INDOOR LOCALIZATION USING INERTIAL NAVIGATION SYSTEMS

by

Touqeer Ahmad

A THESIS

Submitted to the Faculty of the Stevens Institute of Technology

In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE – ELECTRICAL ENGINEERING

_____

Touqeer Ahmad, Candidate

ADVISORY COMMITTEE

_____

Yu-Dong Yao, Advisor Date

_____

Kevin W. Lu, Reader Date

STEVENS INSTITUTE OF TECHNOLOGY

Castle Point on Hudson

Hoboken, NJ 07030

2017

INDOOR LOCALIZATION USING INERTIAL NAVIGATION SYSTEMS

Abstract

Localizing objects in an environment is very important as a lot of tasks depend upon the presence of objects at certain points. Outdoor localization can be easily done using global navigation satellite system (GNSS). This method does not ensure certain accuracy for an indoor environment. A host of indoor localization techniques are available to improve accuracy. Most of them involve installing certain equipment (transmitter and receiver nodes). This thesis investigates an approach based on an inertial measurement unit (IMU) that doesn't involve any equipment installation and can track objects in an indoor environment with precision. The IMU used includes 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer. The accelerometer and gyroscope measure multiple forces that act on the sensor making the measured data noisy. Different filters including 6^{th} order lowpass Butterworth filter, denoised filter, and median filter are used to filter out the noise without effecting the shape of the original signal. The magnetometer measures the magnetic field in all directions providing the absolute magnetic north. All sensors are calibrated to eliminate any bias including acceleration due to gravity measurement by the accelerometer at rest, zero angular velocity by the gyroscope, and heading correctness for the magnetometer.

A complimentary filter is used to estimate the orientation of the object by fusing the data from the accelerometer, gyroscope, and magnetometer. Static acceleration components (gravity and static noises) are removed from its orientation. The Kalman filter is used to predict the position of the object using the dynamic acceleration and dead

reckoning technique. GPS data are also incorporated to provide an initial position and reduce the chances of drift caused by dead reckoning.

The algorithm was tested in real-time, the raw data were logged for different motions and implemented in MATLAB to predict the position of the object. At the end, algorithm was implemented on data collected from three different IMU devices including standalone MPU9255 sensor, iPhone 7, and Jackal Robot. The proposed algorithm shows similar position accuracy compare to other indoor tracking techniques that require equipment installation.

Touqeer Ahmad

Dr. Yu-Dong Yao

December 7, 2017

Electrical and Computer Engineering

Master of Science - Electrical Engineering

Dedication

I dedicate this work to my friends, family, and teachers.

Acknowledgements

All praises are for the God, who has given us the strength and courage to explore the knowledge of the world.

I would take this opportunity to thank Dr. Kevin Lu and Prof. Yu-Dong Yao, my thesis advisors, for their valuable support and encouragement in choosing the thesis topic and proceeding further with the development. Without their guidance and expertise, completion of the thesis would not have been possible. Their words of wisdom will always be remembered, and I am convinced that the knowledge that they have imparted would go a long way in helping me all through my professional career.

Very special thanks go to Mr. Nagarajan and Mr. Piyush for their help throughout the thesis. Thanks to the family and friends for their continuous support and encouragement during the thesis.

Table of Contents

List of Figures ix

List of Tables xi

List of Abbreviations xii

1. Introduction 1

1.1. Literature Review 1

1.2. Problem Formulation 3

1.3. Proposed Solution 4

1.4. System Setup 4

2. Methodology 9

2.1. Data Filtering 10

2.2. Calibration 12

2.3. Gravity Vector 12

2.4. Position Calculation 14

3. Results 16

3.1. Motion in a Straight-Line 16

3.1.1. MPU-9255 16

3.1.2. Jackal Robot IMU Sensor 19

3.1.3. Smartphone (iPhone 7) 22

3.2. Turning Motion 25

3.2.1. MPU-9255 26

3.2.2. Jackal Robot IMU Sensor 28

3.2.3. Smartphone (iPhone 7) 30

4. Cost-Performance Tradeoff 35

5. Recommendations 36

6. Summary and Conclusions 37

Appendices 38

References 53

List of Figures

Figure 1. A Raspberry Pi 3B with an MPU9255 sensor 6

Figure 2. An MPU9255 sensor connected to a Raspberry Pi 3B 6

Figure 3. A Jackal Robot 7

Figure 4. An MPU-9255 sensor connected with a Raspberry Pi 3B and iPhone 7 mounted on a Jackal Robot 8

Figure 5. Flowchart of data processing and position calculation 9

Figure 6. IMU (accelerometer and gyroscope) raw and filtered data 11

Figure 7. Direction cosines, gravity and dynamic acceleration data 13

Figure 8. Calculated and filtered dynamic acceleration and velocity and displacement vector 15

Figure 9. Raw and filtered data from the MPU9255 16

Figure 10. DC's, gravity vector, and dynamic motion data for the MPU9255 17

Figure 11. Dynamic acceleration on the left, and velocity and displacement profiles on the right 18

Figure 12. Movement on the Cartesian plane 19

Figure 13. Raw and filtered data from the Jackal Robot 20

Figure 14. DC's, gravity vector, and dynamic motion vector for the Jackal Robot 20

Figure 15. Dynamic acceleration, velocity and displacement profiles from the Jackal Robot 21

Figure 16. Movement from the Jackal Robot data on the Cartesian plane 22

Figure 17. Raw and filtered data from the iPhone 7 23

Figure 18. DC's, gravity vector and dynamic motion data for the iPhone 7 23

Figure 19. Dynamic acceleration (left side), velocity, and displacement profiles (right side) for the iPhone 7 24

Figure 20. Movement from the iPhone 7 data on the Cartesian plane 25

Figure 21. Raw and filtered MPU9255 Data 26

Figure 22. DC's, gravity vector, and dynamic motion data for the MPU9255 27

Figure 23. Dynamic acceleration, velocity and displacement Profile for the MPU9255 . 28

Figure 24. Raw and filtered data from the Jackal Robot 29

Figure 25. DC's, gravity vector, and dynamic motion vector for the Jackal Robot 29

Figure 26. Dynamic acceleration, and velocity and displacement profile 30

Figure 27. Raw and filtered data from the iPhone 7 31

Figure 28. DC's, gravity vector and dynamic motion data for the iPhone 7 32

Figure 29. Dynamic acceleration (left side), velocity, and displacement profiles (right side) for the iPhone 7 32

List of Tables

Table 1. Results for the MPU9255 33

Table 2. Results for the Jackal Robot 34

Table 3. Results for the iPhone 7 34

Table 4. Gyro grade based on bias stability 35

List of Abbreviations

GPS	Global Positioning System
IMU	Inertial Measurement Unit
MEMS	Microelectromechanical Systems
GNSS	Global Navigation Satellite System
DC’s	Direction Cosines
RF	Radio Frequencies
WLAN	Wireless Local Area Networks
UWB	Ultra-Wide Band
SiP	System in Package
DMP	Digital Motion Processor
RAM	Random Access Memory
ROS	Robot Operating System

1. Introduction

Localizing objects in an environment is very important as a lot of tasks depend upon the presence of objects at certain points. Outdoor localization can be easily done using global navigation satellite system (GNSS). These methods do not ensure certain accuracy for indoor. There are several indoor localization techniques available to improve accuracy.

1.1. Literature Review

Using a variety of technologies, it is possible to track different objects in an indoor environment. Some of them are based on Wireless Local Area Networks (WLAN), Ultra-Wide Band (UWB), and Radio Frequencies (RF), and inertial sensors. By tagging objects with appropriate receivers/tags and deploying a few nodes (access points, receivers, transmitter, etc.) at fixed indoor locations, one can determine and continuously track the location of the objects.

P. Bahl and V.N. Padmanabhan presented an RF-based system for locating and tracking users in an indoor environment. It operates by measuring and processing signal strength information at multiple base stations positioned to provide overlapping coverage in the area of interest. It combines empirical measurements with signal propagation modeling to determine user location. Their experimental results demonstrate the capability of the system to estimate user location with median error of two to three meters [6].

Aleksandr Mikov successfully implemented the inertial tracking by using microelectromechanical systems (MEMS) sensors [1]. He used dead reckoning technique based on step detection and its length and orientation estimation. The trials in the paper

achieved maximum return position error of 7% of total distance and 10-degree error in absolute heading. Unlike similar systems, Mikov's approach doesn't restrict the sensor to be at specific position or any prior knowledge of the environment. Three different modes were introduced for power efficient operation of the developed system.

Before using sensors, it is important to calibrate them for their effective use in different applications. Gietzelt and others introduced a self-calibration technique for accelerometers with an error of 0.009g, better than other calibration techniques available [2]. No specific movement of the sensor is required since the technique calibrates the sensor automatically while the sensor provides readings based on probabilistic and attentive observation of the pattern of accelerometer data. The problem with this technique is the restriction of the placement of accelerometer on torso because a wearing position to the barycenter of the body naturally causes little noise. They also developed an activity detection algorithm to successfully recognize the activity of the person and improve the calibration technique by using the mean values of that activity.

Heading determination is one of the important aspects of tracking an object. The magnetometer is a low-cost sensor that can measure the absolute north heading (magnetic north) by sensing the magnetic field of the earth. Usually, presence of different ferromagnetic materials in the environment can affect the magnetometer measurements of the magnetic field of the earth. It is necessary to calibrate a magnetometer to minimize such effects. Wahdan and others developed 3-D magnetometer calibration using 2-D calibration equations with yaw, pitch, and roll sectors [3]. Calibration for the magnetometer using 2-D technique can be performed for all these sectors, providing efficient 3-D magnetometer

calibration. Unlike similar calibration techniques such as compass swinging, this method doesn't require users to rotate the compass in predefined directions.

Herrera and others modeled the dynamics of a walking user and carried out the data fusion of inertial sensors and a UWB-based system with the Kalman filter. They used dead reckoning technique to track the walking person, and evaluated the Kalman filter with or without associated evaluation criteria. They tested their system in a 4x7 m^2 room and discovered that the standard deviation in X direction decreased from four cm (without associated evaluation criteria) to two cm and in Y direction from six cm to two cm by incorporating the evaluation criteria [4].

Tracking based on dead reckoning requires start position of the object. GPS system can be used to provide the start position of the object when it enters the building or underground tunnels where GPS signal are not available or very low for precise tracking. Belhajem and others presented a fleet management system for the smart city where they successfully able to predict the fleet position using a neural network and EKF (Extended Kalman Filter) on the inertial navigation system fused with GPS. They predicted the position of the fleet in case of GPS failure 95% better by using a machine learning technique of the neural network with a genetic algorithm on their GPS enhanced dead reckoning motion sensors [5].

1.2. Problem Formulation

There are several techniques available to track objects in an indoor environment. Most of them involve installing certain equipment (transmitter and receiver nodes). The purpose of this thesis is to investigate an approach that doesn't involve any equipment

installation and can track objects in an indoor environment with precision. Instead of installing equipment in the environment, one can attach sensors with the object. The sensor should be able to accurately measure the movement and orientation of the object. Given the start position and sensor recordings about movement and orientation, it is possible to accurately predict the new position of the object.

1.3. Proposed Solution

The proposed solution uses inertial sensors and the dead reckoning technique to track objects in an indoor environment. Inertial sensors include 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer. The accelerometer sensor measures how fast the velocity of the sensor is being changed over time. It can be used to check the position change between two points by integrating velocity over time. The 3-axis accelerometer measures the change in each axis. The gyroscope sensor measures rotational velocity in radians per seconds. It can be used to check the change in sensor orientation. The 3-axis gyroscope measures the change in each axis. The magnetometer sensor measures the magnetic field of the earth and can help identify the absolute direction of North. By combining the data from these sensors, one can successfully formulate an algorithm to calculate the new position of the object.

