NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ROBOTIC NAVIGATION AND MAPPING IN GPS-DENIED ENVIRONMENTS WITH 3D LIDAR AND INERTIAL NAVIGATION UTILIZING A SENSOR FUSION ALGORITHM

by

Matthew G. Caspers

September 2021

Thesis Advisor: Xiaoping Yun
Co-Advisor: 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** September 2021	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
ROBOTIC NAVIGATION AND MAPPING IN GPS-DENIED ENVIRONMENTS WITH 3D LIDAR AND INERTIAL NAVIGATION UTILIZING A SENSOR FUSION ALGORITHM

5. FUNDING NUMBERS

6. AUTHOR(S) Matthew G. Caspers

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)

Global Navigation Satellite Systems (GNSS) do not currently offer viable solutions for autonomous robotic navigation in a tactical scenario, as GNSS networks are susceptible to enemy jamming and loss of coverage within buildings. Several methods for interior navigation and mapping fuse data inputs from inertial measurement units (IMU) and light detection and ranging (lidar) sensors to generate more robust simultaneous localization and mapping (SLAM) solutions. However, these methods rely on large point-cloud data sets to achieve SLAM that increases processing requirements. This work seeks to find a novel solution for decreasing point-cloud processing requirements for SLAM by combining IMU and lidar sensor inputs. Utilizing a strap-down navigation algorithm with zero-velocity updates, a fusion algorithm generates an estimated transform between lidar scans that is then provided as the input to an iterative closest point registration (ICP) algorithm to generate a SLAM solution. It was found that the required number of point-clouds for generating SLAM solutions was reduced by at least five times while still maintaining functionality through multiple rotations and translations over several meters. Future work recommendations include expansion of the fusion algorithm onto autonomous platforms and generating more efficient process flows to further reduce SLAM processing requirements.

14. SUBJECT TERMS SLAM, lidar, inertial sensors, navigation, sensor fusion, strap-down	**15. NUMBER OF PAGES** 91
	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 AND MAPPING IN GPS-DENIED ENVIRONMENTS WITH 3D LIDAR AND INERTIAL NAVIGATION UTILIZING A SENSOR FUSION ALGORITHM

Matthew G. Caspers
Captain, United States Marine Corps
BSME, The Ohio State University, 2015

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2021

Approved by: Xiaoping Yun
Advisor

James Calusdian
Co-Advisor

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

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Global Navigation Satellite Systems (GNSS) do not currently offer viable solutions for autonomous robotic navigation in a tactical scenario, as GNSS networks are susceptible to enemy jamming and loss of coverage within buildings. Several methods for interior navigation and mapping fuse data inputs from inertial measurement units (IMU) and light detection and ranging (lidar) sensors to generate more robust simultaneous localization and mapping (SLAM) solutions. However, these methods rely on large point-cloud data sets to achieve SLAM that increases processing requirements. This work seeks to find a novel solution for decreasing point-cloud processing requirements for SLAM by combining IMU and lidar sensor inputs. Utilizing a strap-down navigation algorithm with zero-velocity updates, a fusion algorithm generates an estimated transform between lidar scans that is then provided as the input to an iterative closest point registration (ICP) algorithm to generate a SLAM solution. It was found that the required number of point-clouds for generating SLAM solutions was reduced by at least five times while still maintaining functionality through multiple rotations and translations over several meters. Future work recommendations include expansion of the fusion algorithm onto autonomous platforms and generating more efficient process flows to further reduce SLAM processing requirements.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

CCW	Counterclockwise
CPD	Coherent Point Distribution
EKF	Extended Kalman Filter
FOV	field of view
FQA	Factored Quaternion Algorithm
GNSS	Global Navigation Satellite System
ICP	Iterative Closest Point
IMU	inertial measurement unit
KD tree	k-dimensional tree
KT	KinematicTrajectory algorithm
lidar	light detection and ranging
NDT	Normal Distribution Transform
NED	North-East-Down Frame
ROS	Robotic Operating System
SLAM	Simultaneous Location and Mapping
ZUPT	Zero Velocity Update

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I want to thank my Lord and savior for his providential grace, as well as the prayers and encouragement from family, friends, and fellow believers in the undertaking and completion of this project.

I am indebted to my faculty advisors, Dr. Xiaoping Yun and Dr. James Calusdian. Dr. Yun, without your guidance, patience, encouragement, and the countless hours spent reviewing varied concepts, autonomous navigation theory, and ability to simplify complex theories, this thesis would not have been possible. Dr. Calusdian, heartfelt thanks are due for your constant reality checks in scoping this work, ready solutions for hardware problems, and valuable insights into paths and solutions for moving the thesis forward.

To the ECE gang, thank you for providing inspiration and assistance throughout this journey. To my fellow lab partner and ECE student, Captain Justin Bracci, U.S. Marine Corps, you have my deep appreciation for our daily discussions on research approaches and for your patience in daily acting as an idea sounding board.

Lastly, my deepest thanks go to my family for your encouragement in starting, and throughout, this journey. Without your love, prayers, and encouragement, I would never have endured the rigors demanded.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Autonomous robotic navigation and control utilizes multiple sensors to provide a more thorough and accurate localization of a robot within local space for navigation and decision making processes. However, Global Navigation Satellite System (GNSS) localization solutions are not viable for autonomous robotic navigation in tactical scenarios as GNSS networks suffer from loss of coverage within buildings and are susceptible to enemy electronic attacks. A solution presented in this thesis utilizes a fusion of inertial measurement units (IMU) and light detection and ranging (lidar) based solutions for simultaneous localization and mapping (SLAM) with potential for lower computational costs. Topics covered in this chapter include a motivation for the research, overview of related work, and the purpose and goal of the research conducted.

A. MOTIVATION

Autonomous robotics systems increasingly find applications in military domains as discussed in [1] but often rely on GNSS solutions, a non-viable method inside buildings or when the enemy jams or spoofs signals. Methods utilized for countering this problem rely on a combination of sensors including vision, IMU, and lidar scans to provide localization inside buildings. Many of these methods suffer shortfalls. IMU methods exhibit positional error drift over time as seen in [2] while approaches implementing lidar suffer from higher computational and power costs that decrease responsiveness [3]. Already high carry loads placed on service members in combat lead to the requirement of a platform capable of SLAM, and also light enough for man-packability. High power and demanding computational resources with accurate localization often preclude a lower platform weight. Therefore, methods for obtaining accurate SLAM readings with lower computational costs that aid in weight reduction are required.

B. RELATED WORK

Many solutions for navigation and mapping rely on methods derived from inertial, lidar, and visual-based methods. Discussions of each method follow in this section.

1. Inertial Navigation

A common solution for inertial navigation involves calculating distance traveled through integration of acceleration, and utilization of magnetometers and angular velocity measurements. Past methods for this have been implemented with pedestrian algorithms for navigation as in [2] and [4]. Such a method has also been adapted for use on robotic and cart platforms demonstrated in [2]. Inertial solutions are commonly used in combination with additional localization methods as shown in [5], [6], and [7] to better improve the robustness of additional localization methods. However, inertial methods used alone have significant sensor drift as demonstrated in [2] and [4] and provide minimal information of the surrounding environment beyond distance travelled. In [2] and [4], error correction techniques utilize velocity and position from an inertial measurement unit (IMU) to compensate for drift over larger distances.

2. Lidar Navigation

Lidar solutions utilize sequential scans, stored as point-clouds, to map an environment; the scans are then sequentially registered to generate localization solutions. Point-clouds are ordered data sets of coordinates from all individual laser returns in a lidar scan. Methods for registering the scans differ with several solutions implemented or discussed in [8] and [9]. The advantages of lidar include more detailed information of the surrounding environment, ranging data, and the ability to generate mapping solutions without external illumination required for many vision solutions. However, lidar registration techniques come with higher computational costs as in seen in [9] with longer processing times. Lidar registration drift over longer distances also occurs requiring further processing to correct. As with inertial solutions, fusion of different sensors often gives better transform registration estimates between successive lidar scans and provides more robust localization solutions [8]. The tutorial in [10] covers one such method of combining data with raw IMU orientation outputs to help provide a better rotational estimate prior to lidar registration.

3. Vision Navigation

Since the steady decrease in cost for cameras, SLAM solutions via vision sensors have increased with several methods detailed [8]. Visual solutions utilize a variety of methods for extracting data from the surrounding environment for SLAM that include feature-based, direct, colored depth, and event-based methods [8]. Some methods rely on direct matching of pixels to compute transforms between each visual frame while others rely on features within each frame to track and compute transforms [11]. However, most methods are subject to error from light or texture variation that largely limit visual SLAM usage to indoor scenarios, or visual methods are computationally intensive. The low cost of sensors for most methods, however, makes this an attractive approach as seen by the large body of research available for visual SLAM [8], [11]. Like inertial and lidar approaches, visual navigation can fuse additional sensor inputs to produce more robust solutions as seen in [12] with visual and inertial fusion.

C. PURPOSE AND GOAL

The purpose of this research sought to discern the effectiveness of pairing a zero velocity update (ZUPT) algorithm with lidar point-cloud registration techniques for generating SLAM solutions with the intent of reducing computational costs for lidar registration. The ZUPT algorithm came from modification of research conducted by Druen [2]. In [2], Druen modified a pedestrian tracking algorithm built in [4] for utilization on a cart and robotic platform. The lidar registration and mapping techniques follow from algorithms provided in [13], work done in [9], and scripts modified from [10]. This thesis seeks to achieve three goals. First, determine the individual effectiveness of the ZUPT algorithm and lidar registration techniques for two-dimensional localization. Second, determine if fusing the ZUPT algorithm with a lidar registration technique produces a viable SLAM solution. Finally, determine if combining the two algorithms produces computational processing reductions. The research builds on work previously done by Druen in [2] and by Calusdian in [4] in addition to work done by Payne in [9].

Chapter II includes background concepts utilized throughout the research done. In Chapter III, the hardware and software used in the thesis is covered. Chapter IV covers

each major component of the fusion algorithm while Chapter V will include research results. Chapter VI lays out conclusions drawn from the research as well as limitations and future recommendations for research.

II. BACKGROUND

Background information and concepts utilized throughout this thesis are discussed within this chapter. Specific topics include reference frames and an overview of quaternion and transform operations crucial for the functioning of several algorithms in this thesis. An overview of the ZUPT is covered as well.

A. REFERENCE FRAMES

Within this thesis, four reference frames were utilized for transforming data. The reference frames are the sensor reference frame, quaternion reference frame, cart reference frame, and world reference frame. Each reference frame is explained in greater detail within this subsection. For clarity, throughout this thesis, a leading superscript denotes the reference frame.

1. Sensor Reference Frame

Acceleration data collected from the IMU sensor was collected in the sensor reference frame. The sensor reference frame is fixed to the centroid of the IMU sensor and moves with the IMU sensor. Shown, in Figure 1, are the orientations of the sensor reference frame cartesian axes. The x-direction points backwards from the IMU sensor and aligns with the sensor primary axis. The y-direction aligns with the secondary axis of the sensor. The z-direction points downward from the sensor centroid. The orientation of the sensor reference frame is such that unprocessed acceleration data from the IMU is in the negative x-direction of the sensor frame for all experimentation.

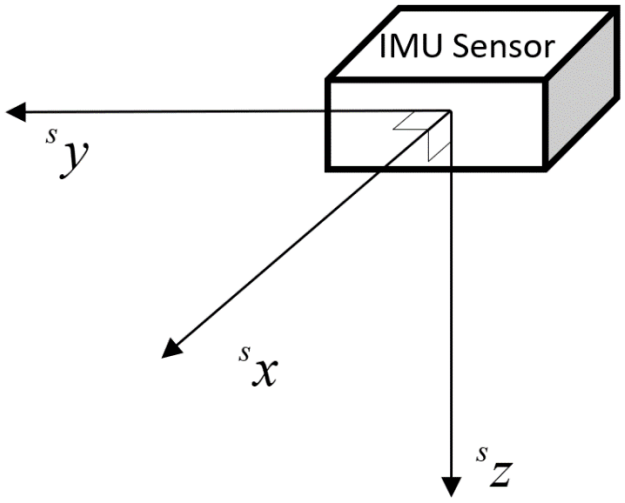

Figure 1. Sensor reference frame

2. Quaternion Reference Frame

To map the data from the sensor frame with pose required multiplication with a quaternion. This led to an intermediate, fixed North-East-Down (NED) frame called the quaternion reference frame. Upon initialization of the IMU, the sensor and quaternion reference frames are the same until the IMU sensor moves. The quaternion reference frame was used for calculating two-dimensional positional data from the IMU sensor acceleration.

3. Cart Reference Frame

An intermediate moving frame of reference was required for fusion of all sensor data prior to transformation into a world frame for mapping. The sensors used in the thesis were not physically coincident and required a common reference frame for correct interpretation and processing of data. As much of the software packages utilized for registering lidar point-clouds required data in the lidar sensor reference frame, the lidar native frame of reference was utilized as an intermediate reference frame. The intermediate reference frame is called the cart reference frame throughout the thesis. The cart reference frame is a cartesian coordinate system with the origin coincident to the centroid of the lidar and the x-direction right from rotational center of the lidar. By setting the cart reference

frame coincident to the lidar sensor, additional transformations of large lidar sensor data sets were avoided. Shown, in Figure 2, is the configuration of the cart reference frame, the coincident lidar native frame, and the sensor reference frame.

