tmp(:,n));

      A_n(:,n) = rotate_v_by_q(A_b(:,n), q(:,n));

end

%initialize a counter to do zero velocity update
tick = 0;

%processing via numerical integration
% Raw velocity
V_x = cumtrapz(T, A_n(2,:));
V_y = cumtrapz(T, A_n(3,:));
V_z = cumtrapz(T, A_n(4,:));


for i = 1:length(r)

    t(i) = tick;

    %Corrected Velocity
    % Adapted from Code Written by James Calusdian

    %Va = Vc - ((Vc(final time)/final time)*t), t = [0, finaltime]
    VC_x(i,1) = V_x(i) - ((V_x(length(V_x))/length(V_x))*t(i));
    VC_y(i,1) = V_y(i) - ((V_y(length(V_y))/length(V_y))*t(i));
    VC_z(i,1) = V_z(i) - ((V_z(length(V_z))/length(V_z))*t(i));

    tick = tick +1;

end

%Raw Position
X_x =cumtrapz(T, V_x);
X_y =cumtrapz(T, V_y);
X_z =cumtrapz(T, V_z);
```

```
%Corrected Position
XC_x =cumtrapz(T, VC_x);
XC_y =cumtrapz(T, VC_y);
XC_z =cumtrapz(T, VC_z);

for m = 1:length(r)

    VC(m) = sqrt((VC_x(m)^2)+(VC_y(m)^2)+(VC_z(m)^2));

    V(m) = sqrt((V_x(m)^2)+(V_y(m)^2)+(V_z(m)^2));

    XC(m) = sqrt((XC_x(m)^2)+(XC_y(m)^2)+(XC_z(m)^2));

    X(m) = sqrt((X_x(m)^2)+(X_y(m)^2)+(X_z(m)^2));

end

%Plot of accelerations
figure(1)
subplot(211)
plot(T_r, A_y, T_r, A_x, T_r, A_z)
title('Raw acceleration')
legend('Y-axis', 'X-axis', 'Z-axis', 'Location', 'BEST');
% annotation('line',[.5376 .5376], [.919 .5809])
% annotation('line',[0.65 0.7339], [.9262 .5857])

subplot(212)
plot(T, A_n(3, :), T, A_n(2,:), T, A_n(4,:))
title('Rotated acceleration')
legend('Y-axis', 'X-axis', 'Z-axis', 'Location', 'BEST');

%Plot of velocities
figure(2)
subplot(211)
plot(T, V_y, T, V_x, T, V_z, T, V);
title('Raw Velocity')
legend('Y-axis', 'X-axis', 'Z-axis', 'Combined', 'Location', 'best');
subplot(212)
plot(T, VC_y, T, VC_x, T, VC_z, T, VC);
title('Corrected Velocity')
legend('Y-axis', 'X-axis', 'Z-axis', 'Combined', 'Location', 'best');

%Plot of positions
figure(3)
subplot(211)
plot(T, XC_x, T, XC_y, T, XC_z)
legend('X-axis', 'Y-axis', 'Z-axis','Location', 'best');
title('position w/respect to time')
subplot(212)
plot(XC_x, XC_y)
title('Position via x-y vector')
```

```
%Plot of angles through motion
figure(4)
subplot(311)
plot(T_r, AngleM(1,:));
title('Roll through motion')
xlabel('time')
ylabel('degrees')
subplot(312)
plot(T_r, AngleM(2,:));
title('Pitch through motion')
xlabel('time')
ylabel('degrees')
subplot(313)
plot(T_r, AngleM(3,:));
title('Yaw through motion')
xlabel('time')
ylabel('degrees')

display(X(length(X)))
display(XC(length(XC)))
```

APPENDIX D. HARDCODED FQA SCRIPT

```
function [q, error, flag] = fqa(a, m);
% a is 3x1, m is 3x1

% Code provided by James Calusdian from [6], and adapted from
% code written by Xiaoping Yun,  May 7, 2008

% input a = 3-dim acceleration, m=3-dim local magnetic measurement

%Mref = [0.4943  0.0  0.8693];

epsilon = 0.10;   % accuracy control constant
singular_flag = 0;
alpha = 30*pi/180;   % offset angle