1.4. System Setup

InvenSense MPU-9255 is used as the inertial sensor. It is a System in Package (SiP) that fuses 9-axis (accelerometer, gyroscope, and magnetometer) sensor with Digital Motion Processor (DMP). Digital Motion Processor present in sensor can successfully identify the activity of the user if used in the wearable devices for humans in about three seconds [7].

For this thesis, calculations from the DMP are not used as they give three seconds delay prediction that is not acceptable while tracking an object in real time.

Raspberry Pi 3 model B is used to integrate the MPU-9255 sensor using I2C (Inter-Integrated Circuit [9]) protocol at hex address of “0x0C” to get the 9-axis data (3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer). Raspberry Pi is a single board computer with the 1.2 GHz ARM Cortex-A53 CPU and 1GB RAM [8]. The data with almost 40 times per second for all the 9-axis of sensor is logged inside the Raspberry Pi 3 on pre-defined path and then exported to an external system (windows) using SSH (Secure Shell [10]) to run the developed tracking algorithm on MATLAB [11]. Circuit diagram of MPU-9255 attached with Raspberry Pi 3B is shown in Figure 1 using Fritzing tool [12]. Figure 2 shows that a MPU-9255 sensor is connected to a Raspberry Pi 3B.

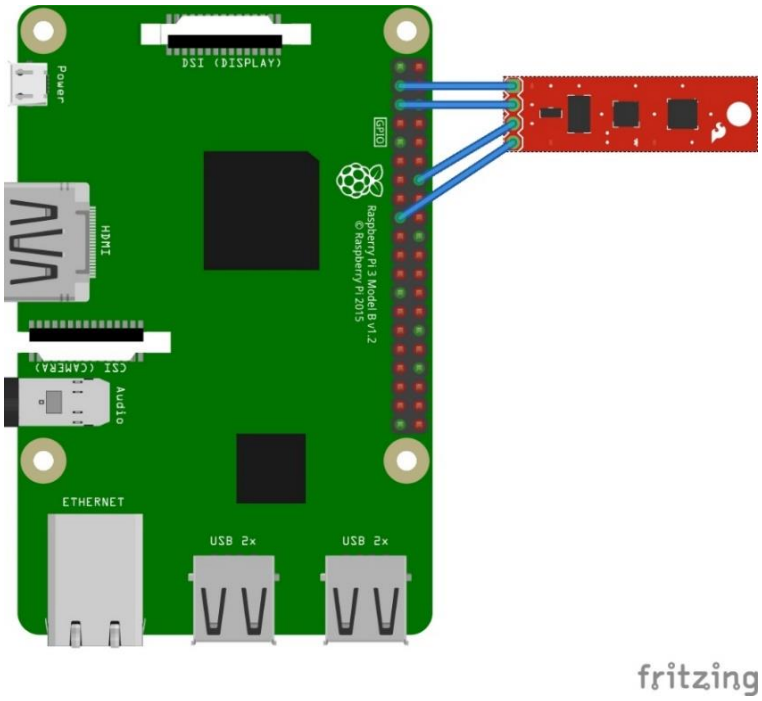

Figure 1. A Raspberry Pi 3B with an MPU9255 sensor

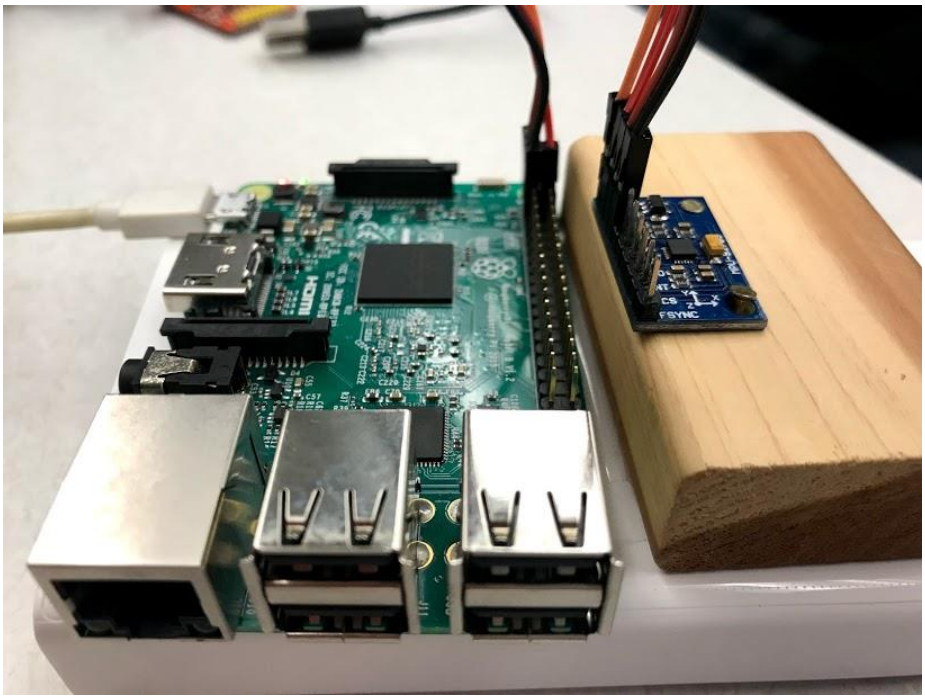

Figure 2. An MPU9255 sensor connected to a Raspberry Pi 3B

The Raspberry Pi 3B connected with the MPU9255 sensor is mounted on top of a Jackal Robot [13] to move the system around in the environment. The Jackal Robot has its own inertial measurement unit (IMU), LORD MicroStrain 3DM-GX3-25. The data from this sensor is also logged inside the robot while in motion to compare the result obtained from both sensors. Figure 3 shows the front side of the Jackal Robot with the IMU sensor position labeled [14].

Figure 3. A Jackal Robot

To compare the proposed solution working on different IMU sensors, data from iPhone 7 IMU sensor (6-axis InvenSense MPU-6500 accelerometer and gyroscope, and 3-axis Bosch BMA280 accelerometer) is logged on the iPhone 7 iOS app “MATLAB mobile” [15] and then exported to MATLAB running on windows machine using

MATLAB mobile connector. Figure 4 shows an MPU-9255 sensor connected with the Raspberry Pi 3B and iPhone 7 mounted on top of the Jackal Robot.

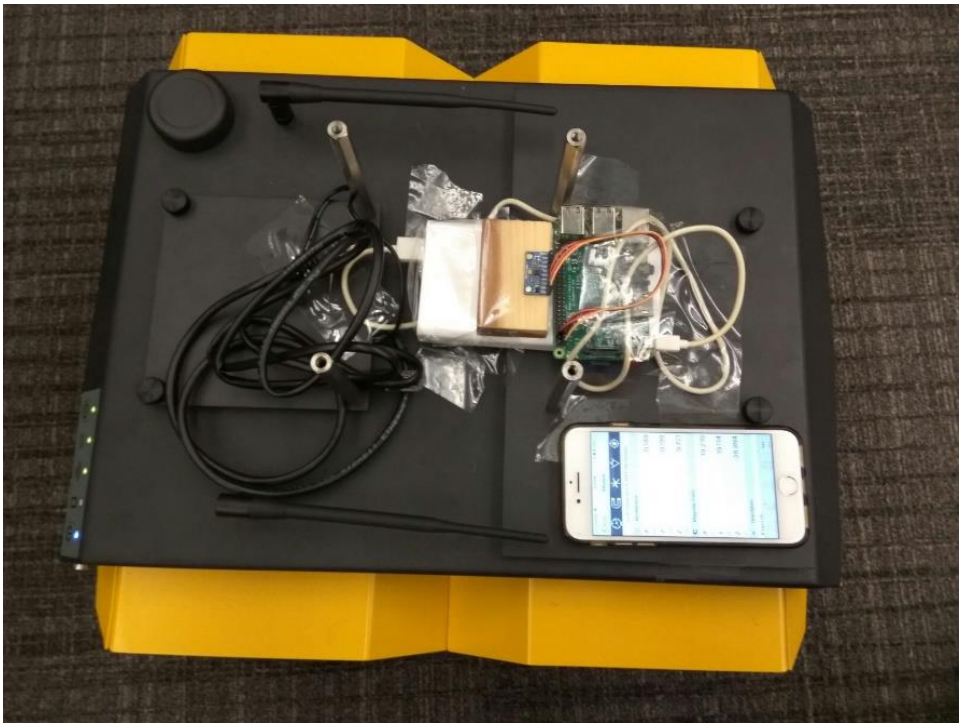

Figure 4. An MPU-9255 sensor connected with a Raspberry Pi 3B and iPhone 7 mounted on a Jackal Robot

2. Methodology

The algorithm developed for this thesis has three parts including data collection, filtering, and calculation as shown in Figure 5. The data from a gyroscope and accelerometer are angular velocity (Gyro) and acceleration (Acc). These data are processed through a Butterworth low-pass (LP) filter, denoised filter, and median filter, and denoted by Gyro_f and Acc_f. A complimentary filter fuses the filtered data of gyroscope and accelerometer to derive direction cosines (DCs) that are combined with the filtered acceleration data to calculate gravity vectors and then dynamic motion vectors. Integration of dynamic acceleration provides a velocity profile and then integration of the velocity vector produces position coordinates.

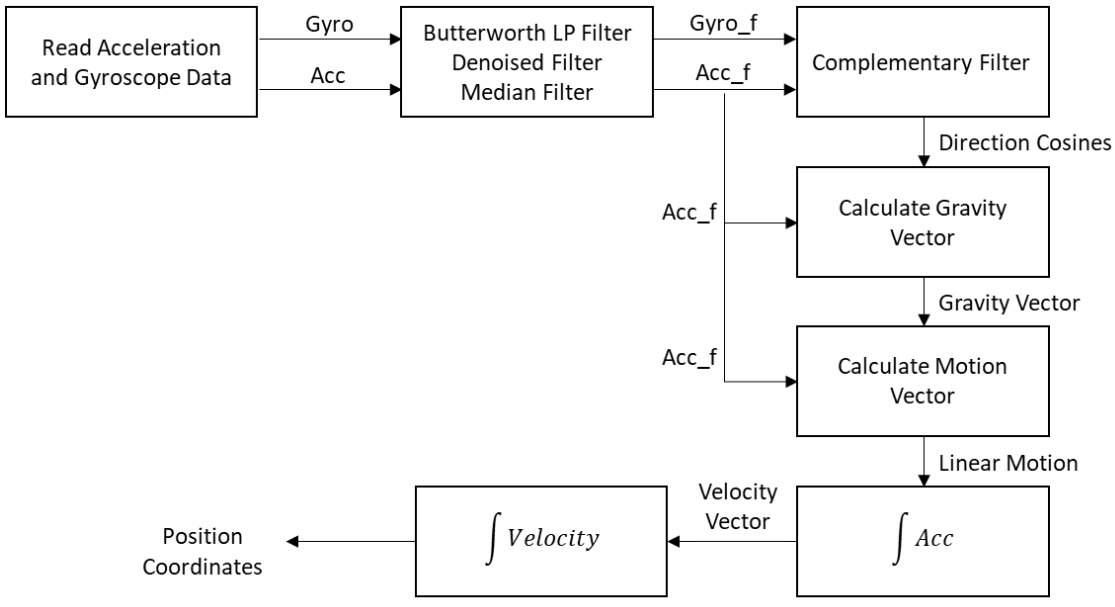

Figure 5. Flowchart of data processing and position calculation

2.1. Data Filtering

Raw data obtained from the sensors are very noisy and need data processing for further use. First step is to filter out the noise from the raw data without losing the shape of the data curve.