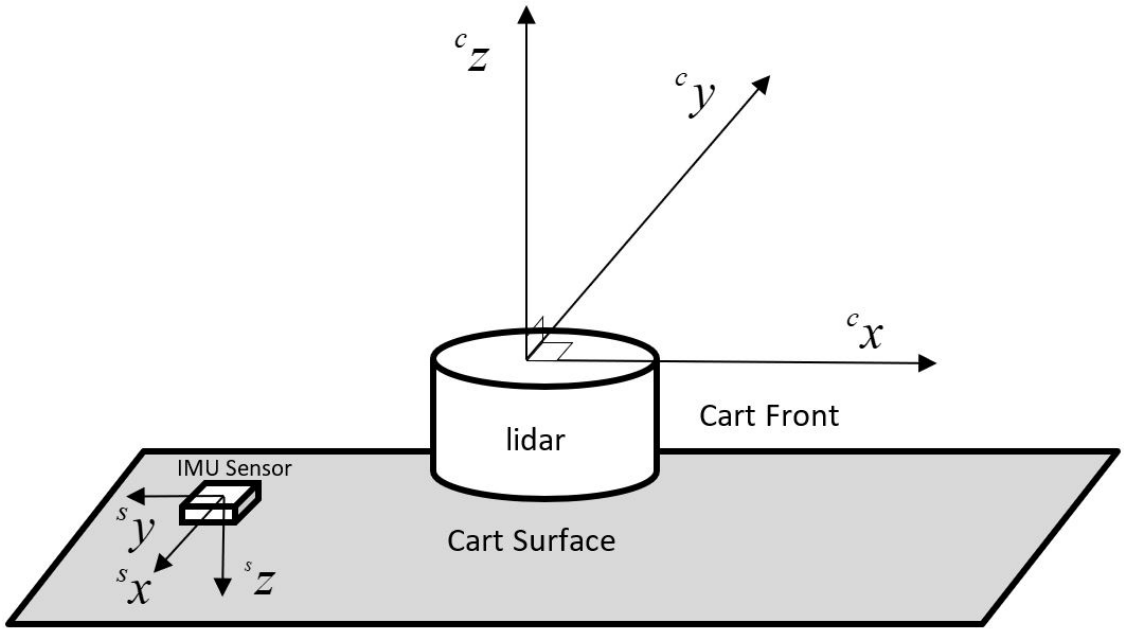

Figure 2. Cart reference frame

4. World Reference Frame

The world mapping frame is a fixed frame of reference with the origin set on initialization of the lidar sensor. As the lidar sensor utilizes the cart reference frame, a cartesian coordinate system follows as shown in Figure 2 with the z-direction right from the rotational axis of the lidar, y-direction forward, and z-direction up. The world reference frame and cart reference frame are the same at initialization of the lidar until the lidar moves.

B. QUATERNIONS AND TRANSFORMS

A mixture of quaternions and transformations were utilized for transforming data into the correct reference frames throughout the thesis research. Usage of quaternions was

crucial in the ZUPT algorithm for correctly interpreting positional data. Quaternions are hyper-complex numbers that can act as rotational operators in three-dimensional space [14]. From [4], quaternions are a mixture of a scalar and a vector and sometimes are defined in notation as a 4×1 array

$$q = [q_0, \vec{q}] = [q_0, q_1, q_2, q_3]. \tag{1}$$

As presented in [4], given two quaternions in the form of Equation (1), quaternion multiplication is

$$aq = a_0 q_0 - \vec{a} \cdot \vec{q} + \vec{a}_0 \vec{q} + q_0 \vec{a} + \vec{a} \times \vec{q}. \tag{2}$$

The conjugate of a quaternion, denoted as q^* from [4], is calculated as such:

$$q^* = [q_0, -q_1, -q_2, -q_3]. \tag{3}$$

Building off the work done in [4], quaternions were used for transformation of the IMU acceleration data in the sensor frame into the quaternion reference frame for use in two-dimensional application of a ZUPT algorithm. The orientation of the IMU sensor is such that only x-axis acceleration data of the IMU in the sensor frame is relevant for this thesis work. The general form of acceleration collected is a 4×1 array

$$^s a = \left[0, {}^s a_x, 0, 0\right] \tag{4}$$

where the y and z acceleration components are zero. Zero padding must be included with three-dimensional vectors for proper quaternion operations. As in [4], acceleration in the sensor frame is multiplied with a pose quaternion to rotate the sensor acceleration into the quaternion frame and is calculated as such:

$$^q a = q_p (^s a q_p^*). \tag{5}$$

The resulting rotated acceleration components form a 4×1 array

$$ {}^{q}a = [0, {}^{q}a_x, {}^{q}a_y, {}^{q}a_z] \tag{6} $$

with non-zero components for the x, y, and z components in the quaternion reference frame. By rotating the acceleration components into the quaternion frame, the data is now in a two-dimensional form that allows usage of a modified ZUPT algorithm from [2] and [4]. The result of the two-dimensional ZUPT algorithm is positional data for an initial transform for use in a lidar registration method.

Homogenous transformations were also used in this thesis work. Translational and rotational operations were required in a specific format for use by several software packages. The *rigid3d* object utilized in MATLAB is one such program requiring data in a homogenous transform format to allow for various operations to occur [15]. A three-dimensional homogenous transform consists of a 4×4 matrix with a 3×3 rotational matrix and a 3×1 translational vector with additional zero and non-zero padding for proper matrix operations [16]. Homogenous transforms enable rotational and translational operations between two reference frames via matrix operations and are explained in greater detail in [14] and [16]. A generic homogenous transform matrix is of the form

$$ {}^{q}_{c}T = \begin{bmatrix} & R_{3\times3} & & p_{3\times1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7} $$

where the subscript and superscript notation indicates a transformation from the quaternion reference frame to the cart reference frame.

C. ZERO VELOCITY UPDATE

For determining positional data from acceleration, a ZUPT algorithm similar to [2] and [4] was utilized for correction of errors on collected acceleration data. The work done in this thesis research was constrained to allow for known start and stop intervals. This enabled application of the ZUPT algorithm since the beginning and ending acceleration data had a known zero velocity. Without the ZUPT algorithm, direct integration of acceleration data leads to large errors as shown in previous work done in [2] and [4]. The

modification of the ZUPT algorithm into two dimensions is discussed in more detail later in this thesis research.

The ZUPT works by correcting the sensor bias and error propagation throughout the collected data, an inherent issue with inertial measurement units. Given the actual acceleration term, denoted as $a_a(t)$, with an error term over the time interval called ε, from [4], the total acceleration value from the IMU sensor is

$$a(t) = a_a(t) + \varepsilon, \qquad t = [0, T]. \tag{8}$$

From [4], after integration, the error term will carry over across the entire period into the velocity term as

$$v(t) = v_a(t) + \varepsilon t, \qquad t = [0, T]. \tag{9}$$

To correct this, it is known at $t = T$ that $v_a(t) = 0$ m/s since the cart is motionless. Thus, from [4], the error term over the entire period reduces to

$$\varepsilon = \frac{v(T)}{T}. \tag{10}$$

Substitution of Equation (10) into Equation (9), from [4], and reordering leads to the corrected velocity term

$$v_a(t) = v(t) - \frac{v(T)}{T} t, \qquad t = [0, T]. \tag{11}$$

The corrected velocity in Equation (11) can then be integrated to obtain position data.

Implementing the ZUPT algorithm requires selection of acceleration points that correspond to zero velocities. In [4], the process for selecting zero velocity points in acceleration data was automated via flagging changes in angular velocity from a human gait. However, the hardware configuration in this thesis is aboard a cart that limits angular

motion and precludes using gait angular velocity changes for detecting zero velocity points. Rather, as done in [2], the acceleration corresponding to zero velocity points is manually selected. This manual windowing process is shown in Figure 3 for one-dimensional acceleration data in the sensor frame which has been collected over one meter of distance travelled in a straight line.

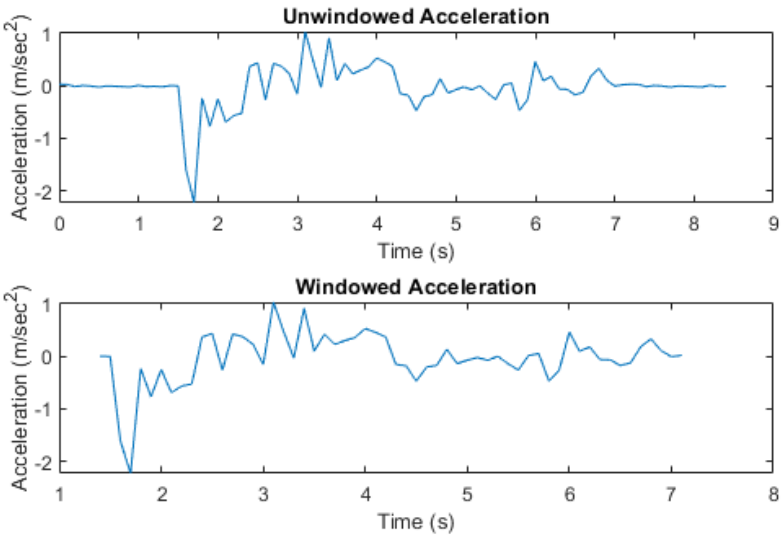

Figure 3. Before and after windowing of sensor acceleration

In Figure 3, the windowing of the data ensures a more accurate localization from the ZUPT algorithm. Without windowing, after double integration through the ZUPT algorithm, greater positional error was exhibited than with windowing.

After windowing of the acceleration, the acceleration data is integrated and corrected with the ZUPT algorithm. In Figure 4, the uncorrected velocity in the sensor frame and corrected velocity after application of the ZUPT algorithm to correct the velocity are shown. The IMU sensor has travelled one meter in a straight line. Note that the corrected velocity is now shifted to zero at the start and end points since the points are known to have zero velocity from experimental constraints.

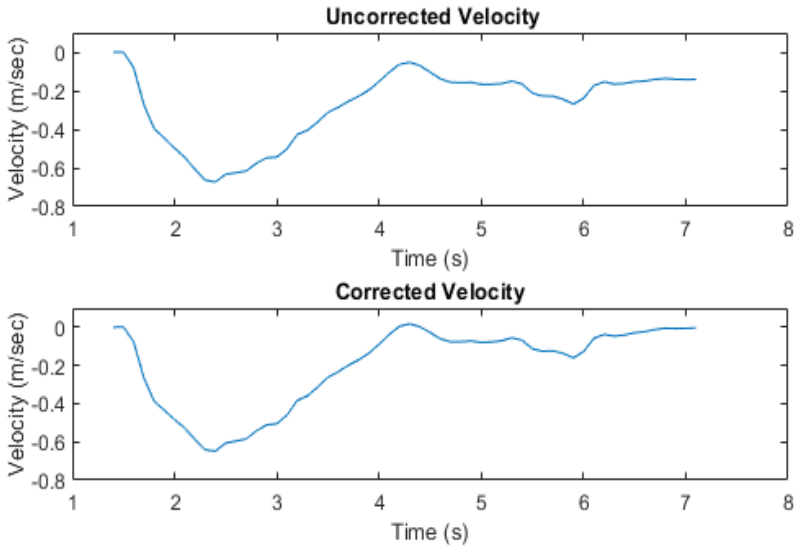

Figure 4. Before and after ZUPT for sensor velocity

After the ZUPT algorithm is applied to correct the IMU velocity in the sensor frame, the data is integrated a final time to generate a positional estimate. A comparison is shown in Figure 5 of the generated positional data without the ZUPT algorithm and with the ZUPT algorithm.

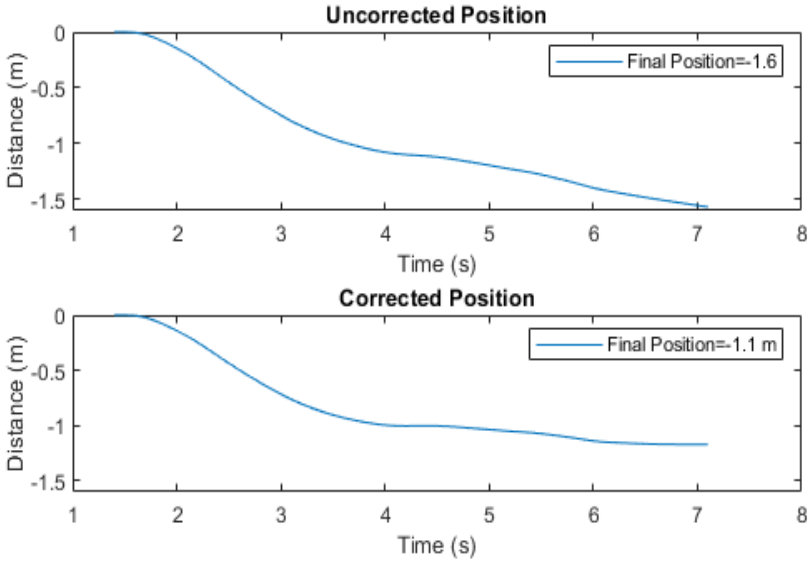

Figure 5. With and without ZUPT for sensor position

The uncorrected position shows the accumulation of drift leading to larger errors in final computed position compared to the ZUPT method. Of note for Figure 5, the acceleration, velocity, and position are negative due to the orientation of the IMU sensor within the sensor frame.

THIS PAGE INTENTIONALLY LEFT BLANK

III. HARDWARE AND SOFTWARE

Hardware utilized in this work consisted of an IMU sensor, three-dimensional lidar sensor, a mobile cart platform, and a laptop computer. A description of each component is discussed in greater detail within this chapter.

A. HARDWARE

1. Lord MicroStrain 3DM-GX5-25

The IMU sensor utilized in this research was the Lord MicroStrain 3DM-GX5-25. This system contains triaxial accelerometers, gyroscopes, and magnetometers, and a pressure altimeter and temperature sensor within a small package less than 36 mm × 37 mm. Shown, in Figure 6, is the MicroStrain 3DM-GX25-25 used in this thesis research.

Figure 6. MicroStrain 3DM-GX5-25

The sensor is capable of sampling acceleration at up to 4 kHz for the gyroscope and 1 kHz for the accelerometer. However, the MicroStrain sensor implements an onboard Extended Kalman Filter (EKF) for bias correction and other computations for acceleration, angular velocity, and quaternion generation that limits data output to 500 Hz [17]. For experimentation, the data output with the EKF was set to 100 Hz, the default sample rate set by the software collecting data. The MicroStrain 3DM-GX5-25 sensor is an updated model from what was used in [2] and was powered from a laptop.

2. Velodyne Lidar Puck

The 3D lidar sensor used in this thesis research was a Velodyne Lidar Puck. The Velodyne Puck has 16 laser channels operating at 903 nm that are capable of returning range and intensity of reflectivity for each laser return [18]. The range accuracy for each range point is ±3 cm. The sensor is capable of operating in single return mode as default or dual return mode for operation in restrictive environments such as foliage or dust. For this research, the lidar was set to single return mode as experimentation was done indoors. The vertical field of view (FOV) is 22º and the horizontal FOV is 360º. For capture of 360º horizontal FOV, the 16-sensor array physically rotates between 5 and 20 Hz inside the main housing. Data for this research was collected at 10 Hz, the default value. One full scan of the lidar produces over 28,000 individual range return points in cartesian coordinates as well as an equal number of intensity values for each point. Shown, in Figure 7, is the Velodyne Lidar Puck used in this thesis research. The sensor was powered from an extension cord trailing behind the cart.

Figure 7. Velodyne Lidar Puck

3. Cart Configuration

The Velodyne Lidar Puck and MicroStrain sensor were both fixed to a movable cart as shown in Figure 8. This enabled mobile data collection and eliminated many of the

additional hardware and software requirements for implementation on a robotic platform as done in [9].