% x is 3x2, first col = magnetometer, second col = accelerometer.

a_bar = a/norm(a);            % make sure that it is normalized
m_b = m/norm(m);

sin_th = a_bar(1);
cos_th = sqrt(1-sin_th^2);

% singularity avoidance algorithm
if (cos_th <= epsilon)
    singular_flag = 1;
    q_offset =cos(alpha/2)*[1 0 0 0]' + sin(alpha/2)*[0 0 1 0]';

    a_bar_q = [0; a_bar];
    m_b_q = [0; m_b];
    a_q_offset    = rotate_v_by_q(a_bar_q,q_offset);
    m_q_offset = rotate_v_by_q(m_b_q,q_offset);

    a_bar = a_q_offset(2:4);
    m_b = m_q_offset(2:4);

else
   % do not do anything other than setting the flag.
   singular_flag = 0;
end


% elevation quaternion-y
sin_th = a_bar(1);%h(1);
%cos_th = sqrt(1-sin_th^2);
cos_th = sqrt(a_bar(2)^2 + a_bar(3)^2);  %J.C. 1/30/2009

% computing half-angle values
cos_half_th=sqrt((1+cos_th)/2);
if (cos_th<=-1)      % this "if" is needed since sign(0) = 0.
```

```
    sin_half_th = 1;
else
    sin_half_th=sign(sin_th)*sqrt((1-cos_th)/2);
end


qe = cos_half_th*[1;0;0;0] + sin_half_th*[0;0;1;0];

%%%% Roll Quaternion-x
b = [a_bar(2) a_bar(3)];
c = b/norm(b);
sin_phi = -c(1);
cos_phi = -c(2);


cos_half_phi=sqrt((1+cos_phi)/2);
if (cos_phi<=-1)
    sin_half_phi = 1;
else
    sin_half_phi=sign(sin_phi)*sqrt((1-cos_phi)/2);
end


qr = cos_half_phi*[1;0;0;0] + sin_half_phi*[0;1;0;0];

%%%% Azimuth Quaternion-z
% Commented out due to magnetometer unable to be read and quaternion
% hardcoded

% qe_inv = [qe(1);-qe(2);-qe(3);-qe(4)];
% qr_inv = [qr(1);-qr(2);-qr(3);-qr(4)];
% m_b_q = [0; m_b];
%
% q_er = q_mult2(qe,qr);
% q_er_inv =[q_er(1); -q_er(2); -q_er(3); -q_er(4)];
% m_e = q_mult2(q_er,q_mult2(m_b_q, q_er_inv));
%
% M = [m_e(2),m_e(3)];
% M = M/norm(M);
% N = [1; 0];
% tmp =  [ M(1) M(2);
%          -M(2) M(1)]*N;
% cos_psi = tmp(1);
% sin_psi = tmp(2);
%
% cos_half_psi=sqrt((1+cos_psi)/2);
% if (cos_psi<=-1)                    %%%% IMPORTANT %%%  if it is
written as cos_psi==-1, it does not work.
%                                     %%%% cos_psi is potentially less
than -1.
%     sin_half_psi = 1;
% else
%      sin_half_psi=sign(sin_psi)*sqrt((1-cos_psi)/2);
% end
```

```
%
% qa = cos_half_psi*[1;0;0;0]+ sin_half_psi*[0;0;0;1];

%assume no yaw
qa = [1;0;0;1];

q_tmp1 = q_mult2(qe,qr);
q_tmp = q_mult2(qa,q_tmp1);

if (singular_flag == 1)
    q = q_mult2(q_tmp, q_offset);
else
    q = q_tmp;
end

error = cos_th;
flag = singular_flag;
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. EULER ANGLE FUNCTION

```
function EulerAngles=Euler(u)
%Code provided by James Calusdian from [6]

q0=u(1);
q1=u(2);
q2=u(3);
q3=u(4);

B=[q0^2+q1^2-q2^2-q3^2 2*(q1*q2+q3*q0) 2*(q1*q3-q0*q2);
   2*(q1*q2-q0*q3) q0^2-q1^2+q2^2-q3^2 2*(q2*q3+q0*q1);
   2*(q1*q3+q0*q2) 2*(q2*q3-q0*q1) q0^2-q1^2-q2^2+q3^2];

% if (B(1,3) >=1)
%     B(1,3) =1;
% elseif (B(1,3) <=-1)
%     B(1,3) =-1;
% end

phi=atan2(B(2,3),B(3,3));

theta=-asin(B(1,3));
psi=atan2(B(1,2),B(1,1));

EulerAngles=[phi;
             theta;
             psi];
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F. ROTATION FUNCTION