The accelerometer sensor measures how fast the velocity of the sensor is being changed over time. It could be measuring the natural vibrations without any actual motion. These vibrations might result in inaccurate position measurement. These vibrations are zero mean white gaussian noise that can be removed by filters such as Butterworth filter, Chebyshev filter (Type I) and Gaussian filter, and a sixth-order Butterworth LP filter is used that provides the best results to retain desirable component of the signal. During the start and the end of the motion or during sudden movements, the accelerometer measures sudden spikes that might include noises, and the median filter can minimize this effect.

The gyroscope sensor measures angular velocity in radians per seconds. Like accelerometer, gyroscope could be affected by natural vibrations and have some noise on its axis. As these noise components are small with respect to the actual angular velocity measured by the gyroscope, these small values can be removed by filters such as median filter, wavelet denoised filter and Butterworth filter, and a wavelet denoised filter is used that provides the best results to keep the desirable component of the signal. For sudden changes in orientation, the gyroscope could measure sudden spikes that result in inaccurate orientation calculations. The median filter can remove these sudden changes in the angular velocity.

Figure 6 shows the raw data collected from an MPU-9255 sensor and the results from the above-mentioned filters. To check the filter response for noises discussed earlier, sensors were placed on a table for about 40 seconds, suddenly moved for about 3 seconds, and static for 10 seconds during the data collection. The top two subfigures in Figure 6 show the raw data with noises from the gyroscope and accelerometer. By applying Butterworth LP filter for acceleration and the denoised wavelet for angular velocity, the noise is reduced without affecting the signal quality and shape. The middle two subfigures show the filtered data. Then the median filter smooths out any sudden spikes as shown in the bottom subfigures.

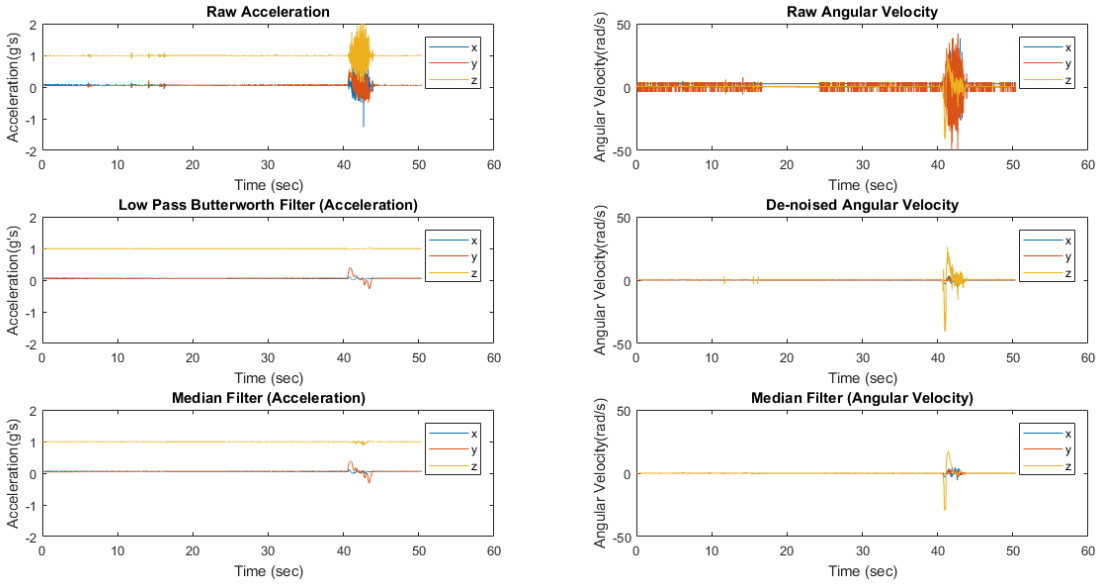

Figure 6. IMU (accelerometer and gyroscope) raw and filtered data

2.2. Calibration

After filtering out the noise, the IMU can have bias in its sensor data. To remove the bias, it is recommended to calibrate the IMU. This was done by customizing the self-calibration technique introduced by Gietzelt *et al.* As the technique presented is best used in case of putting accelerometer at torso height that is the most stable position in human body while moving. To improve the technique so that one can place the IMU in any situation, the data from sensor are recorded for about three seconds before any movement to remove any bias.

2.3. Gravity Vector

Accelerometer data can be divided into static and dynamic components. The static component includes constant force of gravity acting on the sensor, whereas the dynamic component includes sensor movement and vibration. As filtering out the noise from the accelerometer data only removes the noise present in dynamic component, it is necessary to remove the static component from the data to get the actual dynamic movement of the sensor. The static component, i.e., gravity, can be ignored for these scenarios:

- If the sensor has a fixed orientation and can only move in two-dimensional space, the axis that measures the gravity can be totally ignored.
- If the sensor is at static position and only changes its orientation.

In this thesis, the sensor can be in any orientation and can move in three-dimensional space. It is necessary to remove the gravity factor from the accelerometer data. The static component, i.e., gravity, is fixed that is 9.8 m/s^2. Without knowing the device orientation, one cannot remove the static component. Device orientation calculated from

just the accelerometer data during its motion is not reliable as it has the static and the dynamic components, and can result in inaccurate orientation calculation. Although the gyroscope can be used to calculate the orientation of the device, one need to integrate the angular velocity to get angular distance that can be further transformed to get device orientation. This method can cause drift in the calculations of orientation.

To avoid this situation of using accelerometer and gyroscope separately to get reliable orientation, the accelerometer and the gyroscope data are fused to calculate the reliable orientation when the sensor is static or moving. A complimentary filter is used for this purpose with more weight to the gyroscope data during sensor motion and more weight to the accelerometer data during static part where sensor is moving very slow or at rest.

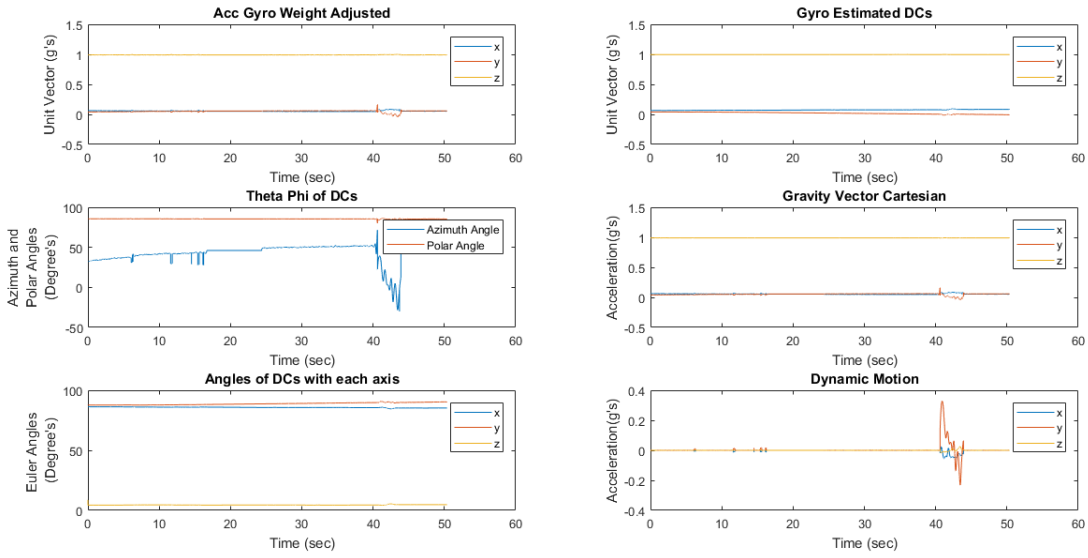

Figure 7. Direction cosines, gravity and dynamic acceleration data

Direction cosines or DC's are calculated separately from the accelerometer and the gyroscope data. The DC's calculated by the gyroscope are shown on the top right subfigure in Figure 7. The top left subfigure shows the weighting factor adjusted by the gyroscope during the motion. As the sensor was at rest for the first 40 seconds, the weighting factor for each axis of the gyroscope was minimal and it increases to adjust the orientation accordingly for three seconds when the sensor is in motion and it moved back to minimal values. The polar angle theta and azimuth phi, and Euler angles of DC's for the fused data after applying weighting factor are shown on the left middle and bottom subfigures. The gravity vector calculated from these DC's in the range of (-g,+g) is displayed on the right middle subfigure. After calculating the gravity vector, it is simply subtracted from the filtered acceleration calculated in the previous step to get the dynamic component of the motion as shown in the right bottom subfigure.

2.4. Position Calculation

Calculating displacement after getting dynamic component seems rather simple by just double integrating it to get the final position but doing just that provides unreliable results and inaccurate position tracking.

The dynamic motion calculated in the last step is first filtered to minimize the noise introduced during its calculation that is not significant as displayed in the bottom left subfigure of Figure 8 but it can result in inaccurate calculations in some cases where change in orientation is significant enough.

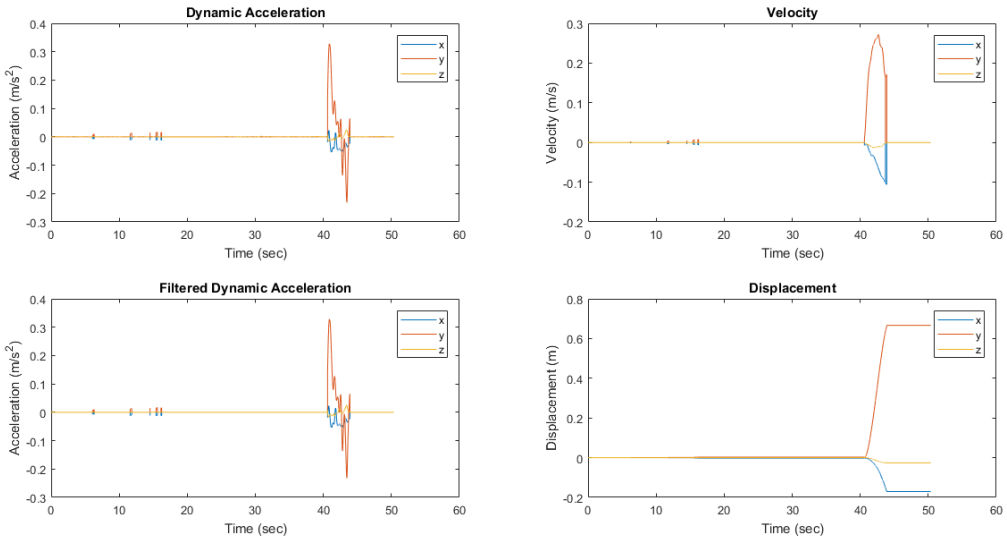

Figure 8. Calculated and filtered dynamic acceleration and velocity and displacement vector

Filtered acceleration is then integrated to the velocity profile of the sensor that can be seen in the right top subfigure of Figure 8. In some cases, it was observed that incorporating three stages of filtering, to minimize the effect of noise on acceleration data, results in asymmetric peak reduction of the start and end of the motion. Ideally, the accelerometer should have net zero value for any motion having same inertial state at its initial and final instances. In some cases, it does not have net zero. To minimize this effect, a threshold was introduced in the gyroscope data to make sure that if the sensor is moving very slowly or at rest, it should report zero velocity at that instant. This effect can also be seen in the right top subfigure. Just before the velocity profile moves back to zero, there is a sudden jump in it as during the motion, this was intentionally introduced to check the behavior of the threshold. At the end, the velocity profile is simply integrated to get the displacement profile that can be seen in the right bottom subfigure.