Figure 8. Cart with lidar and MicroStrain sensors

The Velodyne was mounted on the long symmetrical axis of the cart with the MicroStrain offset from the sensor to originally allow for limited interference of magnetometer measurements. The MicroStrain sensor is mounted 19 cm × 7 cm from the Velodyne Lidar Puck origin. The overall cart dimensions are 92 cm × 83 cm × 65 cm.

4. DELL Latitude 3560 Laptop

A Dell Latitude 3560 laptop was used for data collection and processing. The Dell laptop had an Intel i3-5005U CPU operating at 2 GHz with 16 GB of memory. Ubuntu 20.04 was used for the operating system in conjunction with MATLAB 2021a. During data collection, the laptop was placed behind the lidar and MicroStrain sensor and operated off internal battery power.

B. SOFTWARE

Multiple types of software exist for interfacing with the lidar and IMU sensors with each having its own benefits and costs. Since MATLAB has a robust suite of tools and programs available for lidar and IMU processing, this makes it an ideal software to use without writing completely new code. Additionally, thesis work done previously in [2] and [9] relied on MATLAB as well which allowed for quicker adaptation of previous work into this thesis research. Robotic Operating System (ROS) was also used in this work for connecting sensors and systems for collecting data.

1. MATLAB

Work done in this research was implemented on MathWorks MATLAB Release 2021a. MATLAB is a software preferred for use in engineering research due to its matrix-based computational language and the robust library of algorithms for different platforms and applications [19]. MATLAB has algorithms designed for collecting and reading data packets from ROS which were used for experimentation collection with the MicroStrain sensor and the Velodyne Lidar Puck. As Velodyne is a lidar product increasingly utilized by many applications, MATLAB contains developed packages for quick integration of lidar without a separate operating ROS node. Toolboxes from MATLAB used in this research include ROS, Navigation, Mapping, Computer Vision, Automated Driving, and Sensor Fusion and Tracking.

2. Robotic Operating System (ROS)

ROS was utilized in this research for connecting various sensors and systems together for experimentation. ROS is a communications and robotics coding framework that standardizes communications traffic between sensors and platforms and provides tools and algorithms for implementation of robotics applications [20]. The MicroStrain sensor has native software available but was unable to directly integrate with MATLAB. Instead, ROS was utilized to connect the sensor to MATLAB. The Velodyne Lidar Puck had direct support in the MATLAB ROS toolbox and did not require installing additional ROS packages. The MicroStrain sensor required a separately running ROS node and the

installation of an external ROS package, *ros_mscl*, to integrate into MATLAB. The ROS version utilized was Noetic with Ubuntu 20.04 installed on the Dell Latitude 3560 laptop.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. FUSION ALGORITHM COMPONENTS

The fusion algorithm developed in this work consists of four main components: the kinematic trajectory algorithm, ZUPT algorithm, the lidar point-cloud registration algorithm, and the lidar point-cloud mapping algorithm. Each component is discussed in greater detail along with brief summaries of alternate methods or solutions investigated for several of the components.

A. KINEMATIC TRAJECTORY ALGORITHM

Implementation of the modified ZUPT algorithm from [2] and [4] required a quaternion array for proper transformation of data as discussed in Chapter II. The IMU sensor orientation on the cart and limiting of cart movement to only forward motion and rotation led to collection of sensor frame x-axis acceleration data only. To reorient the acceleration data into the quaternion frame for future integration required use of quaternions to conduct the transformation. Previous work done in [21] on Factored Quaternion Algorithms (FQA) was implemented in [2] and [4] and could have been applied to this thesis research with modification. However, FQA was not explored for quaternion generation due to the extensive work already conducted with the method in [2], [4], and [21].

An algorithm from the MATLAB Navigation toolbox designed to generate trajectories and quaternions from linear acceleration and angular velocity inputs was explored instead. Titled *kinematicTrajectory* (KT), the algorithm was utilized for quaternion generation since it was a previously untested method for quaternion generation within the lab and offered quick implementation via MATLAB [22]. However, the KT algorithm was proprietary to MATLAB, and the source code for the KT algorithm was unavailable for analysis to determine what methodology for quaternion generation was occurring. To verify the viability of the KT quaternion, a left turn test track was created where the cart was moved forward one meter, rotated 90^{o} counterclockwise (CCW) to a new heading, and then moved one meter along the new heading. Euler angles were then produced from the generated quaternions to see if the angles matched the cart movement

and rotation. For simplicity, the test track was standardized for experimentation and is referred to as the standard left turn track throughout this work. For comparison of methods, the MicroStrain IMU sensor quaternion generation function was also tested. Shown, in Figure 9, are the Euler angles of the MicroStrain EKF quaternion produced from the left turn track.

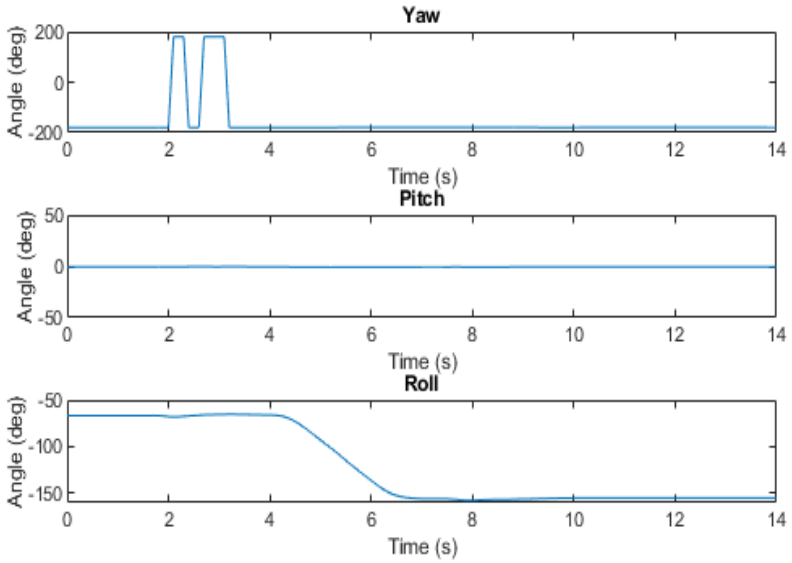

Figure 9. Euler angles from MicroStrain EKF quaternion

Since the cart was rotated 90° about the z-axis of the sensor and quaternion reference frames, a negative 90° yaw angle would have been expected from the MicroStrain EKF quaternion. However, the MicroStrain EKF quaternion produced erroneous Euler angles with the yaw angle indicating sign flip errors within the quaternion scalar value. Sign flip errors can indicate a trigonometric division by zero; since the MicroStrain EKF quaternion code is proprietary, confirmation of this hypothesis was not possible. The Microstrain EKF quaternion roll angle, corresponding to rotation about the x-axis, was also erroneous as the cart was physically incapable of rotating around that axis. For comparison, shown in Figure 10 are Euler angles produced from the KT quaternion.

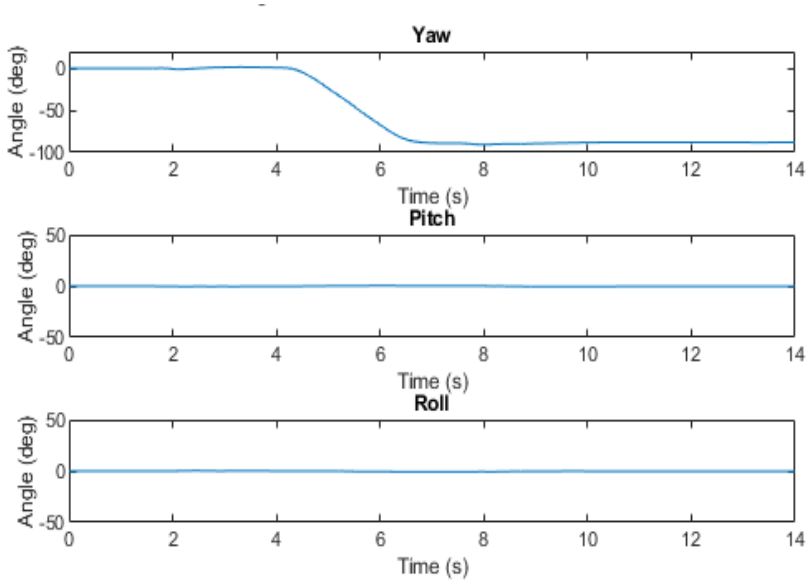

Figure 10. Euler angles from KT quaternion

In contrast to the MicroStrain EKF quaternion, using the same data and left turn track, the KT algorithm produced Euler angles correctly corresponding to cart movement. The KT algorithm did occasionally produce sign flip errors as in the MicroStrain quaternion, however, this was due to the scalar quaternion values approaching zero at a few data points and erroneously being converted with a MATLAB toolbox function. A simple smoothing filter was implemented to correct near zero scalar quaternion data points. Once corrected, the KT algorithm was robust enough to provide a quaternion solution usable in the ZUPT algorithm.

B. ZERO VELOCITY UPDATE

To generate positional data required integrating the IMU acceleration data to find position. Previous work in [2] and [4] used a strap-down zero velocity update technique that was well documented for generating positional data. However, the MATLAB KT algorithm was also capable of generating position from sensor frame acceleration and angular velocity inputs [22]. As with the quaternion generation, the methodology and code for position estimation within the KT algorithm was unknown and proprietary. Testing, however, demonstrated that the KT algorithm highly depended on cart speed. Shown, in

Figure 11, is a comparison on the left turn test track of the KT algorithm position output at two different cart speeds.

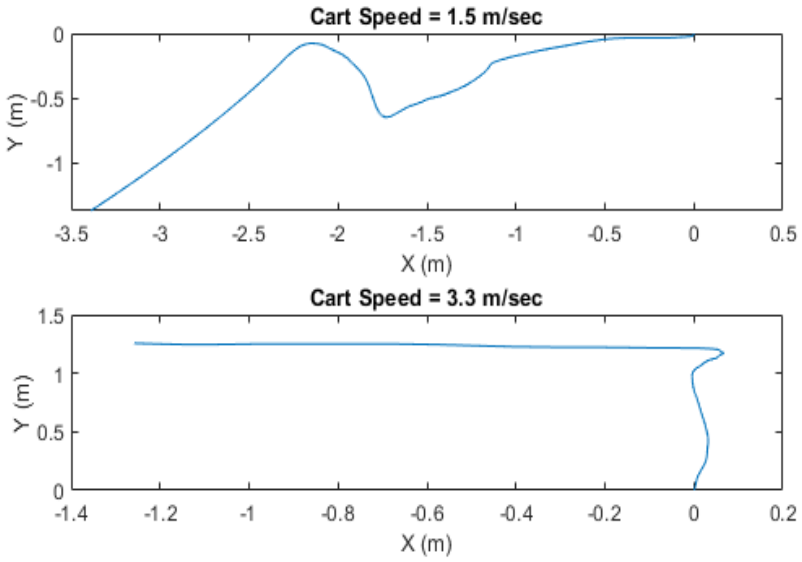

Figure 11. KT algorithm results at two different cart speeds

At faster cart speeds, the KT algorithm was more accurate in final location determination as seen in the lower half of Figure 11 where the left turn test track is more clearly defined. However, a high cart speed was undesirable since it was deemed unlikely that a cart or robot using this research in the future would travel at higher speeds in normal operating conditions. In contrast, the ZUPT algorithm proved to be more robust at slower speeds and produced more consistent positional data. Shown, in Figure 12, is the ZUPT algorithm positional plot on the same left turn track as used with the KT algorithm.

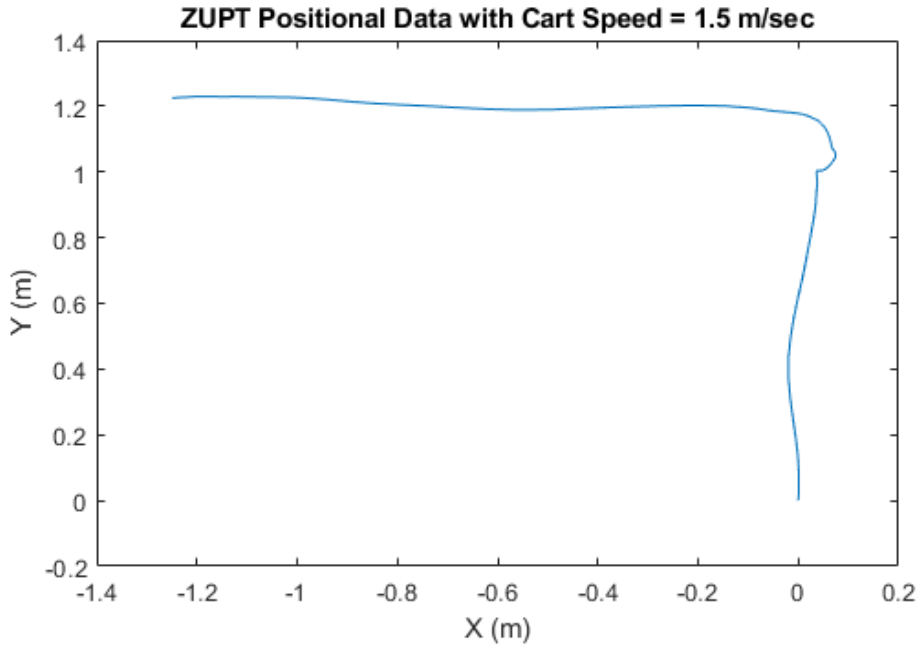

Figure 12. ZUPT algorithm on left turn test track

For comparison of consistency between both algorithms, several data collection runs on the left turn track were conducted. Illustrated in Table 1 are the results of the KT algorithm positional data versus the ZUPT algorithm positional data at the same cart speed on the standard left turn test track.

Table 1. Comparison of methods for calculated distance traveled

	KT		ZUPT	
	Final X	Final Y	Final X	Final Y
Run 1	-4.322	-1.241	-0.901	0.469
Run 2	-6.236	-2.744	-0.737	0.94
Run 3	-3.564	-0.547	-0.849	0.727
Run 4	-2.257	-2.064	-0.808	1.336
Run 5	-3.389	-1.371	-1.245	1.229
Mean	-3.9536	-1.5934	-0.908	0.9402
SD	1.474	0.838	0.197	0.356
Variance	2.173	0.703	0.039	0.126

The high variance and inaccuracy of the positional data generated from the KT algorithm led to selection of the ZUPT algorithm for use in generating position estimates from the IMU sensor. The ZUPT algorithm was expanded from work done in [2] and [4] to allow for application on a cart moving two-dimensionally. Expansion into two dimensions required rotation of the windowed IMU sensor acceleration into the quaternion frame using the KT quaternion. Once rotated into the quaternion frame, the acceleration data was integrated and corrected with the ZUPT algorithm. Further integration generated corrected position data for use in the fusion algorithm.