```
function u=rotate_v_by_q(v,q)
%Code provided by James Calusdian from [6]

q_inv= [q(1) -q(2) -q(3) -q(4)]';

u = q_mult2(q,q_mult2(v,q_inv));
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX G. QUATERNION MULTIPLICATION FUNCTION

```
function qout=q_mult2(p,q)
%Code provided by James Calusdian from [6]

P_mat = [p(1) -p(2) -p(3) -p(4);
         p(2)  p(1) -p(4)  p(3);
         p(3)  p(4)  p(1) -p(2);
         p(4) -p(3)  p(2)  p(1)];
qout = P_mat*q;
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX H. OPTITRACK PROCESSING SCRIPT

```
%%Import data
close all
clear all
% Code generated by MATLAB for importing data
%import all the data
import_a_microstrain
import_time_microstrain
% Code generated by MATLAB for importing data
import_angles
roll = VarName4;
pitch = VarName5;
yaw = VarName6;

% Code generated by MATLAB for importing data
import_time_optitrack
Time_O = VarName7;

%clear out read in variable names to clean up workspace.
vars= {'VarName4', 'VarName5', 'VarName6', 'VarName7'};
clear(vars{:})
clear vars

%Starting point for IMU data
I_s = 181;

%Starting point for OptiTrack data
O_s = 509;

%start IMU data at verified point
AccX_v = AccX(I_s:end);
AccY_v = AccY(I_s:end);
AccZ_v = AccZ(I_s:end);
Time_v = Time(I_s:end);
Time_v = Time_v - Time_v(1);

%start OptiTrack data at verified point
[Time_O_n, roll_n, pitch_n, yaw_n] = filter_repeat(Time_O, roll, pitch,
yaw);

roll_v = roll_n(O_s:end);
pitch_v = pitch_n(O_s:end);
yaw_v = yaw_n(O_s:end);
Time_O_v = Time_O_n(O_s:end);
Time_O_v = Time_O_v-Time_O_v(1);

%% Run simulation
%inperpolate values to get data to match up
Ax = interp1(Time_v, AccX_v, Time_O_v);
Ay = interp1(Time_v, AccY_v, Time_O_v);
Az = interp1(Time_v, AccZ_v, Time_O_v);
```

```
Angles = [roll_v, pitch_v, yaw_v];

%convert Euler angles to Quaternions from optitrack
for i = 1:length(Angles)

%     Q_mat(:,i) = eul2quat(Angles(i,:), 'XYZ');

    q = myEuler2quaternion(yaw_v(i), pitch_v(i), roll_v(i));
    Q_opt(:,i) = q;

end

%calculate the quaternion with FQA from acceleration to compare
for k = 1:length(Ax)

    %sample accelerometer
    am =[Ax(k); Ay(k); Az(k)];
    m = [0; 0; 0];

    qm(:, k) = fqa_hardcode(am, m);

end

%range of the data
range_OPT = [273:469];
t = Time_O_v(range_OPT);
t_0 = t - t(1);
A_x = Ax(range_OPT)*9.8;
A_y = Ay(range_OPT)*9.8;
A_z = Az(range_OPT)*9.8;
Q = Q_opt(:,range_OPT);

for n = 1:length(range_OPT)

       %Rotate from the body frame into the navigation frame
       A_b = [0; A_x(n); A_y(n); A_z(n)];
       A_n(:,n) = rotate_v_by_q(A_b, Q(:,n));

end

%processing via numerical integration
% Raw velocity
V_x = cumtrapz(t_0, A_n(2,:));
V_y = cumtrapz(t_0, A_n(3,:));
V_z = cumtrapz(t_0, A_n(4,:));

% for i = 1:length(range_OPT)
%
%     %Corrected Velocity
%     %Va = Vc - ((Vc(final time)/final time)*t), t = [0, finaltime]
%     VC_x(i) = V_x(i) - ((V_x(end)/t_0(end))*t_0(i));
```