3. Results

Datasets are collected for several pre-defined movements to evaluate the developed algorithm. The dataset comprises of the sensor measurements and ground truth of actual distance. The results of straight-line and turning motions are discussed below.

3.1. Motion in a Straight-Line

The system setup shown in Figure 4 is moved indoor along a straight line from one point to another for about 4 meters.

3.1.1. MPU-9255

During the prescribed motion, the Raspberry Pi 3B recorded the MPU9255 sensor data. Figure 9 shows the raw and filtered data with the acceleration in the left three subfigures and the angular velocity in the right three subfigures similar to Figure 6.

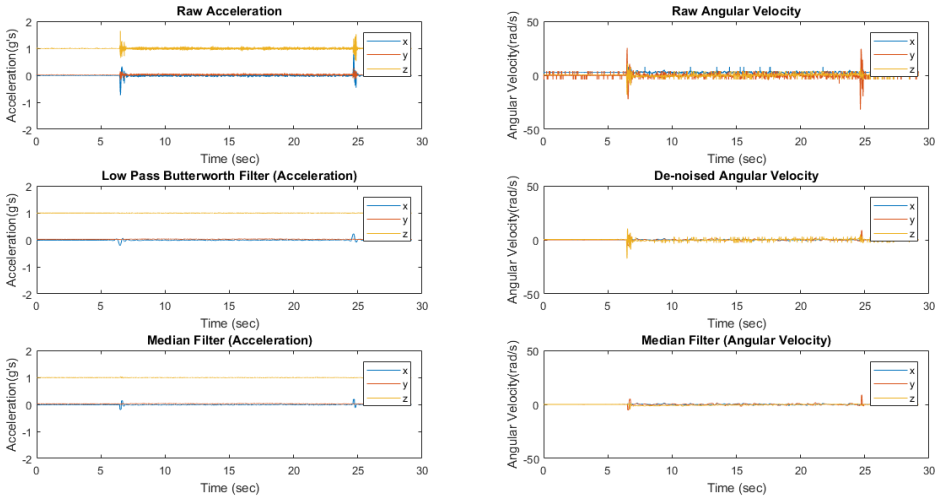

Figure 9. Raw and filtered data from the MPU9255

The sensor recorded sudden changes in acceleration and angular velocity at the start and end of the motion. Since the movement was on the x-y plane, the results on the left show 1g acceleration due to gravity in the z axis. The results on the right show insignificant changes in the angular velocity because the device is not changing its orientation.

Figure 10 shows the gyro adjusted weighting factor, direction cosines (DC's), polar and azimuth angles, gravity vector, Euler angles, and dynamic motion like Figure 7. The gravity vector and Euler angles are unchanged throughout the motion since the movement is on the x-y plane.

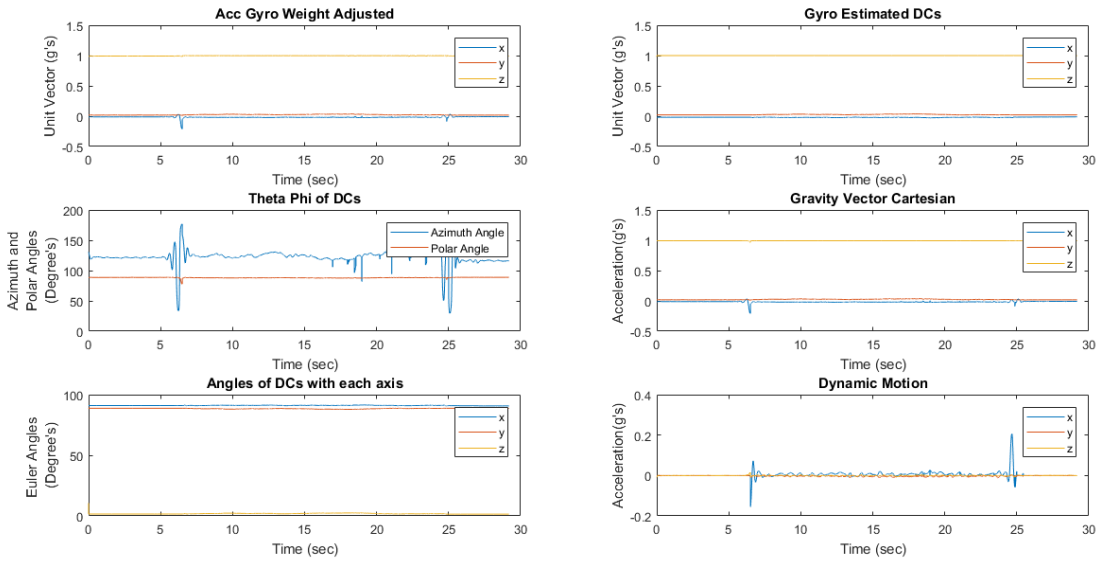

Figure 10. DC's, gravity vector, and dynamic motion data for the MPU9255

The final step was to calculate the velocity profile and then the distance traveled. Figure 11 shows the dynamic component of the acceleration and its corresponding velocity and displacement profiles. The displacement profile indicates zero displacement before the

motion, increasing displacement in x and y directions during the motion, and zero displacement when stopped. Additional data for less than five seconds were logged to make sure that the algorithm provides zero drift after the sensor stopped.

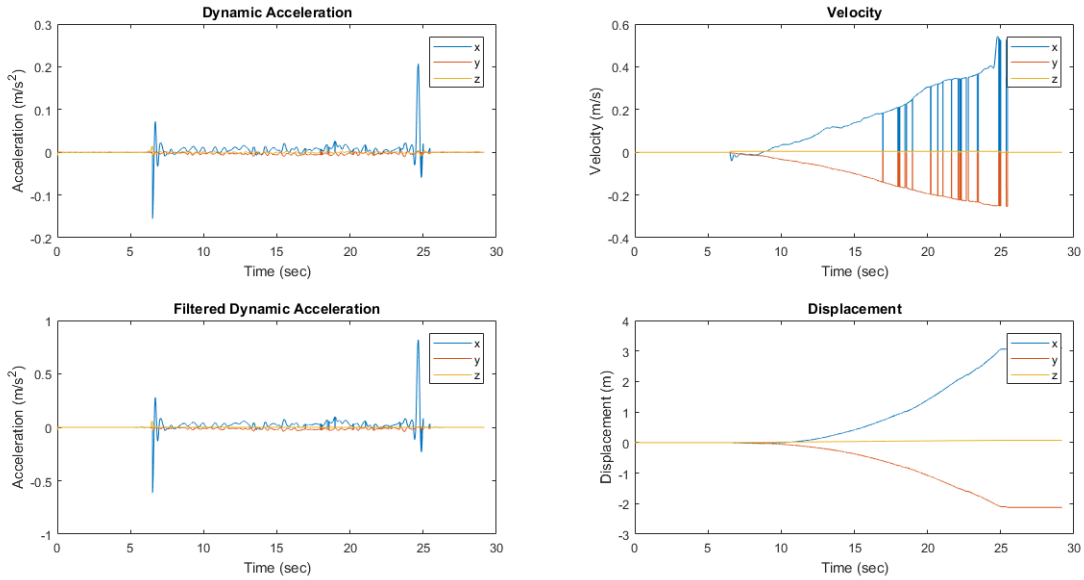

Figure 11. Dynamic acceleration on the left, and velocity and displacement profiles on the right

Figure 12 depicts the motion of the sensor on the Cartesian plane where green dot (upper left) represents the start of the motion and blue dot (lower bottom) represents the end of the motion. The total displacement obtained from the algorithm was 3.76 meters compared to the ground truth of 4 meters with a 6% error.

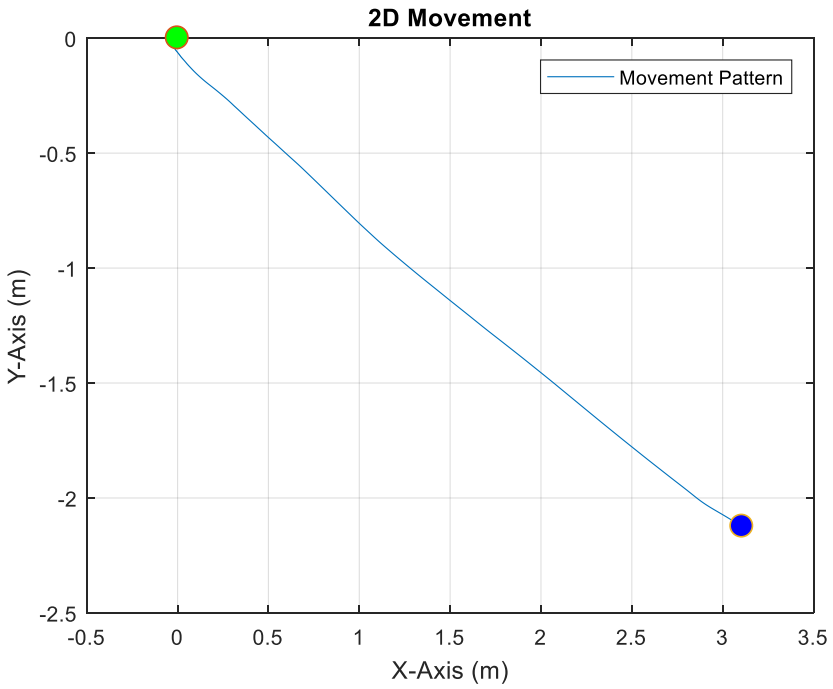

Figure 12. Movement on the Cartesian plane

3.1.2. Jackal Robot IMU Sensor

For the same straight-line motion as discussed above, the data were recorded from the IMU (LORD MicroStrain 3DM-GX3-25) inside the Jackal robot, using the "rosbag" command [16] of the Robot Operating System (ROS). Figure 13 shows the raw and filtered data recorded on the Jackal robot and then processed by MATLAB version 2017a on Windows 10 Education.