C. LIDAR POINT-CLOUD REGISTRATION

Velodyne three-dimensional lidar scans were stored in MATLAB as point-cloud objects which contained the x, y, and z coordinates for each point in addition to intensity values [23]. Successive point-clouds from the lidar required registration in order to obtain transformations for building a three-dimensional world map. Several methods exist for registering point-clouds in order to obtain the required three-dimensional transforms for mapping. The three methods covered in this work are the iterative closest point, normal distribution transform, and coherent point drift registration methods.

Point-cloud registrations can be computed as either rigid, affine, or nonrigid in MATLAB. Rigid transforms retain the shape and size of the scene in addition to applying the same transform to all points in the scene [9], [24]. In this work, only rigid transforms were used since the point-clouds collected were relatively short range and did not exhibit blurring from the motion of the cart. Affine transforms in addition to allowing translation and rotation, also allow for shear and change of scale of objects. Non-rigid transforms allow for individual points in a point-cloud to undergo distinct transforms [24].

1. Iterative Closest Point Method

The Iterative Closest Point Method (ICP) for registering point-clouds is a popular approach to registration of point-clouds and was used in [9], [25], and [22]. The ICP algorithm follows direct from [9], [25], and [26]:

1. Find the closest points between two sets of points in three-dimensional space.
2. Find the rigid transformation that minimizes the distance between the corresponding points in the first and second point sets.
3. Apply the generated transformation to the first set.
4. Continue steps one through three until convergence is achieved at a defined tolerance level or no change occurs.

From [9], The ICP method relies on two assumptions: first that the point-clouds are of the same environment, and second that two consecutive point-clouds are not too far apart. While the first is intuitive, the boundaries of the second assumption are challenged in this work by fusion with additional sensor inputs. For this work, the MATLAB Computer Vision toolbox function for ICP registration was utilized. The algorithm has greater computational effectiveness when point matching since it utilizes k-dimension trees (KD tree) searches for decreasing point correspondence computations and indexing [13]. Both [9] and [26] discuss KD tree data structures in greater detail and the benefits of using them. ICP registration between point-clouds alone over a large separation or rotation can lead to matching degeneration, a limitation where the ICP algorithm fails to achieve the correct convergence solution [9], [26]. Limitations of ICP are the assumptions that the closest point is the corresponding point, which may not always be true, and that an initial transformation estimate is required to eventually achieve convergence [9]. This initial estimate can be zero if the scans are assumed to be taken at a constant velocity with no large rotations occurring between each scan. Despite these limitations, the ICP method is computationally cheaper than other methods, and it is easier to implement making it a popular choice for point-cloud registrations [9], [26].

2. Normal Distribution Transform Method

Another registration method explored in this work was the Normal Distribution Transform (NDT). A two-dimensional application was proposed in [27]; the NDT method was expanded into three-dimensions by [28]. The NDT method relies on subdividing the space into voxels, cubes in three-dimensional space, to find correspondence between two point-cloud scans. A probability distribution function from the points within each voxel of

the first scan is then generated and used for matching between the scans [9]. The NDT algorithm step summary follows directly from [9] and [27]:

1. For two sets of points in three-dimensional space, divide into equal sized voxels containing multiple points.
2. Find a normal distribution for each voxel in the first scan.
3. Give an initial estimate of transform parameters.
4. Map each point in the second scan to the first scan coordinate frame using the initial estimated transform parameters.
5. Calculate the normal distribution of the mapped points.
6. Calculate the score for the transformation parameters by evaluation of the distribution and summing the results.
7. Calculate a new parameter estimate by optimizing the score via Newton's Algorithm.
8. Continue step three through seven until convergence is achieved.

Like ICP, the NDT requires an initial transform estimate, which can be zero or non-zero, to work. For this research, the NDT algorithm explored came from the MATLAB Computer Vision Toolbox.

3. Coherent Point Drift Method

The Coherent Point Drift (CPD) method differs from ICP and NDT since it is capable of rigid, affine, and non-rigid transformation between two point-cloud scans. Additionally, the CPD method does not require an initial transform estimate to function [9], [29]. The CPD method uses Gaussian Mixture Model centroids for computing transforms between two point-cloud scans and is computationally slower than ICP or NDT as found in [9]. The CPD method is explained in detail in [29]. The CPD algorithm explored in this work came from the MATLAB Computer Vision Toolbox.

4. Comparison of Point-Cloud Registration Methods

To obtain a better understanding of the performance and capabilities of the three registration methods, the left turn track experiment was used for collecting data. A series of point-clouds from the lidar sensor were collected and processed using the three different registration techniques and then mapped to a world map frame. Shown, in Figure 13, is an accumulated point-cloud on a left turn track after registration with the NDT method and mapping.

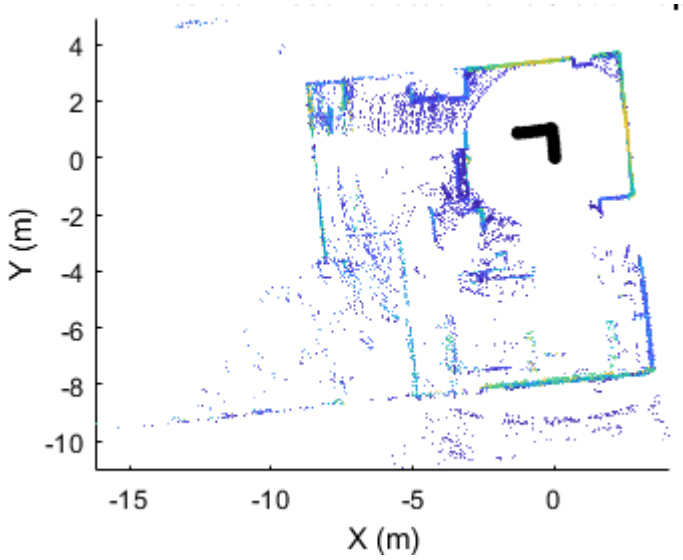

Figure 13. NDT accumulated point-cloud map

Visually none of the three registration methods showed a significant difference in localization and mapping ability. However, tabulated in Table 2 are the final position results over five data collection runs of registration with the three methods followed by mapping. First implemented in the ZUPT algorithm alone experimentation, the standard left turn track was used again since rotation was desired in addition to translation for better understanding registration performance. Each collection run is 400 point-clouds that are down-sampled to 26 point-clouds for better registration and mapping effectiveness. Down-sampling and tuning of parameters was done for each registration method.

Table 2. Final position after registration and mapping of only point-clouds

	ICP		NDT		CPD	
	X (m)	Y (m)	X (m)	Y (m)	X (m)	Y (m)
Run 1	-1.172	0.693	-1.18	0.892	1.05	0.745
Run 2	-1.372	0.725	-1.29	0.875	-1.12	0.8
Run 3	-1.22	0.783	-1.256	0.823	-1.104	0.818
Run 4	-1.282	0.779	-1.356	0.743	-4.22	0.726
Run 5	-1.099	0.586	-1.19	0.695	-1.08	0.62
Mean	-1.229	0.713	-1.254	0.805	-1.294	0.741
SD	0.104	0.080	0.072	0.084	1.882	0.077
Variance	0.010	0.006	0.005	0.007	3.541	0.006

For localization, NDT shows marginally better performance for computing the true final position of negative one meter in the x-dimension and one meter in the y-dimension. However, results from [9] on comparison of registration processing times led to selection of the ICP registration algorithm for use in the fusion algorithm. While computational times were not experimentally confirmed against known benchmarks, as done in [9], the processing times observed during experiments done in this work matched the results of [9]. The ICP method processed point-clouds faster than the NDT method which in turn ran faster than the CPD method.

D. POINT-CLOUD MAP BUILDING

Map building from point-clouds consists of applying a series of accumulated absolute transforms to each sequential point-cloud scan in order to align to the world frame. From [9], the k^{th}absolute transform is computed as

$$ {}^{c}_{w}T_{n} = {}^{c_n}_{c_{n-1}}T \, {}^{c_{n-1}}_{w}T_{n-1} \tag{12} $$

where ${}^{c_n}_{c_{n-1}}T$ is the homogeneous transform output from registration of the current point-cloud scan to the previous scan. ${}^{c_{n-1}}_{w}T_{n-1}$ is the previous cumulative absolute transform for

mapping the previous point-cloud scan from the cart frame to the world frame. Once each scan has been transformed into the world frame, it is added to the world map.

Several methods exist for building a map from the aligned point-clouds that are explained in [9], [30], and [31]. For mapping in this work, a similar method as [9] was used. First, a plane fitted ground removal algorithm was used on each point-cloud to enable better registration algorithm performance. The processed point-clouds are then registered, and an absolute transform is computed as shown in Equation (12). The point-cloud is then merged into a world map. Since mapping was not the primary focus of this research, the entire mapping process is computed using the MATLAB Computer Vision Toolbox function scripts and tutorials in [10].

E. FUSION ALGORITHM

The main objective of this work was to combine sensor inputs from two different sensors to create a computationally more cost-effective algorithm for SLAM. This was done by combining inputs from the MicroStrain IMU sensor and Velodyne lidar. First the IMU acceleration and angular velocity inputs are inputted into the KT algorithm to generate a quaternion. The quaternion is then used for rotating the IMU sensor frame acceleration data into the quaternion frame which is then inputted into the ZUPT algorithm to generate an estimate of the cart position. The position estimate is combined with the rotation matrix generated from the KT quaternion into a homogenous transform and passed, as an initial transform estimate between two point-clouds, to the ICP registration algorithm. The registration algorithm iteratively generates a refined transform until convergence is achieved. The final refined transform from the ICP registration is then applied to a mapping algorithm to generate a world map of the accumulated point-clouds and a final position in the world frame. Shown, in Figure 14, is the flow chart of the fusion algorithm to generate the final position. ${}^{c}_{q}T$ is a fixed homogenous matrix that rotates and translates data from the quaternion frame to the cart frame.

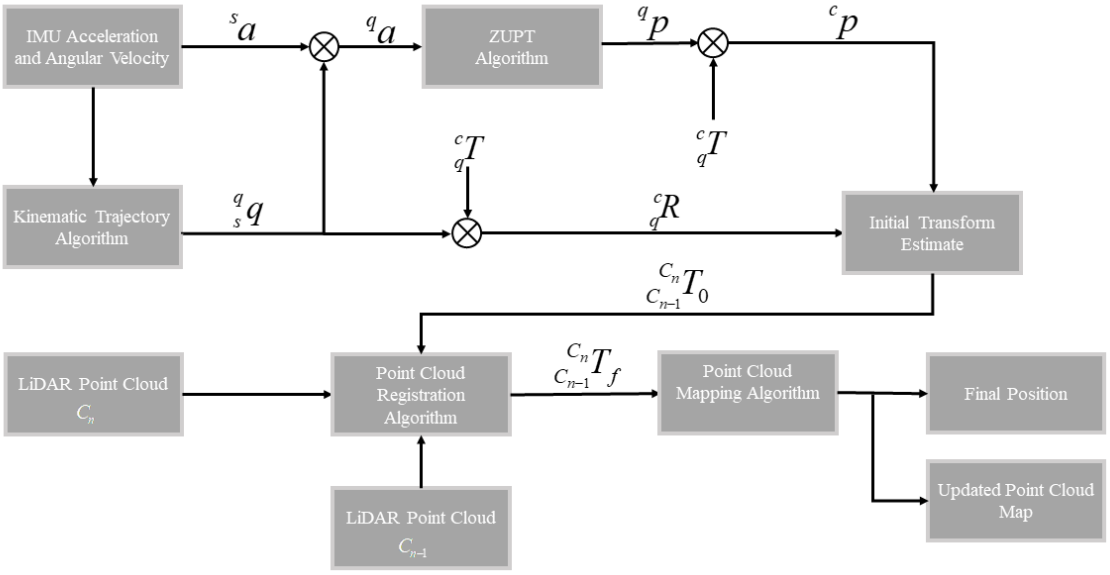

Figure 14. IMU and lidar fusion algorithm

The fusion algorithm acts as a basic complementary filter, combining the advantages of a lidar and an IMU sensor to counteract the disadvantages of each alone, to provide better registration and mapping results at a reduced computational processing cost. Lidar scans offer the capability to map and sense objects within the surrounding environment at a higher computational cost when processing. The IMU has accuracy in shorter time durations, and is often used in such a time scale, but suffers from significant drift over longer time periods. The ZUPT algorithm corrects the IMU drift error and enables generation of accurate transform estimates over time from the IMU. Following IMU drift error correction and transform generation with the ZUPT algorithm, the ZUPT transform estimate was fed into the point-cloud registration algorithm. The transform estimate from the IMU reduced the number of point-clouds required for localization and mapping. The reduction in required point-clouds in turn decreased computational processing requirements for SLAM, a goal of this work.

V. RESULTS

This chapter includes a summary of results for the experiments conducted throughout this work. Results from the fusion algorithm are included in addition to results from expansion of the fusion algorithm into a multi-step program for future expansion.

A. FUSION ALGORITHM

Without an initial estimate from the IMU sensor data, the ICP registration and mapping algorithms required consecutive scans to accurately register, localize, and map the point-clouds after significant rotation and translation as shown in Figure 15. In Figure 15 are 26 point-clouds mapped using ICP registration on a left turn track after downsampling point-cloud scans from 400 point-clouds. Downsampling point-clouds further than 26 led to drift and matching degeneration between successive scans on the map. A method to achieve computational cost reduction is decreasing the total amount of point-clouds required for registration. Ideally, the minimum point-clouds required would be two point-cloud scans. However, without additional sensor inputs, the registration algorithm fails to accurately compute a cumulative absolute transform without a sufficient number of point-clouds. Shown, in Figure 16, is a before and after registration of two point-cloud scans without use of the fusion algorithm.