```
%     VC_y(i) = V_y(i) - ((V_y(end)/t_0(end))*t_0(i));
%     VC_z(i) = V_z(i) - ((V_z(end)/t_0(end))*t_0(i));
%
% end

t_int = t_0';

%Velocity correction X
% Adapted from Code Written by James Calusdian

%correct velocity at starting point of motion
Vx_c = V_x - V_x(1);

%calculate error through motion
error_x = Vx_c(end)/t_int(end)*t_int;

%Correct velocity through motion
VC_x = Vx_c - error_x;

%Velocity correction Y
Vy_c = V_y - V_y(1);

error_y = Vy_c(end)/t_int(end)*t_int;

VC_y = Vy_c - error_y;

%velocity correction Z
Vz_c = V_z - V_z(1);

error_z = Vz_c(end)/t_int(end)*t_int;

VC_z = Vz_c - error_z;

tock = 0;

%Raw Position
X_x =cumtrapz(t_0, V_x);
X_y =cumtrapz(t_0, V_y);
X_z =cumtrapz(t_0, V_z);

%Corrected Position
XC_x =cumtrapz(t_0, VC_x);
XC_y =cumtrapz(t_0, VC_y);
XC_z =cumtrapz(t_0, VC_z);

for m = 1:length(range_OPT)

    VC(m) = sqrt((VC_x(m)^2)+(VC_y(m)^2)+(VC_z(m)^2));

    V(m) = sqrt((V_x(m)^2)+(V_y(m)^2)+(V_z(m)^2));

    XC(m) = sqrt((XC_x(m)^2)+(XC_y(m)^2)+(XC_z(m)^2));
```

```
    X(m) = sqrt((X_x(m)^2)+(X_y(m)^2)+(X_z(m)^2));

end

%Plot of accelerations
figure(1)
subplot(211)
plot(Time, AccX, Time, AccY, Time, AccZ)
title('Raw acceleration')
legend('X-axis', 'Y-axis', 'Z-axis', 'Location', 'BEST');
xlabel('Time (Seconds)')
ylabel('Meters/Second^2')

subplot(212)
plot(t_0, A_n(2,:), t_0, A_n(3,:), t_0, A_n(4,:))
title('Rotated acceleration')
legend('X-axis', 'Y-axis', 'Z-axis', 'Location', 'BEST');
xlabel('Time (Seconds)')
ylabel('Meters/Second^2')

%Plot of velocities
figure(2)
subplot(211)
plot(t_0, V_y, t_0, V_x, t_0, V_z, t_0, V);
title('Raw Velocity')
legend('Y-axis', 'X-axis', 'Z-axis', 'Combined', 'Location', 'best');

subplot(212)
plot(t_0, VC_y, t_0, VC_x, t_0, VC_z, t_0, VC);
title('Corrected Velocity')
legend('Y-axis', 'X-axis', 'Z-axis', 'Combined', 'Location', 'best');

%Plot of positions
figure(3)
subplot(211)
plot(t_0, XC_x, t_0, XC_y, t_0, XC_z)
legend('X-axis', 'Y-axis', 'Z-axis','Location', 'best');
title('position w/respect to time')

subplot(212)
plot(XC_x, XC_y)
axis([0 length(XC_x) 0 length(XC_x)])
title('Position via x-y vector')

%Plot of positions
figure(4)
subplot(211)
plot(X)
title('Raw Position')
subplot(212)
plot(XC)
title('Corrected Position')
```