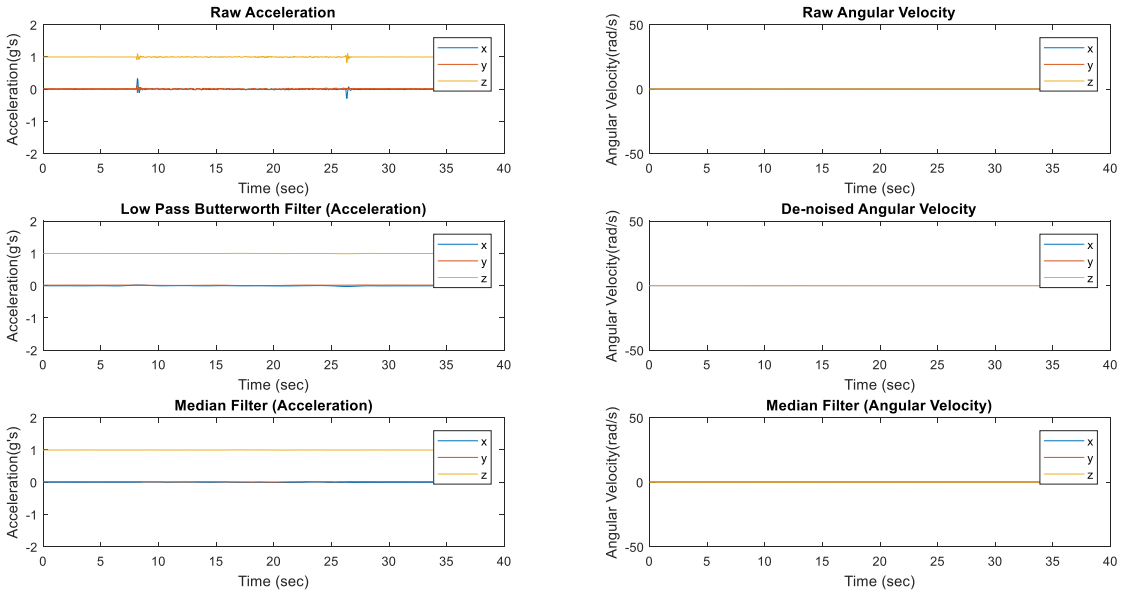

Figure 13. Raw and filtered data from the Jackal Robot

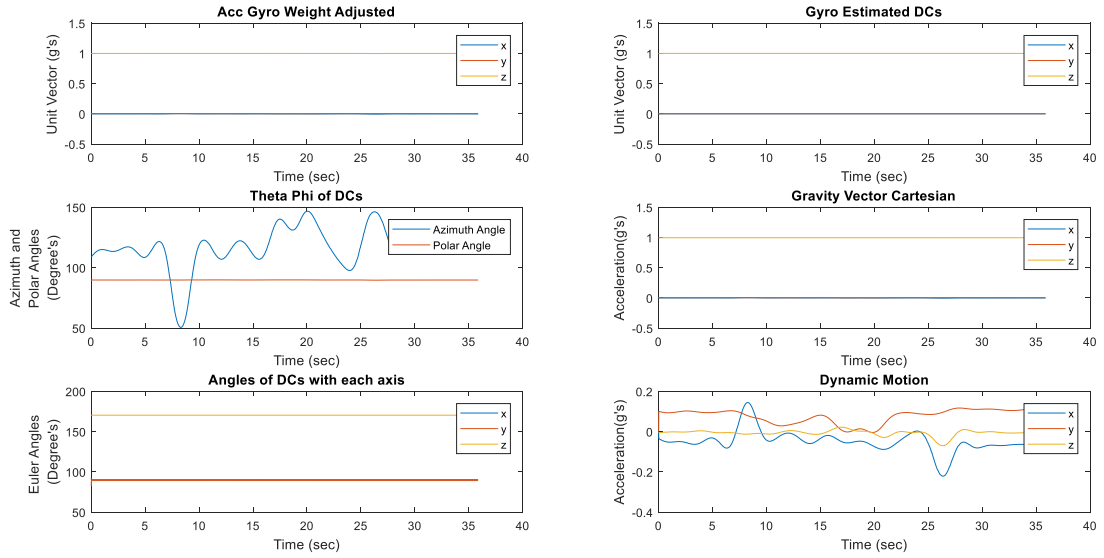

Figure 14. DC's, gravity vector, and dynamic motion vector for the Jackal Robot

Figure 14 shows the gyro adjusted weighting factor, direction cosines (DC's), polar and azimuth angles, gravity vector, Euler angles, and dynamic motion like Figure 7. After filtering the dynamic acceleration, its integral over time results in a velocity profile. Then applying the gyroscope threshold as discussed in Section 2.4, the displacement profile was calculated. Figure 15 shows the dynamic acceleration, velocity, and displacement profiles for the complete motion of the Jackal Robot.

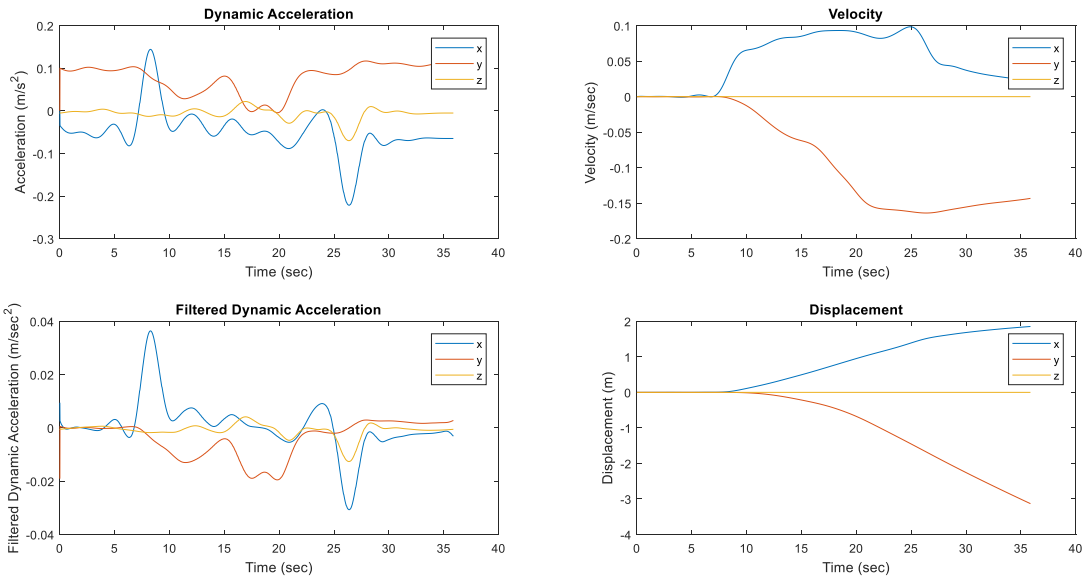

Figure 15. Dynamic acceleration, velocity and displacement profiles from the Jackal Robot

Figure 16 shows the movement of the Jackal Robot on the Cartesian plane where the green dot (upper left) represents the start point of the robot and the blue dot (bottom right) represents the end of the robot movement. The algorithm predicted the total movement of about 3.6445 meters compared to the ground truth of four meters with an

8.8% error. In addition, the odometer on the Jackal Robot reported the total displacement of 3.95 meters.

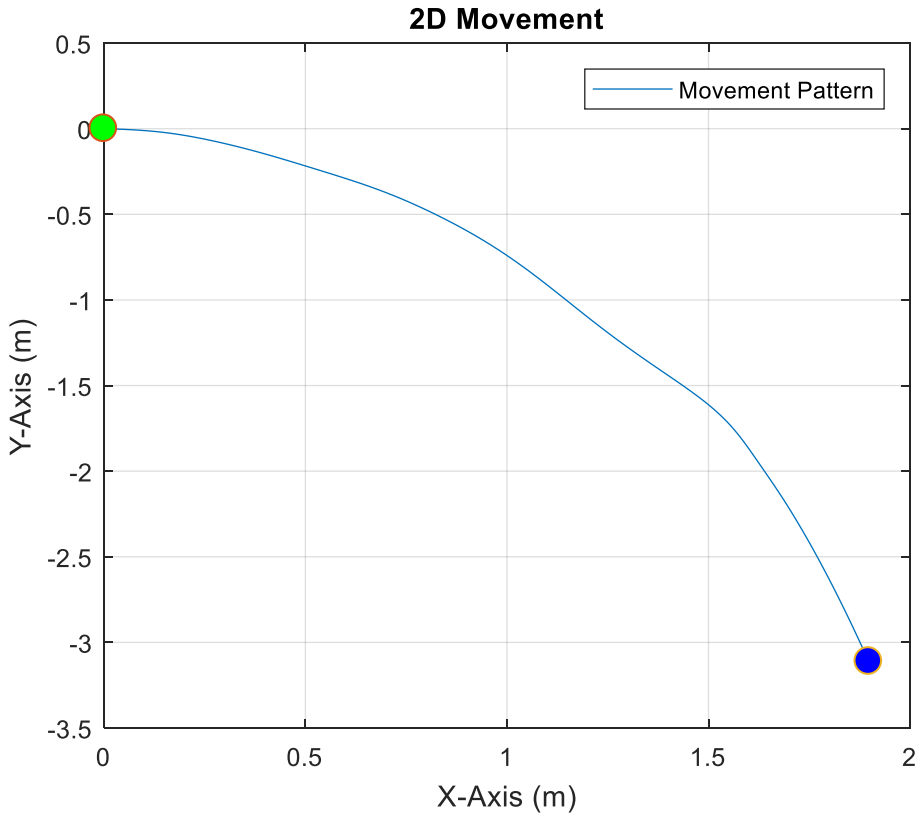

Figure 16. Movement from the Jackal Robot data on the Cartesian plane

3.1.3. Smartphone (iPhone 7)

The iPhone 7 records data from its IMU sensor (6-axis InvenSense MPU-6500 accelerometer and gyroscope, and 3-axis Bosch BMA280 accelerometer) for the same straight-line motion as discussed above, using the MATLAB mobile app [15]. Figure 17 shows the raw and filtered data recorded on the iPhone 7 and processed by MATLAB version 2017a on Windows 10 Education.

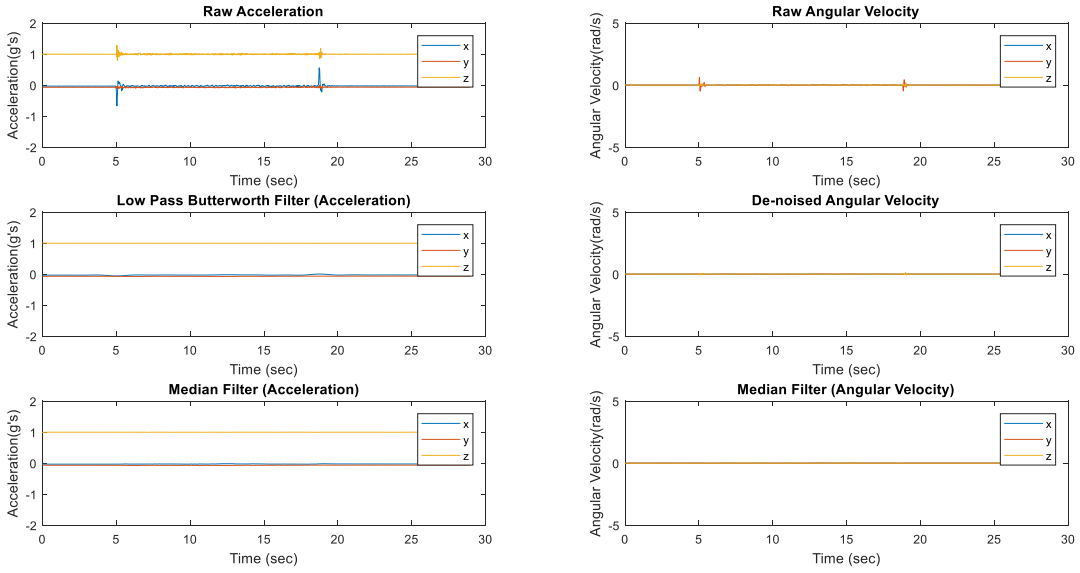

Figure 17. Raw and filtered data from the iPhone 7

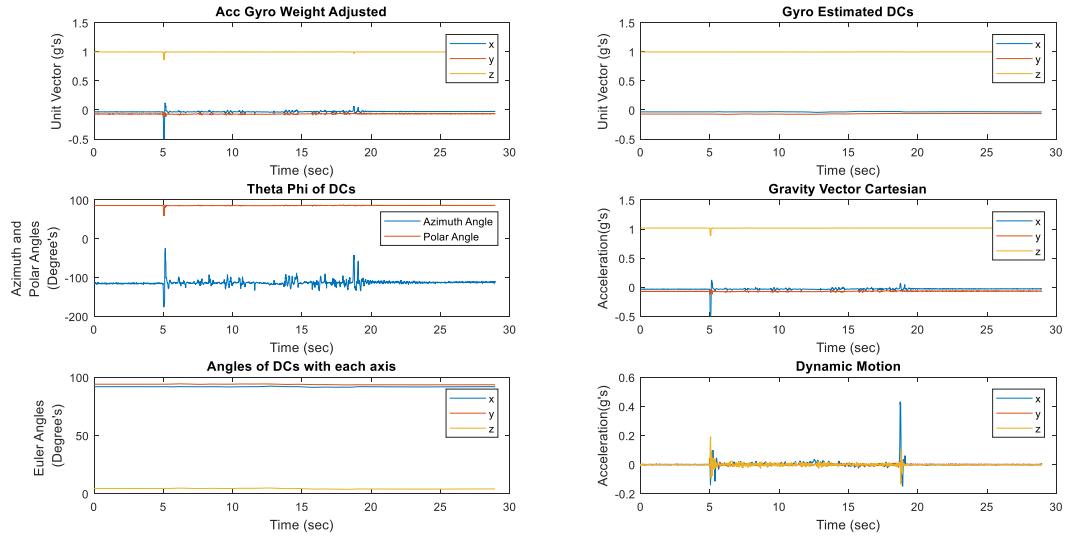

Figure 18. DC's, gravity vector and dynamic motion data for the iPhone 7

Figure 18 shows the gyro adjusted weighting factor, direction cosines (DC's), polar and azimuth angles, gravity vector, Euler angles, and dynamic motion like Figure 7. After filtering the dynamic acceleration, its integral over time results in a velocity profile. The displacement profile was calculated by applying the gyroscope threshold as discussed in Section 2.4. Figure 15 shows the dynamic acceleration, velocity, and displacement profiles for the complete motion of the iPhone 7.