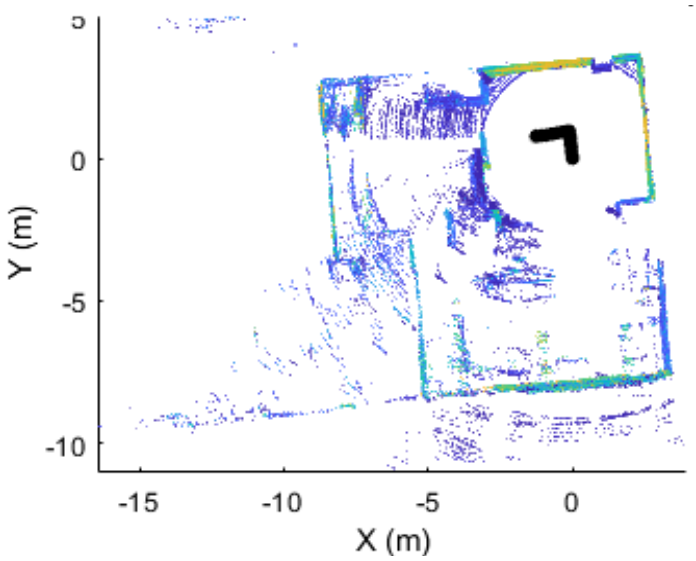

Figure 15. ICP accumulated point-cloud map

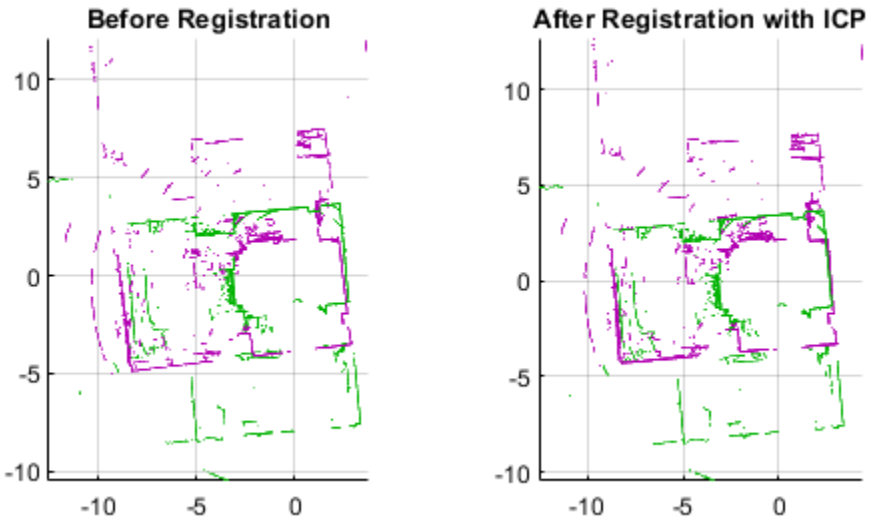

Figure 16. Registration of two scans without fusion

The first point-cloud, green, was taken on initialization of the lidar. The second point-cloud, purple, was taken after the lidar sensor has been translated and rotated on a one meter by one meter 90º left hand turn track as seen in Figure 15. In Figure 16, the ICP algorithm incorrectly selects the closest points as the correct points to match and outputs an incorrect transform, failing to rotate and translate the second point-cloud properly. This

result, if inputted into the mapping algorithm, produces unusable localization and map results.

By using the ZUPT algorithm to provide an initial estimate of motion occurring between the two point-clouds, the fusion algorithm can reduce the number of required points clouds needed for SLAM. Results are shown in Figure 17 where two point-clouds are displayed before and after using the fusion algorithm for registration. One point-cloud, purple, has been collected after the lidar sensor has been translated and rotation on the left turn track.

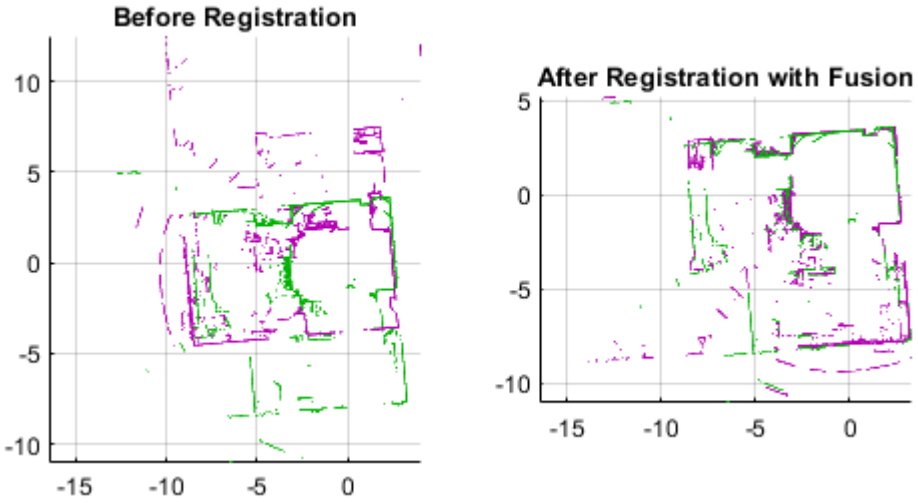

Figure 17. Registration of two point-clouds with fusion

Combining the results shown in Figure 17 with the mapping algorithm produces the desired accumulated point-cloud map that required only two point-clouds versus the 26 point-clouds from Figure 15. Shown, in Figure 18, is the localization and mapping result of the fusion algorithm with only two point-clouds.

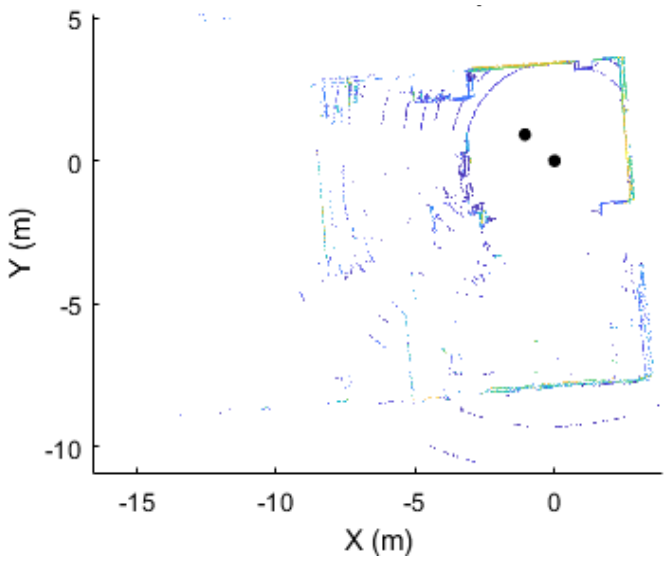

Figure 18. Accumulated point-cloud map after fusion

Empirical comparison of the fusion algorithm localization results with the ZUPT and ICP algorithms alone was also explored. The results of the comparison between each separate localization method are tabulated in Table 3. Fusion produced localization results more consistent than the ZUPT algorithm and more accurate and consistent results than the ICP registration algorithm alone when not comparing point-clouds utilized. The ICP method alone required 26 point-clouds to achieve localization after a left turn versus the two point-clouds needed with the fusion algorithm.

Table 3. Comparison of localization methods

	ZUPT		ICP		Fusion	
	X (m)	Y (m)	X (m)	Y (m)	X (m)	Y (m)
Run 1	-1.067	0.356	-1.172	0.693	-1.064	0.915
Run 2	-0.686	0.936	-1.372	0.725	-1.106	0.940
Run 3	-0.840	0.651	-1.220	0.783	-1.201	1.012
Run 4	-0.660	1.232	-1.282	0.779	-1.286	0.863
Run 5	-1.225	1.192	-1.099	0.586	-1.149	0.779
Mean	-0.992	0.873	-1.231	0.713	-1.151	0.892
SD	0.277	0.371	0.120	0.081	0.096	0.098
Variance	0.076	0.137	0.014	0.006	0.009	0.001
Point-Clouds Used	0		26		2	

B. MULTI-STEP FUSION ALGORITHM

To further the application of this method and explore limitations, the fusion algorithm was expanded to enable localization and mapping around the ECE Controls Laboratory in multiple steps. To compare outputs, the ICP registration alone and the fusion algorithm were used around the same path in the laboratory. Shown, in Figure 19, is the point-cloud map of the ECE Controls Laboratory using the ICP registration alone. A total of 67 point-clouds were needed to map the laboratory after down-sampling from 1000 point-clouds. Drift between point-clouds is visually present as seen in the misaligned walls of the room between each scan. The break in the localization of the cart seen in Figure 19 comes from stitching two sets of point-clouds together. The extension cord powering the lidar sensor could not traverse the entire path around the laboratory.

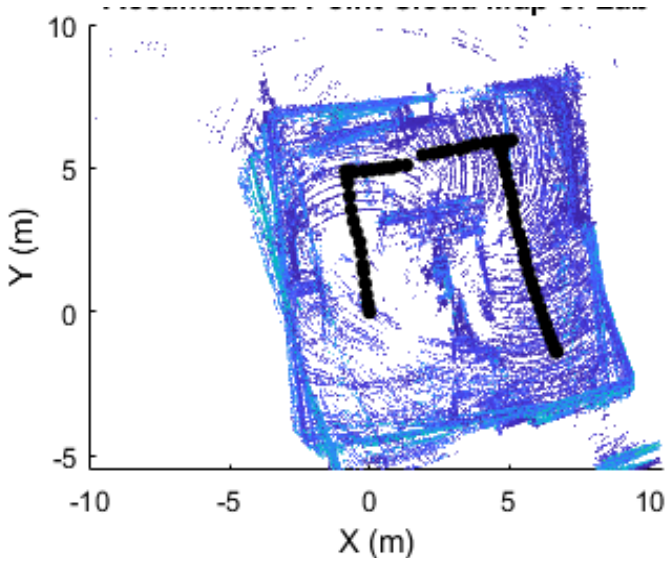

Figure 19. ICP registration alone map of laboratory

Since the expansion of the fusion algorithm was a proof of concept, the maximum distance the fusion algorithm was effective operating on IMU data between lidar scans was not fully explored. When collecting the multi-step fusion algorithm data, incremental stops around the laboratory were conducted where, between each stop, the IMU data was collected, and, at each stop, a lidar point-cloud scan was collected. This simulated a robot navigating via IMU between each stop and using a lidar at each stop to collect a scan for mapping and localization with respect to the environment. Such a process negates running continuous scans of the room with the lidar sensor while the cart is in motion. However, re-initializing of each sensor at the appropriate time in the process is required, a step that can be accomplished via automation on a robotic platform. Since data was collected for post processing, the IMU data was stitched together and manually windowed during the running of the multi-step fusion script. Shown, in Figure 20, is the multi-step fusion algorithm point-cloud map of the ECE Controls Laboratory that uses 13 point-cloud scans to fully map the room.

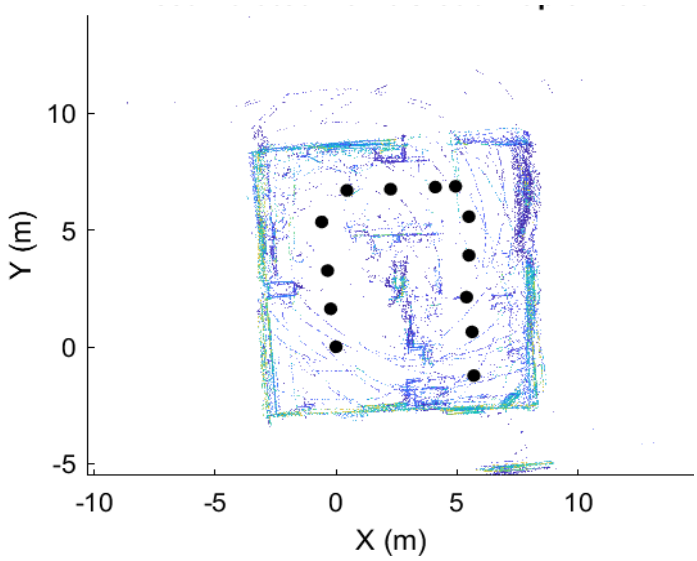

Figure 20. Multi-step fusion algorithm map of laboratory

Whereas the ICP registration algorithm took many point-clouds to map the lab, the multi-step fusion algorithm took far less point-clouds to achieve localization and mapping. The results are tabulated in Table 4 for the multi-step fusion algorithm conducted in the ECE Controls Laboratory. Point-cloud mismatch and drift errors still occur with the multi-step fusion algorithm. However, compared to the ICP alone mapping results in Figure 19, the drift and mismatch errors seen in Figure 20 for the multi-step fusion algorithm results are visually less than the ICP alone registration results.

Table 4. Scans required for localization and mapping of laboratory

	ICP	Fusion	% Reduction
Point-Cloud Scans	67	13	515%

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION

As the DOD increasingly operates in urbanized areas around the world, solutions for autonomous robotics capable of navigation indoors and outdoors will become increasingly important in tactical environments. Portability and power requirements are important design considerations. Lidar offers a promising solution for future navigation solutions but requires processing large data sets. Onboard processing of large data sets increases platform weight with processors and power supplies and is undesirable. Remote processing of large data sets, however, requires higher bandwidth for communication and is not always reliable in tactical scenarios. Finding a solution for reducing processing requirements is therefore required. An algorithm combining inputs from multiple sensors to achieve a more computationally efficient process for point-cloud registration was formulated in this thesis research. Within this chapter, assessment of the goals of this thesis are covered as well as limitations of the fusion algorithm and recommendations for future work.

A. ASSESSMENT OF GOALS

This work had three goals. First, explore the capabilities and effectiveness of the ZUPT algorithm and lidar point-cloud registration algorithm for localization. Second, determine if combining the ZUPT algorithm and lidar point-cloud registration algorithms was feasible. The final goal was determining if computational processing reductions could be achieved by combining the two methods. All three of these goals were achieved in this work.

While the ZUPT algorithm and lidar point-cloud registration methods were capable of localization in two-dimension, each had shortcomings. For the ZUPT algorithm, mapping of the surrounding environment was not possible due to the nature of the IMU sensor. Localization of the cart was the main output from the ZUPT algorithm. Additionally, use of the ZUPT algorithm is only possible at zero velocity points, requiring the platform to make occasional stops. For the lidar point-cloud registration methods, the three methods explored differed in the computational time to register point-clouds as found

in [9]. After tailoring parameters in each method, SLAM results from the ICP, NDT, and CPD methods were comparable for this application. The advantage of using point-cloud registrations as compared to the ZUPT algorithm, however, is that the data from a lidar sensor can be used for SLAM. SLAM allows for better information gathering and decision making in a tactical environment as a robotic platform using SLAM can operate in tactical situations without increasing risk to personnel.

The second and third goals of this research were accomplished as well. Fusion of the ZUPT algorithm and ICP point-cloud registration algorithm was achieved and the results of the fusion algorithm for SLAM were on par or better than previous methods explored. However, since this work was done on a cart, further integration of sensors will likely be required to fully implement this fusion method on a robot in the future. Greater computational efficiency was achieved by reducing the number of point-clouds required for registration, localization, and mapping by almost 500%.