```
figure(5)
subplot(211)
plot(Time_O_v, Q_opt(2,:), Time_O_v, Q_opt(3,:), Time_O_v, Q_opt(4,:))
title('OptiTrack Quaternion');
xlabel('Time (Seconds)')


subplot(212)
plot(Time_O_v, qm(2,:), Time_O_v, qm(3,:), Time_O_v, qm(4,:))
title('FQA Quaternion');
xlabel('Time (Seconds)')

display(X(end))
display(XC(end))

figure(6)
subplot(211)
plot(Time, AccX, Time, AccY, Time, AccZ)
title('Raw acceleration')
legend('X-axis', 'Y-axis', 'Z-axis', 'Location', 'BEST');
xlabel('Time (Seconds)')
ylabel('Meters/Second^2')

subplot(212)
plot(Time_O_v, qm(2,:), Time_O_v, qm(3,:), Time_O_v, qm(4,:))
title('FQA Quaternion');
xlabel('Time (Seconds)')
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX I. EULER ANGLE TO QUATERNION FUNCTION

```
function [q] = myEuler2quaternion(yaw, pitch, roll)
% myEuler2quaternion converts the Euler angles (radians) to the
%quaternion
% with the form [q0 q1 q2 q3 q4] where q0 is the scalar.
% from lecture notes [21]
cPsi2 = cos(yaw/2);
sPsi2 = sin(yaw/2);
cTheta2 = cos(pitch/2);
sTheta2 = sin(pitch/2);
cPhi2 = cos(roll/2);
sPhi2 = sin(roll/2);

q0 = cPsi2*cTheta2*cPhi2 + sPsi2*sTheta2*sPhi2;
q1 = cPsi2*cTheta2*sPhi2 - sPsi2*sTheta2*cPhi2;
q2 = cPsi2*sTheta2*cPhi2 + sPsi2*cTheta2*sPhi2;
q3 = sPsi2*cTheta2*cPhi2 - cPsi2*sTheta2*sPhi2;

q = [q0;q1;q2;q3];
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX J. FILTER REPEAT DATA FUNCTION

```
function [Time_O, Roll, Pitch, Yaw] = filter_repeat(time, roll, pitch,
yaw)
%Code adapted from code provided by James Calusdian.
%Original code written by James Calusdian,
%%%remove_bad_data

%time_opt = test10.time_opt;
%N = length(time_opt);
%array_of_indices = [];

%pitch_opt = test10.pitch_opt;
%yaw_opt = test10.yaw_opt;
%roll_opt = test10.roll_opt;

%for ix = 2:N
%    if time_opt(ix) == time_opt(ix-1)
%        array_of_indices = [array_of_indices , ix];
%    end
%end

%%% fix the time vector
%time_fixed = time_opt;
%time_fixed(array_of_indices) = [];
%time_optitrack_zero = time_fixed - time_fixed(1);

%%% fix the angle data, too
%pitch_opt(array_of_indices) = [];
%yaw_opt(array_of_indices) = [];
%roll_opt(array_of_indices) = [];

%Function to filter out repeat stagnant data from Optitrack data
so%that it
%can be interpolated by processing function
array = [];

%for loop to check for repeated data points
for i = 2:length(time)
    %if data is repeated save the location in array
    if time(i) == time(i-1)
       array = [array, i];
    end

end


    time(array) = [];
    roll(array) = [];
    pitch(array) = [];
    yaw(array) = [];
```

```
    Time_O = time;
    Roll = roll;
    Pitch = pitch;
    Yaw = yaw;