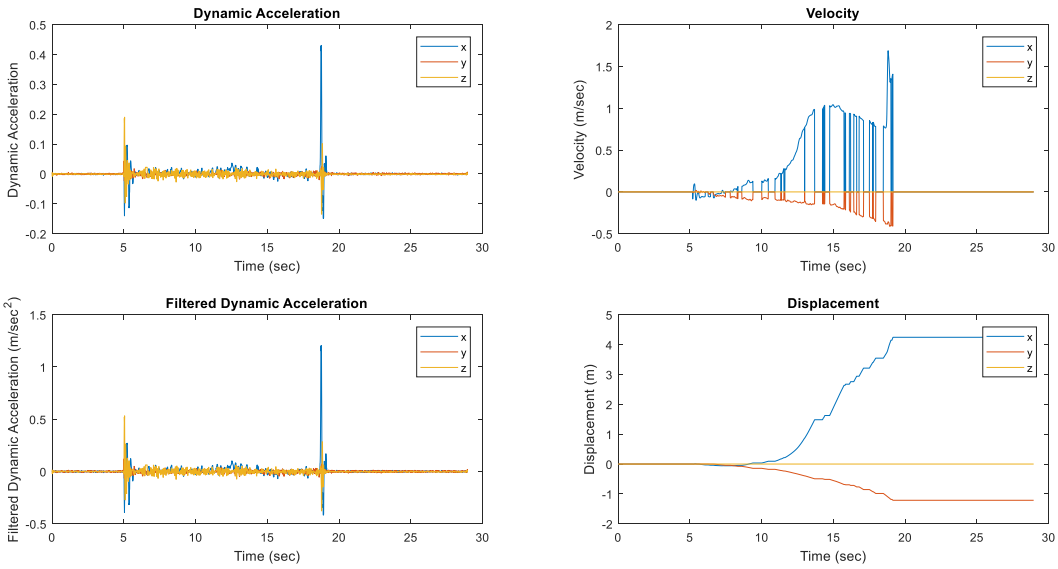

Figure 19. Dynamic acceleration (left side), velocity, and displacement profiles (right side) for the iPhone 7

Figure 20 displays the movement of the iPhone 7 on the Cartesian plane where the green dot (upper left) represents the start point of the smartphone and the blue dot (bottom

right) represents the end of the smartphone movement. The algorithm predicted the total movement of about 4.42 meters compared to the ground truth of four meters with an 10.5% error. The position discrepancy and nonlinear movement pattern are caused by the sampling rate of 10 times per second limited by the iOS MATLAB mobile app.

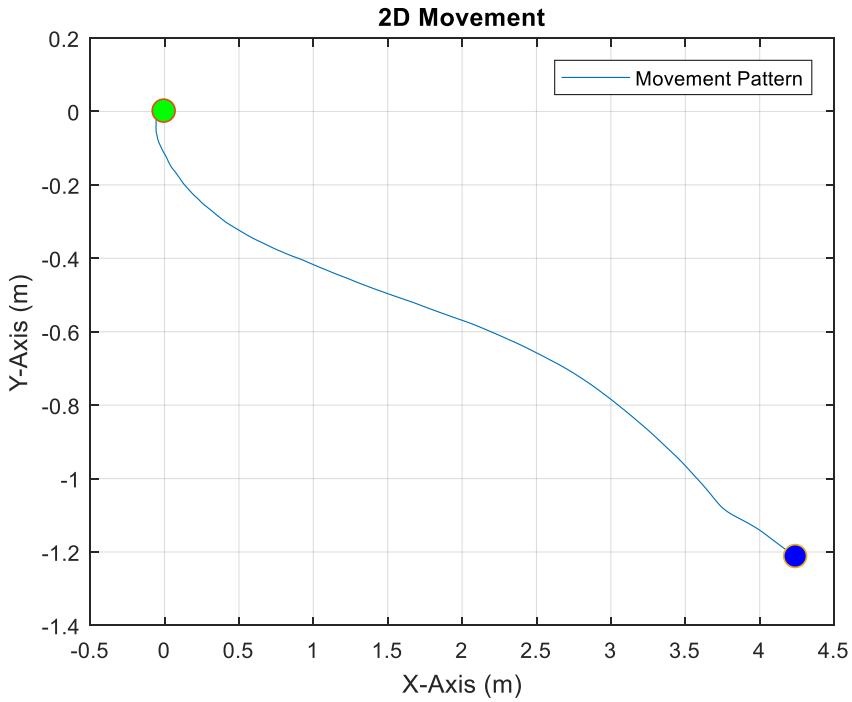

Figure 20. Movement from the iPhone 7 data on the Cartesian plane

3.2. Turning Motion

The system setup shown in Figure 4 is moved indoor along a straight line for about four meters, turned right, then moved along the straight line for about four meters, covering a total distance of about eight meters.

3.2.1. MPU-9255

The Raspberry Pi 3B recorded the MPU9255 sensor data for the above-mentioned motion. Figure 21 shows the raw and filtered data with the acceleration in the left three subfigures and the angular velocity in the right three subfigures similar to Figure 6. The total movement can be divided into five regions: static position before start, motion in the straight-line, turning right while standing, moving again in straight-line, and static after stopped. The sensor recorded sudden changes in acceleration and angular velocity at the start, middle when turning, and end of the motion. Since the movement was on the x-y plane, the results on the left show 1g acceleration due to gravity in the z axis.

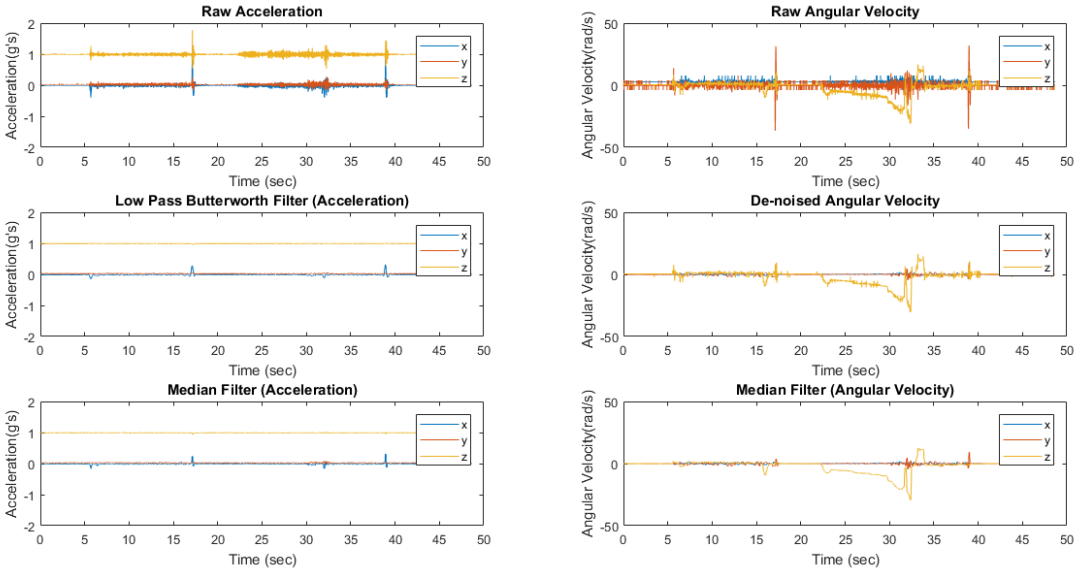

Figure 21. Raw and filtered MPU9255 Data

Figure 22 shows the gyro adjusted weighting factor, direction cosines (DC's), polar and azimuth angles, gravity vector, Euler angles, and dynamic motion like Figure 7. The gravity vector and Euler angles are unchanged throughout the motion since the movement is on the x-y plane. Figure 22 right bottom subfigure clearly shows two parts of motion including the movement of the sensor before and after turning.

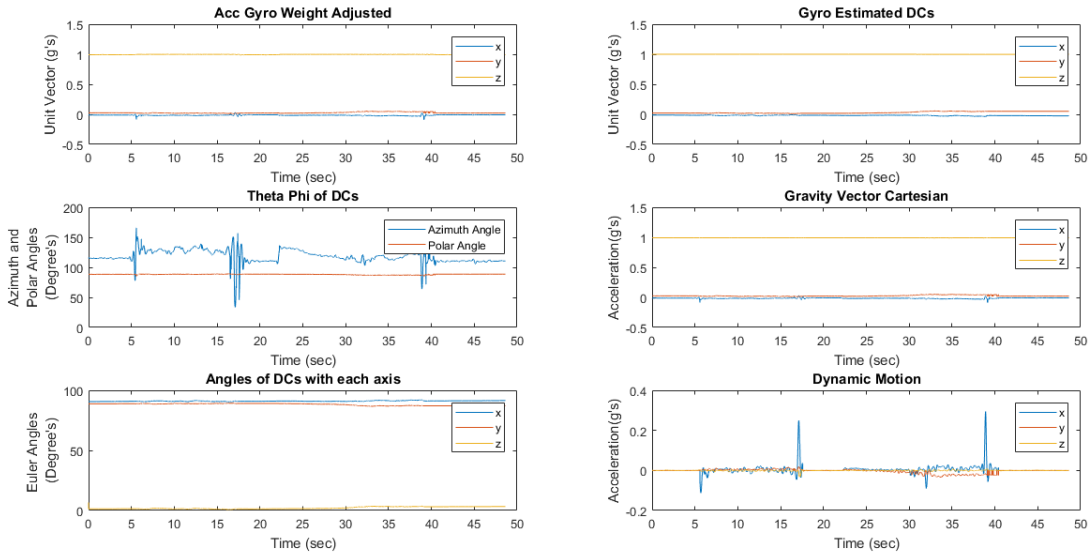

Figure 22. DC's, gravity vector, and dynamic motion data for the MPU9255

Figure 23 displays the dynamic component of the acceleration, and velocity and displacement profiles. The displacement profile indicates zero displacement before the motion, changing displacement in x and y directions during the motion, zero displacement when stopped for turning, changing displacement in x and y directions during the next motion again, and then zero displacement when stopped. Additional data for less than ten seconds were logged to make sure that the algorithm provides zero drift after the sensor

stopped. The algorithm predicted the total distance of 8.1659 meters compared to the ground truth of 8 meters with a 2.07% error.