B. LIMITATIONS

The fusion algorithm has several limitations that were discovered or verified in this research. First, the fusion algorithm requires overlapping points between two point-clouds to generate a localization and mapping solution. Second, the ZUPT requires a high enough velocity to differentiate between IMU noise and sensor readings. Thirdly, the fusion method will not work without zero velocity points which could preclude using the fusion algorithm on many autonomous platforms. Finally, the mapping and localization with the point-clouds still exhibits drift over longer distances, albeit much reduced compared to point-cloud registration techniques alone.

The fusion algorithm still relies on the ICP registration algorithm for computing a final transformation before mapping and localization. The ICP registration algorithm still requires similar points between two point-clouds to achieve convergence for a global transformation. If the lidar sensor has a sudden restricted FOV, such as moving through a long and narrow doorway, the ICP registration algorithm will not converge to the correct transform despite an accurate ZUPT transform estimate. This occurs since the ICP algorithm seeks to match the closest points. If points in one point-cloud have close to zero

overlap with the previous point-cloud in the global environment after the ZUPT transformation estimate, the ICP algorithm will incorrectly assume the closest points are correct and continue iterating until convergence criteria are met. Such a situation leads to an incorrect convergence solution with the ICP algorithm that produces an erroneous global transform estimate. Since this thesis was an initial proof of concept on fusing sensor data, quantitative boundaries on the limitation were not fully explored. However, preliminary investigation showed the problem occurred when the lidar scan FOV was restricted to approximately less than 25% of the previous scan FOV along with a corresponding drop in lidar return points. The limitation could potentially be overcome with an algorithm that bypasses the ICP algorithm to temporarily choose the ZUPT transform estimate as the preferred transform for mapping the point-cloud.

As found in [2], the ZUPT algorithm does not work well when platform velocity is too low. With low velocities, the noise of the sensor degrades the ability of the ZUPT algorithm to localize effectively. Expansion of work done in [2] into the fusion algorithm did not remove this limitation.

The third limitation requires the cart to briefly stop in order to effectively localize using the ZUPT algorithm. Not all platforms are designed to operate in such a manner. Additionally, since the multi-step fusion algorithm relies on a series of re-initializations of the IMU and lidar to get the correct data, platforms designed to account for operating in such a start and stop manner are required.

The final limitation can be overcome with loop detection algorithms, which are discussed in greater detail in [9]. Within MATLAB, tools exist for conducting loop detection and correction that could be implemented under the correct circumstances. Increased computational requirements for using such a method are unknown and loop detection is not always a usable solution as [9] found. Loop detection was not explored in depth in this work as building the fusion algorithm was the main focus of this work.

C. FUTURE WORK RECOMMENDATIONS

1. Adaptation to Robotic Platform

This research was limited to application on a push-cart while collecting data. Implementation on a wheeled robotic platform for future use in autonomous navigation is a logical next step in expanding the fusion algorithm usability. The reduced point-cloud requirements could enable better onboard processing or at least reduce bandwidth usage for transmitting data to an external processing unit. Application to a light weight unmanned aerial vehicle is also a potential area of future research.

2. Fusion Algorithm Optimization

During the multi-step fusion algorithm implementation, the maximum distance before degradation of the fusion algorithm was not explored. The fusion algorithm still relies on a point-cloud registration algorithm which is still costly in processing. Algorithms that cost less computationally while achieving localization and mapping are worth exploring. Optical flow techniques on very light weight platforms such as in [32] are potential areas for future exploration to further optimize the fusion algorithm.

3. Integration with Additional Sensors and Methods

Reduction in point-cloud density has the downside of reducing the point-cloud map resolution. Small objects and obstacles will therefore be harder to detect. Integration of vision sensors or sensors capable of machine-learning to recognize obstacles is a recommended area for future work. Raw point-cloud maps are somewhat limited in usefulness. Application of occupancy maps or feature extraction methods for object detection and mapping is a highly recommended area for pursuit in the future.

APPENDIX A. DATA COLLECTION SCRIPTS

A. IMU_COLLECTION.M

```
% This scripts collects data from an IMU sensor in a rossubscriber node
%and saves to user defined destination. Requires initializing a ros
%subscriber node in the Ubuntu terminal before connecting. See Useful
%Links and Ros Startup.doc file for instructions.

clearvars, clc; close all
rosinit %initialize ros node

IMU=rossubscriber('/gx5/nav/filtered_imu/data'); %subscribe to rosnode
iterations=1;      %number of sample runs desired

%% Set initial parameters
samplerate=10;      %sample rate. Change based on sample rate set in
%roslaunch file.

step=samplerate^-1; %step size for time

%initial position for KT algorithm
InitialPosition=[0 0 0];
IVelocity=[0 0 0];
IAcceleration=[0 0 0];
IAngVel=[0 0 0];

%length of captured data size/samplerate=time (sec)
size=150;

%loop For writing multiple files to a single directory
for m=1:iterations

%Build kinematicTrajectory object with specified parameters.
trajectory=kinematicTrajectory('SampleRate',samplerate,'Position',
InitialPosition,'Velocity',IVelocity); %

i=0;
k=1;
while i<=size    %set code break here for resetting cart
clc

IMUdata=receive(IMU);
[IMUquat, IMUtime,IMUangVel,IMUlinAcc]=IMUdatafunc(IMUdata);  % Pulls
Raw data through IMUdatafunc file to enable near postprocessing of data

%kinematicTrajectory function from matlab
[KTposition,KTorientation,KTvelocity,KTacceleration,
KTangularVelocity]=trajectory(IMUlinAcc,IMUangVel);
```

```
% IMU Variables for saving
IMUQuat(k,:)=IMUquat;
IMUTime(k,:)=IMUtime;
IMUAngVel(k,:)=IMUangVel;
IMULinAcc(k,:)=IMUlinAcc;


%KT variables for saving
KTPosition(k,:)=KTposition;
KTQuat(k,:)=KTorientation;
KTVelocity(k,:)=KTvelocity;
KTAcceleration(k,:)=KTacceleration;
KTAngularVelocity(k,:)=KTangularVelocity;

end
pause(3)

%% Store Data in specified directories
cd %chosen directory
filename=['LIDAR_IMUCombined_Data_Run_2_IMU_',num2str(m),'.mat'];

%Save data in .mat structure with appending of # runs.
Filename=IMUDataAlone_X.mat
save(filename,'IMUQuat','IMUTime','IMUAngVel','IMULinAcc','KTPosition',
'KTQuat','KTVelocity','KTAcceleration','KTAngularVelocity');

clear vars %for next iteration.
end
```

B. IMUDATAFUNCZUPT.M

```
%% Function to order IMU data from ROS node subscriber into usable
%format that can be used in IMU_COLLECTION.m
function [IMUQuat,IMUAngVel,IMULinAcc]=IMUdatafuncZUPT(IMUdata)

%Quaternion Data
IMUQuaternion(:,1)=IMUdata(:,7);
IMUQuaternion(:,2)=IMUdata(:,8);
IMUQuaternion(:,3)=IMUdata(:,9);
IMUQuaternion(:,4)=IMUdata(:,10);

IMUQuata=(quaternion(IMUQuaternion(:,1),IMUQuaternion(:,2),
IMUQuaternion(:,3),IMUQuaternion(:,4)));
IMUQuat=compact(IMUQuata);

% IMU Angular Velocity
IMUAngVel(:,1)=IMUdata(:,4);
IMUAngVel(:,2)=IMUdata(:,5);
IMUAngVel(:,3)=IMUdata(:,6);

% IMU Linear Acceleration
```

```
IMULinAcc(:,1)=IMUdata(:,11);
IMULinAcc(:,2)=IMUdata(:,12);
IMULinAcc(:,3)=IMUdata(:,13)+9.81;   %corrected z acceleration to
remove gravity component

end
```

C. LIDAR_COLLECTION.M

```
%% Script for pulling 3D point-cloud data from Velodyne VLP-16
(PUCK)into a .m file for processing in fusion algorithm and also ICP
alone registration algorithms

clear all, close all; clc

%% initialize lidar
Clouds=velodynelidar('VLP16');
flush(Clouds);

start(Clouds)
L=1000; %number of point-clouds to capture

[PointClouds, timestamps]=read(Clouds,L);
stop(Clouds);

fps = 1/mean(seconds(diff(timestamps))) %determine samplerate

pause(3)
save LIDAR_IMUCombined_Data_LabFullRun_2_LiDAR_Segment_1 PointClouds
timestamps
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. BUILDMAP_ICP.M

```
%%% Takes prerecorded data and creates a point-cloud map using only
point-clouds. Code adapted from [10]

clearvars;clc; close all;

data=load('lidardata_ref_FULLRUNOFLAB_20210422.mat');

stop=400; %if too many point-cloads were collected, use to truncate if
necessary. Drift occurs if lidar sits on one spot too long.

lidarPointClouds=timetable(data.timestamps,data.PointClouds,'VariableNa
mes',{'PointCloud'});

skipFrames=15;  %skip frames to improve drift error
downsamplePercent=0.30; %improves ICP registration. Found to be maximum
downsample allowable for this application.

% create map builder object. this function is a MATLAB function
% that must be included from cited tutorial for code to work.
mapBuilder=helperLidarMapBuildericp('DownsamplePercent',
downsamplePercent,'RegistrationGridStep',1);

%set random number see
rng(0);

closeDisplay=false;
numFrames=height(lidarPointClouds);
tform=rigid3d; %initialize rigid3d object

for n= 1: skipFrames: numFrames%-skipFrames

    % Get the nth point-cloud
    ptCloud=lidarPointClouds.PointCloud(n);


% use transformation from previous iteration as initial estimate for
% current iteration of point-cloud registration. (constant velocity)
       initTform=tform;

% Update map using the point-cloud. updateMapicp is within
% helperLidarMapBuildericp, a MATLAB script.
      tform = updateMapicp(mapBuilder,ptCloud,initTform);

% Update map display. function in helperLidarMapBuilderICP
      updateDisplay(mapBuilder, closeDisplay);
end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. FUSION ALGORITHM SCRIPTS

A. MAIN_FUSION_SCRIPT.M

```
%%% Fusion algorithm takes lidar point-clouds and acceleration inputs
%%% passed through collection scripts and processes and builds a point
%%% cloud map while reducing required number of point-clouds for
%%% accurate localization. Requires ZUPT_TWODIMENSIONAL.m to work along
%%% with helperLidarMapbuildericp from MATLAB
%%% Code adapted directly from [10]

clearvars;clc; close all;

%load collected data
data=load('lidardata_ref_1x1_20210413_lab_5.mat');  %load point-cloud

% %convert to format usable for MATLAB algorithms
lidarPointClouds=timetable(data.timestamps(1:end),data.PointClouds(1:en
d),'VariableNames',{'PointCloud'});

stop=length(data.PointClouds);

startIMU=16;stopIMU=111; %these require manual adjustment for optimal
%ZUPT windowing of acceleration. Passed to D2ZUPT function

% %Set map player limits for later
xlimits=[-10 10];
ylimits=[-10 10];
zlimits=[-5 5];

%%Optional: Align Scans and Mapping for Two Scans Only. Helps to
understand how process works.

skipFrames=400;
frameNum=1;

fixed = lidarPointClouds.PointCloud(frameNum);
moving = lidarPointClouds.PointCloud(skipFrames);

%process pointclouds for removal of ground plane.
fixedProcessed = helperProcessPointCloud(fixed);
movingProcessed = helperProcessPointCloud(moving);

downsamplePercent=0.3; %this variable needs adjustment based on
application. Too much downsampling of the points within the cloud leads
to insufficient data for registering.
fixedDownsampled = pcdownsample(fixedProcessed, 'random',
downsamplePercent);
movingDownsampled = pcdownsample(movingProcessed, 'random'
,downsamplePercent);
```

```
tforminit=ZUPT_TWODIMENSIONAL(filename,startIMU,stopIMU); %Call
%ZUPT_TWODIMENSIONAL that returns a rigid3d object from IMU ZUPT and KT
%algorithms for use with initial ICP registration transform

tform =pcregistericp(movingDownsampled,
fixedDownsampled,'InitialTransform', tforminit); %updated ICP transform
based off initial guess.
movingReg=pctransform(movingProcessed, tform);

% Plots of before and after registration of two point-clouds for better
visualization
hFigAlign = figure;
 subplot(121)
 pcshowpair(movingProcessed, fixedProcessed)
 title('Before Registration ICP')
 view(2)

 subplot(122)
 pcshowpair(movingReg, fixedProcessed)
 title('After Registration ICP')
 view(2)

 helperMakeFigurePublishFriendly(hFigAlign);

%% Mapping and localization of lidar scans with IMU and KT Algorithm
inputs.

skipFrames=399; %go to last frame in data set
downsamplePercent=0.30; %parameter will require adjustment based on
application. 30% was the maximum downsample percent found to work for
this application

% create map builder object. MATLAB function
mapBuilder=helperLidarMapBuildericp('DownsamplePercent',downsamplePerce
nt,'RegistrationGridStep',1);

%ICP optimal parameters for this application are
%downsamplePercent=0.30; RegistrationGridStep=1; These values will vary
%based on application used

%set random number seed
rng(0);

closeDisplay=false;
numFrames=height(lidarPointClouds);

for n= 1: skipFrames: numFrames %-skipFrames

    % Get the nth point-cloud
    ptCloud=lidarPointClouds.PointCloud(n);

    % use transformation from ZUPT and KT algorithms as initial
estimate for second point-cloud frame.
```

```
    initTform=ZUPT_TWODIMENSIONAL(filename,startIMU,stopIMU); % Pass
%tform computed from ZUPT algorithm into ICP registration for better
%initial transform estimation

    % Update transform using the ICP registration
    tform = updateMapicp(mapBuilder,ptCloud,initTform);

    % Update map display
    updateDisplay(mapBuilder, closeDisplay);
end

%% Supporting Functions from MATLAB in [10]
% *|helperProcessPointCloud|* processes a point-cloud by removing
points belonging to the ground plane or ego vehicle.

function ptCloudProcessed = helperProcessPointCloud(ptCloud)

% Check if the point-cloud is organized
isOrganized = ~ismatrix(ptCloud.Location);

% If the point-cloud is organized, use range-based flood fill algorithm
% (segmentGroundFromLidarData). Otherwise, use plane fitting.
groundSegmentationMethods = ["planefit", "rangefloodfill"];
method = groundSegmentationMethods(isOrganized+1);

if method == "planefit"
    % Segment ground as the dominant plane, with reference normal
vector
    % pointing in positive z-direction, using pcfitplane. For organized
    % point-clouds, consider using segmentGroundFromLidarData instead.
    maxDistance    = 5;       % meters
    maxAngDistance = 4;         % degrees
    refVector      = [0, 0, 1]; % z-direction

    [~,groundIndices] = pcfitplane(ptCloud, maxDistance, refVector,
maxAngDistance);
elseif method == "rangefloodfill"
    % Segment ground using range-based flood fill.
    groundIndices = segmentGroundFromLidarData(ptCloud);
else
    error("Expected method to be 'planefit' or 'rangefloodfill'")
end

% Segment ego vehicle as points within a given radius of sensor
sensorLocation = [0, 0, 1];
radius         = 1.5;

egoIndices  = findNeighborsInRadius(ptCloud, sensorLocation, radius);

% Remove points belonging to ground or ego vehicle
ptsToKeep = true(ptCloud.Count, 1);
ptsToKeep(groundIndices) = false;
ptsToKeep(egoIndices)    = false;
```

```
% If the point-cloud is organized, retain organized structure
if isOrganized
    ptCloudProcessed = select(ptCloud, find(ptsToKeep), 'OutputSize',
'full');
else
    ptCloudProcessed = select(ptCloud, find(ptsToKeep));
end
end

%%%
% *|helperMakeFigurePublishFriendly|* adjusts figures so that
screenshot
% captured by publish is correct.
function helperMakeFigurePublishFriendly(hFig)

if ~isempty(hFig) && isvalid(hFig)
    hFig.HandleVisibility = 'callback';
end

end
```