end
```

LIST OF REFERENCES

[1] X. Yun, J. Calusdian, E. R. Bachmann, and R. B. McGhee, "Estimation of human foot motion during normal walking using inertial and magnetic sensor measurements," *IEEE Trans. Instrum. Meas.*, vol. 61, no. 7, pp. 2059–2072, 2012.

[2] W. Storms, J. Shockley, and J. Raquet, "Magnetic field navigation in an indoor environment," *2010 Ubiquitous Position. Indoor Navig. Locat. Based Serv. UPINLBS 2010*, pp. 1–10, 2010.

[3] V. Sazdovski and P. M. G. Silson, "Inertial navigation aided by vision-based simultaneous localization and mapping," *IEEE Sens. J.*, vol. 11, no. 8, pp. 1646–1656, 2011.

[4] H. Lategahn and C. Stiller, "Vision-Only Localization," *IEEE Trans. ON Intel. Transp. Sys.*, vol. 15, no. 3, pp. 1246–1257, 2014.

[5] Y. Zhou, C. L. Law, and F. Chin, "Construction of local anchor map for indoor position measurement system," *IEEE Trans. Instrum. Meas.*, vol. 59, no. 7, pp. 1986–1988, 2010.

[6] J. Calusdian, "A personal navigation system based on inertial and magnetic field measurements," Ph. D. dissertation, Dept. of Electrical Engineering, NPS, Monterey, CA, USA, 2010. [Online]. Available: http://hdl.handle.net/10945/10557.

[7] X. Yun, E. R. Bachmann, and R. B. Mcghee, "A Simplified Quaternion-Based Algorithm for Orientation Estimation From Earth Gravity and Magnetic Field Measurements," *IEEE Trans. Instrum. Meas*., vol. 57, no. 3, pp. 638–650, 2008.

[8] J. O. Nilsson, J. Rantakokko, P. Händel, I. Skog, M. Ohlsson, and K. V. S. Hari, "Accurate indoor positioning of firefighters using dual foot-mounted inertial sensors and inter-agent ranging," *IEEE PLANS, Position Locat. Navig. Symp*., pp. 631–636, 2014.

[9] 3DM-GX3^{R}-25-Miniature Attitude Heading Reference System (AHRS), Document 8400-0033 Revision 003, LORD Corporation MicroStrain Sensing Systems, Williston, VT, 2014. [Online]. Available: http://files.microstrain.com/3DM-GX3-25-Attitude-Heading-Reference-System-Data-Sheet.pdf

[10] 3DM-GX4-25^{TM}-Attitude Heading Reference System (AHRS), Document 8400-0060 Revision A, LORD Corporation MicroStrain Sensing Systems, Williston, VT, 2014. [Online]. Available: http://files.microstrain.com/3DM-GX4-25_Datasheet_(8400-0060).pdf

[11] 3DM-GX5-25-OEM™ OEM Attitude Heading Reference System (AHRS), Document 8400-0093 Revision N, LORD Corporation MicroStrain Sensing Systems,Williston,VT,2019. [Online]. Available: https://www.microstrain.com/site/default/files/applications/files/3dm-gx5-25_datasheet_8400-0093_rev_n.pdf

[12] *3-Space Sensor Miniature Attitude & Heading Reference System User's Manual,* Yost Labs, Portsmouth, OH, USA, 2007. [Online]. Available: https://yostlabs.com/wp/wp-content/uploads/pdf/3-Space-Sensor-Users-Manual-1.pdf

[13] MathWorks, Inc., "What is MATLAB?" 2020. [Online]. Available: https://www.mathworks.com/discovery/what-is-matlab.html

[14] OptiTrack, "Motive: Tracker Motion capture & 6 DOF object tracking." 2020. [Online]. Available: https://www.optitrack.com/products/motive/tracker.html

[15] MathWorks, Inc., "cumtrapz" 2020. [Online]. Available: https://www.mathworks.com/help/matlab/ref/cumtrapz.html?s_tid=srchtitle

[16] MathWorks, Inc., "interp1" 2020. [Online]. Available: https://www.mathworks.com/help/matlab/ref/interp1.html

[17] C. S. Hargadine, "Mobile robot navigation and obstacle avoidance in unstructured outdoor environments," M.S. thesis, Dept. of Electrical Engineering, NPS, Monterey, CA, 2017. [Online]. Available: https://hdl.handle.net/10945/56937

[18] M. R. Audette, "Interactive Map Making for Route Planning and Obstacle Avoidance in an Unstructured Outdoor Environment," M.S. thesis, Dept. of Electrical Engineering, NPS, Monterey, CA, 2018. [Online]. Available: https://hdl.handle.net/10945/60406

[19] C. Lebrun, "Vision-based Terrain Classification and Learning to Improve Autonomous Ground Vehicle Navigation in Outdoor Environments," M.S. thesis, Dept. of Electrical Engineering, NPS, Monterey, CA, 2019. [Online]. Available: https://hdl.handle.net/10945/63474

[20] A. Magee, "Place-Based Navigation for Autonomous Vehicles with Deep Learning Neural Networks," M.S. thesis, Dept. of Electrical Engineering, NPS, Monterey, CA, 2019. [Online]. Available: https://hdl.handle.net/10945/64012

[21] "Equations of Motion for Three Dimensional Rigid Bodies," class notes EC4330: Navigation, Missile, and Avionics Systems, Dept. of Electrical and Computer Engineering, Naval Postgraduate School, Monterey, CA, USA, spring 2019.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California