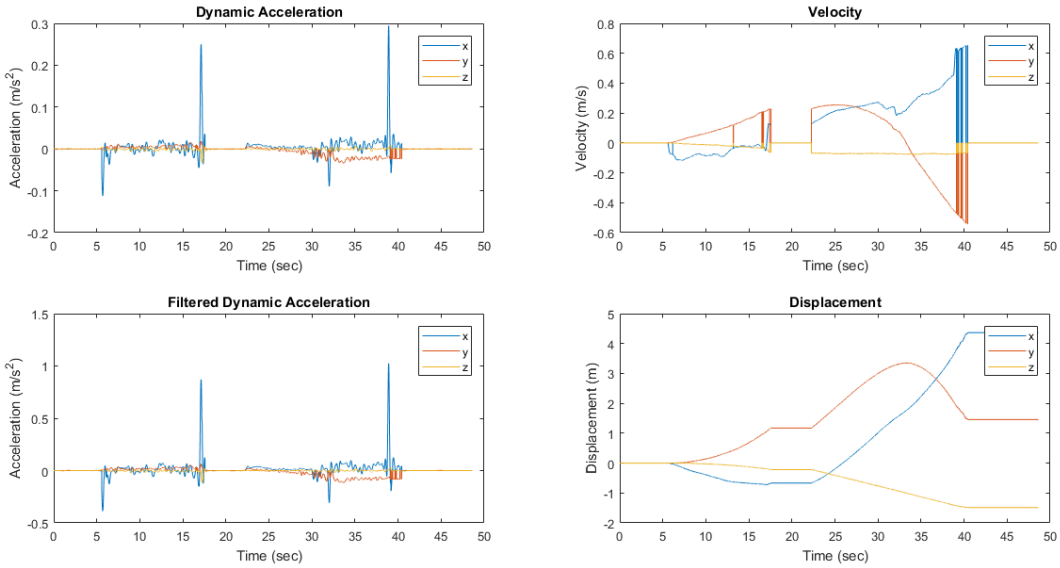

Figure 23. Dynamic acceleration, velocity and displacement Profile for the MPU9255

3.2.2. Jackal Robot IMU Sensor

For the same turning motion as discussed above, the data were recorded from the IMU (LORD MicroStrain 3DM-GX3-25) inside the Jackal robot. Figure 24 shows the raw and filtered data recorded on the Jackal robot and processed by MATLAB. Figure 25 shows the gyro adjusted weighting factor, direction cosines (DC's), polar and azimuth angles, gravity vector, Euler angles, and dynamic motion like Figure 7.

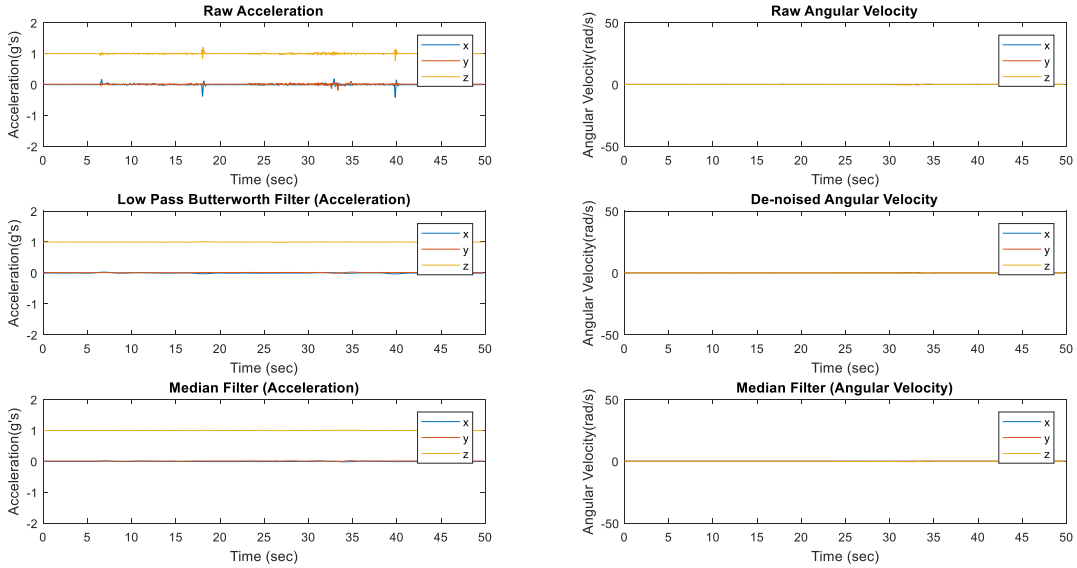

Figure 24. Raw and filtered data from the Jackal Robot

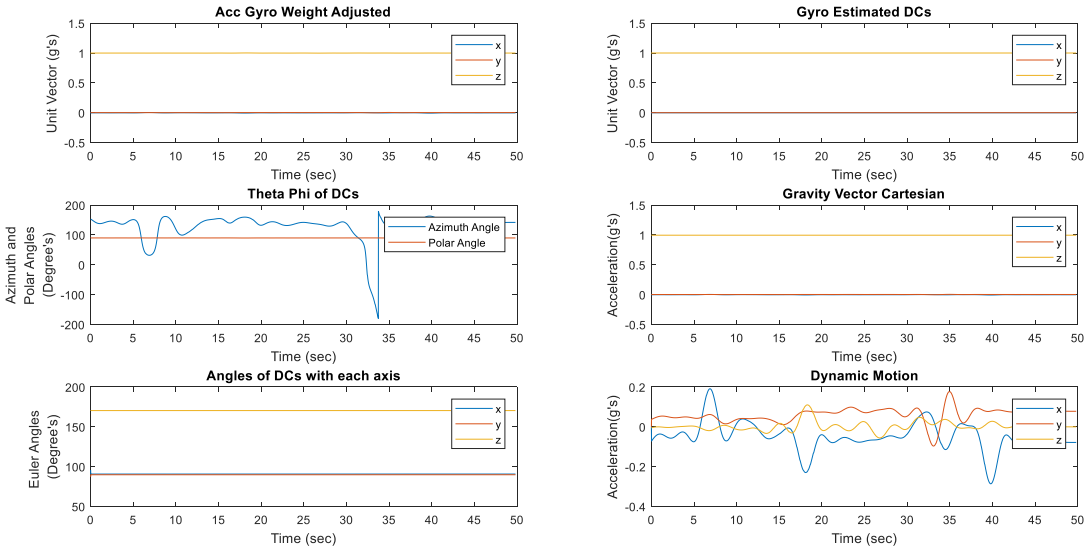

Figure 25. DC's, gravity vector, and dynamic motion vector for the Jackal Robot

After filtering the dynamic acceleration, its integral over time results in a velocity profile. Then applying the gyroscope threshold as discussed in Section 2.4, the displacement profile was calculated. *Figure 26* shows the dynamic acceleration, velocity, and displacement profiles for the complete motion of the Jackal Robot for the turning motion. The algorithm predicted the total distance of 6.86 meters compared to the ground truth of 8 meters with a 14.25% error. In addition, the odometer on the Jackal Robot reported the total displacement of 7.92 meters.

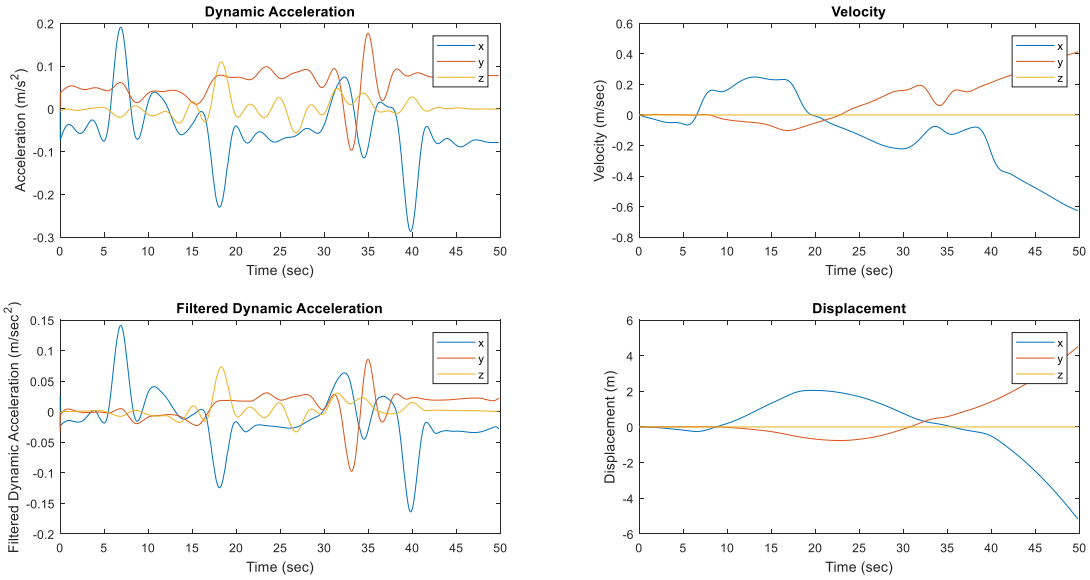

Figure 26. Dynamic acceleration, and velocity and displacement profile

3.2.3. Smartphone (iPhone 7)

The iPhone 7 records data from its IMU sensor (6-axis InvenSense MPU-6500 accelerometer and gyroscope, and 3-axis Bosch BMA280 accelerometer) for the same turning motion as discussed above. Figure 27 shows the raw and filtered data recorded on

the iPhone 7 and processed by MATLAB. Figure 28 shows the gyro adjusted weighting factor, direction cosines (DC's), polar and azimuth angles, gravity vector, Euler angles, and dynamic motion like Figure 7. The angular velocity shows significant changes when the iPhone 7 was turning as shown in the right three subfigures.

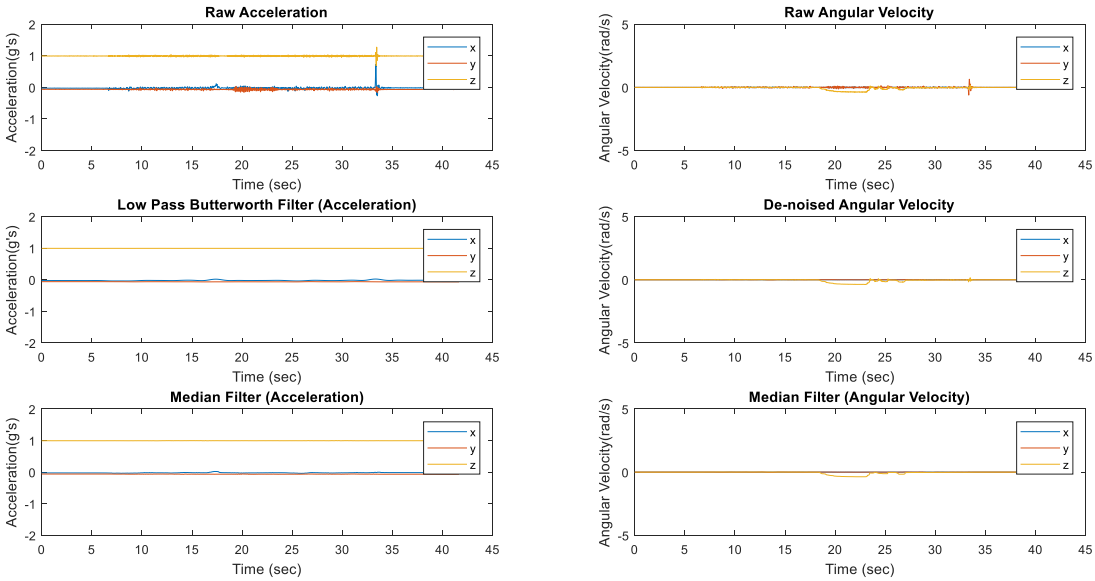

Figure 27. Raw and filtered data from the iPhone 7

After filtering the dynamic acceleration, its integral over time results in a velocity profile. The displacement profile was calculated by applying the gyroscope threshold as discussed in Section 2.4. Figure 29 shows the dynamic acceleration, and velocity and displacement profiles for the complete motion of the iPhone 7. The algorithm predicted the total distance of 7.79 meters compared to the ground truth of 8 meters with a 2.62% error.