B. ZUPT_TWODIMENSIONAL.M

```
%This function takes input linear acceleration from a .mat file and
%processes into two dimensional data for use in the ZUPT algorithm. It
%passes back positional and rotational data for implementation into
%homogenous transform estimate that is used in MAIN_FUSION_SCRIPT.m
%% Code adapted from work done in [2] and [4]

function [tformZUPT]=ZUPT_TWODIMENSIONAL(filename,start,stop)
% start;stop  %adjust for windowing acceleration values. Default length
% is 141

load(filename);

R=[0 -1 0; %Rotation to from quaternion to cart frame
   -1 0 0;
   0 0 -1];
q_b=(rotm2quat(R));

% %Optional: Rotate kinematicTrajectory positional data into
navigational frame for

% %plotting Not required for algorithm.
% O=zeros(length(KTPosition(:,1)),1);
% KTPosition=[O'; KTPosition(:,:)'];
% for m=1:length(KTPosition(1,:))
%     KTPosition_n(:,m)=rotate_v_by_q(KTPosition(:,m),q_b);
% end
%
```

```
%% check for sign flip of quaternion. If so, implement qinv for %
quaternion correction
KTQuat_vector=compact(KTQuat); %  convert KTQuat from quaternion object
to a vector
quatmin=1e-4; %value where quaternion will flip if less than



% For values less than quatmin, quaternion flips since this is close to
%zero. Therefore, take last known good one. Acts as a smoothing filter
for n=1:length(KTQuat);
    if n>=2; %assumes first value is not less than quatmin. May not
always hold true as an assumption
     if KTQuat_vector(n,1)< quatmin
            KTQuat_vector(n,:)=KTQuat_vector(n-1,:);
     end
    end
end

% concetenat linear acclerations array  for rotation with quaternion
O=zeros(length(IMULinAcc(:,1)),1);
IMULinAcc=[O';IMULinAcc(:,:)'];


%Rotate IMU acceleration data with quaternion for acceleration rotation
%into quaternion frame
for m=1:length(IMULinAcc(1,:))
    IMULinAcc_n(:,m)=rotate_v_by_q(IMULinAcc(:,m),KTQuat_vector(m,:));
end


%% Use ZUPT algorithm to integrate rotated data


samplerate=10; %Hz
AccX=IMULinAcc_n(2,:); % *(-1) due to orientation of sensor on cart.
this is x direction
AccY=IMULinAcc_n(3,:); %Acceleration in y direction
AccZ=IMULinAcc_n(4,:); %Acceleration in z direction

Time=[0:1/samplerate:(length(AccX)/samplerate-1/samplerate)];

% Acceleration and time at the hand selected data range the cart is
moving
Ax = AccX(start:stop);
Ay = AccY(start:stop);
Az = AccZ(start:stop);
T = Time(start:stop);

% Uncorrected velocity in sensor frame
AccXs=IMULinAcc(2,:);
Axs=AccXs(start:stop);
Vxs=cumtrapz(Time,AccXs);
```

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

% processing via numerical integration for 2D
% Raw velocity
Vx = cumtrapz(T, Ax);
Vy = cumtrapz(T, Ay);
Vz = cumtrapz(T, Az);


for i = 1:length(Vx)
t(i) = tick;
tick = tick +1;

% % Corrected Velocity
% % Va = Vc - ((Vc(final time)/final time)*t), t = [0, finaltime]
VCx(i,1) = Vx(i) - ((Vx(length(Vx))/length(Vx))*t(i));
VCy(i,1) = Vy(i) - ((Vy(length(Vy))/length(Vy))*t(i));
VCz(i,1) = Vz(i) - ((Vz(length(Vz))/length(Vz))*t(i));

end

% Raw position
X = cumtrapz(T,Vx);
Y = cumtrapz(T,Vy);
Z = cumtrapz(T, Vz);


% Corrected Position
XC =cumtrapz(T, VCx');
YC =cumtrapz(T, VCy');
ZC =cumtrapz(T, VCz');

% Rotate Corrected velocity to LiDAR navigational frame

O=zeros(length(XC),1);
PC_q=[O XC' YC' ZC'];

for m=1:length(PC_q(:,1))
    PC_n(:,m)=rotate_v_by_q(PC_q(m,:),q_b);
end


%Optional: Quaternion data processing for plotting
% for k=1:length(IMUQuat)
%     IMUSensorAngles(k,:)=rad2deg(quat2eul(IMUQuat(k)));
%     KTangles(k,:)=fliplr(rad2deg(Euler(KTQuat_vector(k,:))))');
% end

%% Variables to pass back to main fusion script
```

```
XYZpos=[PC_n(2,(stop-start+1)),PC_n(3,(stop-start+1)),PC_n(4,(stop-
start+1))]; %Translation vector for passing into rigid3d

tformZUPT=rigid3d(quat2rotm(KTQuat(stop)),XYZpos); %rigid3D object for
better initial transform of point-cloud


% %%Optional Plotting

% % ZUPT Plots
% %plot positions direct from KT algorithm
% figure(1)
% plot(KTPosition(2,:),KTPosition(3,:)); xlabel('X (m)');ylabel('Y
(m)');title('MATLAB Positional Data with Turn, LiDAR Frame (1m x 1m)');
% % axis([-3.5 .1 -1.4 1.3]) %Left Turn Plots
% % axis([-0.1 1.7 -.1 1.7])    %Right Turn Plots
%
% figure(2)
% plot(PC_n(2,:),PC_n(3,:));xlabel('X (m)');ylabel('Y (m)');title('ZUPT
Positional Data with Turn, LiDAR Frame (1m x 1m)');
% % axis([-3.5 .1 -1.4 1.3]) %Left Turn Plots
% % axis([-0.1 1.7 -.1 1.7])    %Right Turn Plots
%
% figure(3)
% subplot(211)
% plot(T, VCx)
% title('X velocity Corrected');xlabel('Time (s)'); ylabel('Velocity
(m/sec)');
% subplot(212)
% plot(T, VCy)
% title('Y Velocity Corrected');xlabel('Time (s)'); ylabel('Velocity
(m/sec)');
% sgtitle('ZUPT Corrected Velocity in Quaternion Frame')

% figure(4)
% subplot(211)
% plot(Time, AccXs)
% title('X Acceleration'); xlabel('Time (s)'); ylabel('Acceleration
(m/sec^2)');
% subplot(212)
% plot(T,Axs)
% title('X Acceleration windowed.'); xlabel('Time (s)');
ylabel('Acceleration (m/sec^2)');
% sgtitle('Raw X Acceleration in Sensor Frame')

% figure(5)
% subplot(211)
% plot(T, PC_q(:,2))
% title('X Position corrected');xlabel('Time (s)');ylabel('Distance
(m)');
% subplot(212)
% plot(T, PC_q(:,3))
% title('Y Position Corrected');xlabel('Time (s)');ylabel('Distance
(m)');
```

```
% sgtitle('ZUPT Positions in Quaternion Frame')

% Quaternion Plots
% endtime=120;

% figure(6)
% subplot(311)
% plot([0:.1:endtime],IMUSensorAngles(:,1)); xlabel('Time (s)');
ylabel('Angle (deg)');title('Yaw')
% axis([0 Time(end) -200 200])
% subplot(312);
% plot([0:.1:endtime],IMUSensorAngles(:,2)); xlabel('Time (s)');
ylabel('Angle (deg)'); title('Pitch');
% axis([0 Time(end) -50 50])
% subplot(313)
% plot([0:.1:endtime],IMUSensorAngles(:,3)); xlabel('Time (s)');
ylabel('Angle (deg)'); title('Roll');
% axis([0 Time(end) -160 -50])
% sgtitle('Angles Produced from Sensor Quaternion')

% figure(7)
% subplot(311)
% plot([0:.1:endtime],KTangles(:,1)); xlabel('Time (s)'); ylabel('Angle
(deg)');title('Yaw')
% axis([0 Time(end) -185 20])
% subplot(312);
% plot([0:.1:endtime],KTangles(:,2)); xlabel('Time (s)'); ylabel('Angle
(deg)'); title('Pitch');
% axis([0 Time(end) -50 50])
% subplot(313)
% plot([0:.1:endtime],KTangles(:,3)); xlabel('Time (s)'); ylabel('Angle
(deg)'); title('Roll');
% axis([0 Time(end) -50 50])
% sgtitle('Angles Produced from Kinematic Trajectory Quaternion')

end
```

APPENDIX D. MULTI-STEP FUSION

A. MAIN_FUSION_MULTISTEP.M

```
%%% FUSION MULTISTEP algorithm takes lidar point-clouds and
acceleration inputs
%%% passed through collection scripts and processes and builds a point
%%% cloud map. Requires ZUPT_TWODIMENSIONAL_MULTISTEP.m,
%%%helperLidarMapbuildericp from MATLAB, and ACCELERATIONPLOT.m to
%%%work. Hand selection for windowing accelerationdata is also
required. Code adapted from [10]

clearvars;clc; close all;

%load collected data
data=load('LIDAR_IMUCombined_Data_LabFullRun_2_LiDAR_Stitched_RightTurn
.mat');
%load pointcloud data from LiDAR. Track data (1x1, etc) must match IMU
track data

filename='LIDAR_IMUCombined_Data_LabFullRun_2_IMU_Stitched_RightTurn.ma
t'; %filename to pass raw data from IMU into ZUPT algorithm

% %convert to format usable for MATLAB algorithms
lidarPointClouds=data.PointClouds(1:end);
stop=length(data.PointClouds);

% %Set map player limits for later
xlimits=[-10 10];
ylimits=[-10 10];
zlimits=[-5 5];

% User input prompts

Accelplot(filename) %load accleration data for showing where to manual
select accleration windowning for ZUPT and number of steps. Stitched
from several collection runs of IMU.

prompt1='Input number of Stops from Acceleration Figure:'; %number of
point-clouds collected
prompt2='Input start index for windowing to the last step:';
% acceleration profiles between each point-cloud.
prompt3='Input end index for windowing the last step:';

Stops=input(prompt1); %used for iterative loop for matching scans to
accelration profiles

downsamplePercent=0.31; %parameter will require adjustment based on
application

% create map builder object
```

```
mapBuilder=helperLidarMapBuildericp('DownsamplePercent',downsamplePerce
nt,'RegistrationGridStep',1);
    %ICP optimal parameters for this application are
    %downsamplePercent=0.30; RegistrationGridStep=1; These values will
vary based on application
closeDisplay=false;

%% Loop for step iterations based on Figure 1
tformglobal=rigid3d;

for p=1:Stops
clc
fprintf('Stop %u:\n',p-1)
NextFrame=p;
% index for next step ahead
if p>1
 %these require manual adjustment for optimal ZUPT windowing of
acceleration. Passed to ZUPT function
startIMU=input(prompt2);stopIMU=input(prompt3);
else startIMU=p; stopIMU=2; %placeholder values to get past first loop
iteration. InitTform will essentially be identity rotation.
end

%% Mapping and localization of LiDAR scans with IMU and KT Algorithm
inputs.

%set random number see
rng(0);

%get the next point-cloud from NEXTFRAME input. Point-clouds have also
been stitched into one file and pulled into script as with IMU data.

    ptCloud1=lidarPointClouds(NextFrame);
    ptCloud=helperProcessPointCloud(ptCloud1);

% use transformation from previous iteration as initial estimate for
    % second point-cloud frame

initTform=ZUPT_TWODIMENSION_MULTISTEP(filename,startIMU,stopIMU,tformgl
obal);
    % Pass tform computed from ZUPT algorithm into ICP registration for
    %bettter initial transform estimation

    % Update transform using the ICP registration
    tform = updateMapicp(mapBuilder,ptCloud,initTform);

    %update global quaternion with incremented transform from ICP.
    %Passes to ZUPT algorithm for correct acceleration correction.

quatglobal=q_mult2(rotm2quat(tformglobal.Rotation),rotm2quat(tform.Rota
tion)'); %since the imu initializes after each lidar scan at a stop,
the quaternion will reset. This accounts for the reinitilization.
    rad2deg(Euler(quatglobal))
    tformglobal=rigid3d(quat2rotm(quatglobal'),tform.Translation);
```

```
    %homogeneous transform required since IMU is reinitialized between
each LIDAR scan

    % Update map display
    updateDisplay(mapBuilder, closeDisplay);
end

%% Supporting Functions from MATLAB in [10]
%%%
% *|helperProcessPointCloud|* processes a point-cloud by removing
points
% belonging to the ground plane or ego vehicle.
function ptCloudProcessed = helperProcessPointCloud(ptCloud)

% Check if the point-cloud is organized
isOrganized = ~ismatrix(ptCloud.Location);

% If the point-cloud is organized, use range-based flood fill algorithm
% (segmentGroundFromLidarData). Otherwise, use plane fitting.
groundSegmentationMethods = ["planefit", "rangefloodfill"];
method = groundSegmentationMethods(isOrganized+1);

if method == "planefit"
    % Segment ground as the dominant plane, with reference normal
vector
    % pointing in positive z-direction, using pcfitplane. For organized
    % point-clouds, consider using segmentGroundFromLidarData instead.
    maxDistance    = 5;       % meters
    maxAngDistance = 4;         % degrees
    refVector      = [0, 0, 1]; % z-direction

    [~,groundIndices] = pcfitplane(ptCloud, maxDistance, refVector,
maxAngDistance);
elseif method == "rangefloodfill"
    % Segment ground using range-based flood fill.
    groundIndices = segmentGroundFromLidarData(ptCloud);
else
    error("Expected method to be 'planefit' or 'rangefloodfill'")
end

% Segment ego vehicle as points within a given radius of sensor
sensorLocation = [0, 0, 1];
radius         = 1.5;

egoIndices  = findNeighborsInRadius(ptCloud, sensorLocation, radius);

% Remove points belonging to ground or ego vehicle
ptsToKeep = true(ptCloud.Count, 1);
ptsToKeep(groundIndices) = false;
ptsToKeep(egoIndices)    = false;

% If the point-cloud is organized, retain organized structure
if isOrganized
```

```
    ptCloudProcessed = select(ptCloud, find(ptsToKeep), 'OutputSize',
'full');
else
    ptCloudProcessed = select(ptCloud, find(ptsToKeep));
end
end

%%%
% *|helperAddLegend|* adds a legend to the axes.
function helperAddLegend(hAx, labels)

% Add a legend to the axes
hLegend = legend(hAx, labels{:});

% Set text color and font weight
hLegend.TextColor  = [1 1 1];
hLegend.FontWeight = 'bold';
end

%%%
% *|helperMakeFigurePublishFriendly|* adjusts figures so that
screenshot
% captured by publish is correct.
function helperMakeFigurePublishFriendly(hFig)

if ~isempty(hFig) && isvalid(hFig)
    hFig.HandleVisibility = 'callback';
end

end
```

B. ZUPT_TWODIMENSION_MULTISTEP.M

```
%This function takes inputed linear acceleration from a .mat file and
%processes into two dimensional data for use in the ZUPT algorithm. It
%passes back positional and rotational data for implementation into
%homogenous transform estimate that is used in MAIN_FUSION_MULTISTEP.m
%script. It also accounts for IMU reinitializing the quaternion after
%each lidar stop, otherwise the quaternion would give a zero heading
%and not sum any past rotations that occurred in previous steps.
%% Code adapted from work done in [2] and [4]

function
[tformZUPT]=ZUPT_TWODIMENSION_MULTISTEP(filename,start,stop,tformICP)

% start;stop  %adjust for windowing acceleration values. Default length
is 141

%% Processing script for M Files from IMU_COLLECTON.m script
load(filename);

%% IMU accelerations are unprocessed and in sensor frame
```

```
quatmin=1e-4;
R=[0 -1 0; %Rotation to cart frame
   -1 0 0;
   0 0 -1];

q_b=(rotm2quat(R));

%% checking for errors in quaternion.
KTQuat_Scalar=compact(KTQuat); %  convert KTQuat from quaternion object
to a vector

% For values less than quatmin, quaternion will not adjust properly
since this is close to zero. Therefore, take last known good one. Acts
as a smoothing filter

for n=1:length(KTQuat);
    if n>=2;
%assumes first value is not flipped. May not always hold as an
%assumption

    if KTQuat_Scalar(n,1)< quatmin
            KTQuat_Scalar(n,:)=KTQuat_Scalar(n-1,:);
     end
    end
end



% Concatenate linear accleration array for rotation with quaternion
O=zeros(length(IMULinAcc(:,1)),1);
IMULinAcc=[O';IMULinAcc(:,:)'];

ICPQuat_global=rotm2quat(tformICP.Rotation);


%Rotate IMU acceleration data with quaternion for acceleration
corrections
%to GLOBAL FRAME
for m=1:length(IMULinAcc(1,:))
Quat_Global(m,:)=rotate_v_by_q(KTQuat_Scalar(m,:),ICPQuat_global');
IMULinAcc_n(:,m)=rotate_v_by_q(IMULinAcc(:,m),KTQuat_Scalar(m,:));
end


%% use ZUPT algorithm to integrate rotated data
samplerate=10; %hz
% size=121;
AccX=IMULinAcc_n(2,:); % *(-1) due to orientation of sensor on cart.
%this is x direction
AccY=IMULinAcc_n(3,:); %Acceleration in y direction
AccZ=IMULinAcc_n(4,:); %Acceleration in z direction

Time=[0:1/samplerate:(length(AccX)/samplerate-1/samplerate)];
```

```
% Acceleration and time at the hand selected data range the robot is
moving
Ax = AccX(start:stop); % rosbag message format already in m/sec^2
Ay = AccY(start:stop);
Az = AccZ(start:stop);
T = Time(start:stop);

% Uncorrected velocity in sensor frame
AccXs=IMULinAcc(2,:);
Axs=AccXs(start:stop);
Vxs=cumtrapz(Time,AccXs); %uncorrected velocity in sensor frame

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

% processing via numerical integration for 3D
% Raw velocity
Vx = cumtrapz(T, Ax);
Vy = cumtrapz(T, Ay);
Vz = cumtrapz(T, Az);


for i = 1:length(Vx)
t(i) = tick;
tick = tick +1;

% % Corrected Velocity
% % Va = Vc - ((Vc(final time)/final time)*t), t = [0, finaltime]
VCx(i,1) = Vx(i) - ((Vx(length(Vx))/length(Vx))*t(i));
VCy(i,1) = Vy(i) - ((Vy(length(Vy))/length(Vy))*t(i));
VCz(i,1) = Vz(i) - ((Vz(length(Vz))/length(Vz))*t(i));

end

% Raw position
X = cumtrapz(T,Vx);
Y = cumtrapz(T,Vy);
Z = cumtrapz(T, Vz);


% Corrected Position
XC =cumtrapz(T, VCx');
YC =cumtrapz(T, VCy');
ZC =cumtrapz(T, VCz');

% Rotate Corrected velocity to LiDAR navigational frame
O=zeros(length(XC),1);
PC_q=[O XC' YC' ZC'];

for m=1:length(PC_q(:,1))
    PC_n(:,m)=rotate_v_by_q(PC_q(m,:),q_b);
end
```

```
%% Variables to pass out PC_n=XYZ
XYZpos=[PC_n(2,(stop-start+1)),PC_n(3,(stop-start+1)),PC_n(4,(stop-
start+1))]; %Translation vector for passing into rigid3d

tformZUPT=rigid3d(quat2rotm(KTQuat(stop)),XYZpos);

%rigid3D object for better initial transform of point-cloud
%note that KTQuat portion of tfromZUPT is a segmented rotational
%matrix. i.e. IMU reinitialized between scans. This prevents ICP
%algorithm from over rotating. The translational data needs the global
%quaternion taken from ICP algorithm for correct absolute rotation of
the translational data.

end
```

C. ACCELPLOT.M

```
%This function is used to help in visually identifying where to %index
%acceleration segments to allow for proper windowing in multistep
%fusion function.

function a=Accelplot(filename)

load(filename);
plot(IMULinAcc(:,1));xlabel('Index');ylabel('X Acceleration(Unwindowed)
m/s^2');title('Use for Step Input,Manually Windowed indexs');

end
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. ADDITIONAL FUNCTION SCRIPTS

A. Q_MULT2.M

```
%Code provided from [4]. For quaternion multiplication.
function qout=q_mult2(p,q)

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;
```

B. ROTATE_V_BY_Q.M

```
%Code provided from [4]. Rotates vector using quaternions.

function u=rotate_v_by_q(v,q)

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

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

C. EULER.M

```
%Code provided from [4]. Computes euler angles from a quaternion.

function EulerAngles=Euler(u)

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];

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

LIST OF REFERENCES

[1] Commandant of Marine Corps, "38th Commandant's planning guidance," Washington, DC, USA, 2020 [Online]. Available: https://www.marines.mil/News/Publications/MCPEL/Electronic-Library-Display/Article/1907265/38th-commandants-planning-guidance-cpg/

[2] S. S. Druen, "Robotic navigation in GPS-denied environments using the strapdown navigation algorithm with zero-velocity updates," M.S. thesis, Dept. of Elec. and Comput. Eng., NPS, Monterey, CA, USA, 2020 [Online]. Available: https://calhoun.nps.edu/handle/10945/65509

[3] M. Vlaminck, H. Luong, and W. Philips, "Multi-resolution ICP for the efficient registration of point clouds based on octrees," *2017 APR Int. Conf. MV*, May 2017, pp. 334–337 [Online]. doi: 10.23919/MVA.2017.7986869

[4] J. Calusdian, "A personal navigation system based on inertial and magnetic field measurements," PhD dissertation, Depart. of Elect. Eng., Naval Postgraduate School, 2010 [Online]. Available: https://calhoun.nps.edu/handle/10945/10557

[5] J. Tang *et al.*, "LiDAR scan matching aided inertial navigation system in GNSS-denied environments," *Sensors*, vol. 15, no. 7, pp. 16710–16728, Jul. 2015 [Online]. doi: 10.3390/s150716710

[6] X. Zuo, P. Geneva, W. Lee, Y. Liu, and G. Huang, "LIC-fusion: LiDAR-inertial-camera odometry," *2019 IEEE/RSJ Int. Conf. on Intelligent Robots and Sys.*, Nov. 2019, pp. 5848–5854 [Online]. doi: 10.1109/IROS40897.2019.8967746

[7] H. Ye, Y. Chen, and M. Liu, "Tightly coupled 3D lidar inertial odometry and mapping," *2019 Int. Conf. on Robotics and Automation (ICRA)*, May 2019, pp. 3144–3150 [Online]. doi:10.1109/ICRA.2019.8793511

[8] C. Debeunne and D. Vivet, "A review of visual-lidar fusion based simultaneous localization and mapping," *Sensors*, vol. 20, no. 7, Apr. 2020 [Online]. doi: 10.3390/s20072068

[9] J. S. Payne, "Autonomous interior mapping robot utilizing LIDAR localization and mapping," M.S. thesis, Dept. of Elect. and Comput. Eng., NPS, Monterey, CA, USA, 2020 [Online]. Available: https://calhoun.nps.edu/handle/10945/66121

[10] MathWorks, "Build a map from lidar data." [Online]. Available: https://www.mathworks.com/help/driving/ug/build-a-map-from-lidar-data.html

[11] T. Taketomi, H. Uchiyama, and S. Ikeda, “Visual SLAM algorithms: A survey from 2010 to 2016,” *IPSJ Trans. Comput. Vis. Appl.*, vol. 9, no. 1, p. 16, Jun. 2017 [Online]. doi: 10.1186/s41074-017-0027-2

[12] G. Huang, “Visual-inertial navigation: A concise review,” *2019 Int. Conf. on Robotics and Automation (ICRA)*, May 2019, pp. 9572–9582 [Online]. doi: 10.1109/ICRA.2019.8793604

[13] MathWorks, “Register two point clouds using ICP algorithm.” [Online]. Available: https://www.mathworks.com/help/vision/ref/pcregistericp.html

[14] J. Kuipers, *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality*. Princeton University Press, 1999.

[15] MathWorks, “rigid3d.” [Online]. Available: https://www.mathworks.com/help/images/ref/rigid3d.html

[16] Yun, Xiaoping, “Description of position and orientation,” Class Notes EC4310: Fundamentals of Robotics, Dept. of Elect. and Comput. Eng., NPS, Monterey, CA, USA, Fall 2020.

[17] Parker Lord, “3DM-GX5-AHRS Microstrain Sensing Product Datasheet.” 2020 [Online]. Available: https://www.microstrain.com/inertial-sensors/3dm-gx5-25

[18] Velodyne, “VLP-16 User Manual.” [Online]. Available: https://velodynesupport.zendesk.com/hc/en-us/categories/115000212233-Product-Information

[19] MathWorks, “What Is MATLAB?” [Online]. Available: https://www.mathworks.com/discovery/what-is-matlab.html

[20] Lentin Joseph and Jonathan Cacace, *Mastering ROS for Robotics Programming*, 2nd ed. Birmingham, UK: Packt, 2018.

[21] 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 Tran Inst Meas.*, vol. 57, no. 3, pp. 638–650, Mar. 2008 [Online]. doi: 10.1109/TIM.2007.911646

[22] MathWorks, “kinematicTrajectory.” [Online]. Available: https://www.mathworks.com/help/nav/ref/kinematictrajectory-system-object.html

[23] MathWorks, “Object for storing 3-D point cloud.” [Online]. Available: https://www.mathworks.com/help/vision/ref/pointcloud.html

[24] MathWorks, “Point cloud SLAM overview.” [Online]. Available: https://www.mathworks.com/help/vision/ug/point-cloud-registration-workflow.html#responsive_offcanvas

[25] P. J. Besl and N. D. McKay, “A method for registration of 3-D shapes,” *IEEE Tran Pattern Anal Mach Intell*, vol. 14, no. 2, pp. 239–256, Feb. 1992 [Online]. doi: 10.1109/34.121791

[26] Baerentzen, Jakob Andreas, A. Anton Jens Gravesen Francois, and A. Henrik, *Guide to Computational Geometry Processing*. London, UK: Springer, 2012.

[27] P. Biber and W. Strasser, “The normal distributions transform: A new approach to laser scan matching,” *Proceedings 2003 IEEE*, Oct. 2003, vol. 3, pp. 2743–2748 [Online]. doi: 10.1109/IROS.2003.1249285.

[28] M. Magnusson, “The three-dimensional normal-distributions transform: An efficient representation for registration, surface analysis, and loop detection,” 2009 [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:oru:diva-8458

[29] A. Myronenko and X. Song, “Point set registration: Coherent point drift,” *IEEE Trans Pattern Anal Mach Intell*, vol. 32, no. 12, pp. 2262–2275, Dec. 2010 [Online]. doi: 10.1109/TPAMI.2010.46

[30] M. Dimitrievski, D. V. Hamme, P. Veelaert, and W. Philips, “Robust matching of occupancy maps for odometry in autonomous vehicles,” *Conf on Comp. Vision, Imaging and Computer Graphics Theory and App.*, Jul. 2021, pp. 626–633 [Online]. doi: 10.5220/0005719006260633

[31] Y. Chen and G. Medioni, “Object modelling by registration of multiple range images,” *Image Vis. Comput.*, vol. 10, no. 3, pp. 145–155, Apr. 1992 [Online]. doi: 10.1016/0262-8856(92)90066-C

[32] K. McGuire, G. de Croon, C. De Wagter, K. Tuyls, and H. Kappen, “Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone,” *IEEE Robotics and Automation Letters*, Apr. 2017, vol. 2 [Online]. doi: 10.1109/LRA.2017.2658940

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California