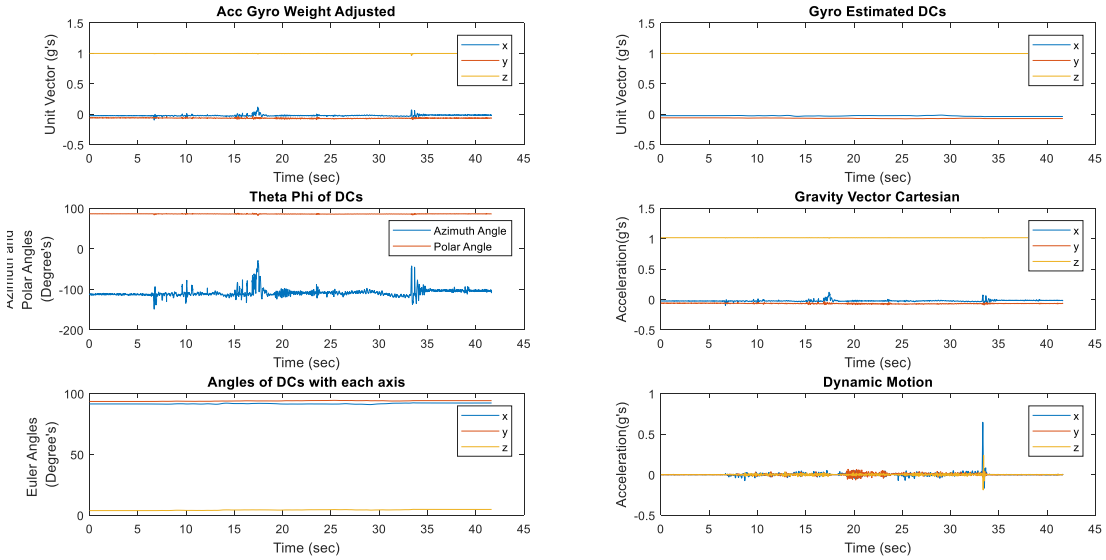

Figure 28. DC's, gravity vector and dynamic motion data for the iPhone 7

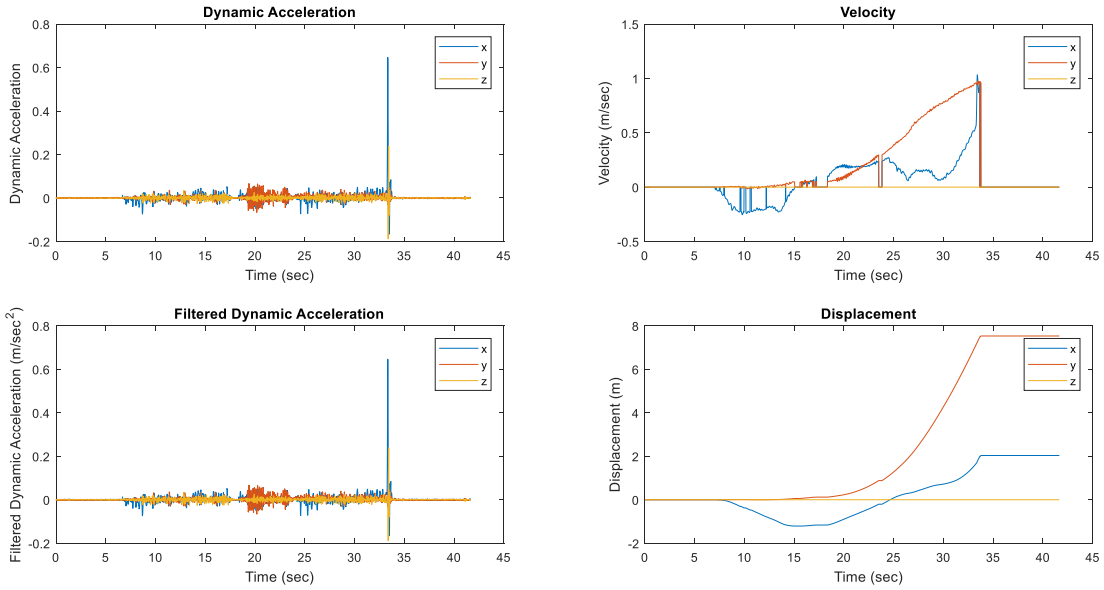

Figure 29. Dynamic acceleration (left side), velocity, and displacement profiles (right side) for the iPhone 7

Tables 1 to 3 compare the predicted distance with the ground truth for MPU9255, Jackal Robot, and iPhone 7, respectively. The proposed algorithm gave the error between 0.24 to 0.42 meters for 4 meters of ground truth, and 2.1 to 3.73 meters for 25 meters of ground truth from more than ten datasets.

Table 1. Results for the MPU9255

Ground Truth (meters)	Predicted Distance (meters)	Error (meters)	Error (%)
4	4.346	0.35	8.65
4	3.7643	0.24	5.89
4	3.6499	0.35	8.75
8	8.1659	0.17	2.07
25	23.2645	1.74	6.94
25	22.8998	2.1	8.4

Table 2. Results for the Jackal Robot

Ground Truth (meters)	Predicted Distance (meters)	Error (meters)	Error (%)
4	3.6445	0.36	8.88
5	4.3694	0.63	12.61
8	6.86	1.14	14.25
25	28.1057	3.11	12.42

Table 3. Results for the iPhone 7

Ground Truth (meters)	Predicted Distance (meters)	Error (meters)	Error (%)
4	4.42	0.42	10.5
5	5.4202	0.42	8.4
8	7.79	0.21	2.62
13	14.3845	1.38	10.65
25	21.269	3.73	14.92

4. Cost-Performance Tradeoff

Although the inertial sensors used in this thesis are noisy and do not have adequate dynamic ranges, they are small and consume low power, making it suitable for indoor localization. Reduction in physical size or power consumption can compromise performance in contrast to expensive devices that are large and consume more power. One needs to periodically calibrate low-cost devices to remove bias. In particular, the gyroscope bias instability is significant in low-cost MEMS-based IMU [17] as shown in Table 4.

Table 4. Gyro grade based on bias stability

Performance grade	**Gyro Bias Stability (º/hr)**
Consumer	30-1000
Industrial	1-30
Tactical	0.1-30
High-end tactical	0.1-1
Navigation	0.01-0.1
Strategic	0.0001-0.01

Filtering out the noise and bias from the IMU can attenuate the desirable signal and lead to inaccurate localization. Different thresholds and data fusion are applied to minimize the effect as discussed in Chapter 2.

5. Recommendations

This thesis presented an indoor localization technique that can be used in existing systems to improve their indoor location accuracy. Using a high-performance IMU can help obtain more accurate results than those from a low-cost IMU used in this thesis. Potential applications are as follows:

- The proposed algorithm can be used in mobile devices to predict user locations inside a building or structure.
- The algorithm can be incorporated inside a robot to predict locations in an indoor environment. The Jackal Robot position estimation using IMU data is not reliable [18] and can be improved by the proposed algorithm.
- It can be fused with GPS tracking devices to predict locations when GPS data is not available.

6. Summary and Conclusions

This thesis provides a robust localization technique based on dead reckoning method using low-cost inertial sensors that can work in any environment. The technique comprises of collecting data from motion sensors, filtering and calibrating the data and calculating gravity vector and the velocity and displacement profiles. The proposed algorithm gave the error between 0.24 to 0.42 meters for 4 meters of ground truth, and 2.1 to 3.73 meters for 25 meters of ground truth from more than twelve datasets. The comparison is also presented by implementing the proposed algorithm on data collected from devices including a standalone MPU9255, the Jackal Robot, and the iPhone 7.

For the data collected from the iPhone 7, the sampling frequency is limited to 10 times per second by the MATALB mobile app. This sampling frequency is not enough for the proposed algorithm to predict accurate locations compared to the standalone MPU9255 and the Jackal Robot that have more than 40 times per second. The technique shows similar location accuracy with respect to other indoor localization techniques that require equipment installation, thus providing an inexpensive solution for indoor localization.

Appendices

Appendix A. PYTHON CODE – To log MPU-9255 data in Raspberry Pi 3

```
#!/usr/bin/env python3

#Libraries and functions needed
import smbus            #For using I2C bus
import math
import gpsd             #For GPS Module
import datetime
from time import sleep
import RPi.GPIO as GPIO

#Initiliazation of I2C bus
bus = smbus.SMBus(1)

#Status LED initiliazation
GPIO.setmode(GPIO.BCM)
LIGHT_PIN = 25

#Device Addresses
address = 0x68      # Sensor I2C address
address_mag = 0x0c  #Sensor address for magnetometer

#Control Addresses
power_mgmt_1 = 0x6b

# Register address from MPU 9255 register map for Accelerometer
accel_config = 0x1c
accel_xout_h = 0x3b
accel_yout_h = 0x3d
accel_zout_h = 0x3f

# Register address from MPU 9255 register map for Gyroscope
gyro_config = 0x1b
gyro_xout_h = 0x43
gyro_yout_h = 0x45
gyro_zout_h = 0x47

# Register address from MPU 9255 register map for Magnetometer
usr_cntrl = 0x6a
int_pin_conf = 0x37
cntrl = 0x0a
mag_xout_l = 0x03
mag_yout_l = 0x05
mag_zout_l = 0x07

"""
            Common Functions to be used
"""

def read_byte(address, adr):
            return bus.read_byte_data(address, adr)

def read_word(adr):
            high = bus.read_byte_data(address, adr)
            low = bus.read_byte_data(address, adr+1)
            val = (high << 8) + low
            return val

def read_word_2c(adr):
            val = read_word(adr)
```

```
        if (val >= 0x8000):
                return -((65535 - val) + 1)
        else:
                return val

def read_word_mag(address, adr):
        low = read_byte(address, adr)
        high = read_byte(address, adr+1)
        val = (high << 8) + low
        return val

def read_word_2c_mag(address, adr):
        val = read_word_mag(address, adr)
        if (val >= 0x8000):
                return -((65535 - val) + 1)
        else:
                return val

"""
        Accelerometer Data
"""
def acceleromter():
        try:

                # Setting power register to start getting sesnor data
                bus.write_byte_data(address, power_mgmt_1, 0)
                # Setting Acceleration register to set the sensitivity
                # 0,8,16 and 24 for 16384,8192,4096 and 2048 sensitivity respectively
                bus.write_byte_data(address, accel_config, 8) # for +/- 4g

                accel_xout = read_word_2c(accel_xout_h) #We just need to put H byte address
                accel_yout = read_word_2c(accel_yout_h) #as we are reading the word data
                accel_zout = read_word_2c(accel_zout_h)

                accel_xout_scaled = accel_xout / 8192.0 #According to the sensitivity you set
                accel_yout_scaled = accel_yout / 8192.0
                accel_zout_scaled = accel_zout / 8192.0
                return accel_xout_scaled,accel_yout_scaled,accel_zout_scaled
        except Exception as e:
                return 'accel_xout_scaled','accel_yout_scaled','accel_zout_scaled'

"""
        Gyroscope Data
"""
def gyroscope():
        try:

                # Setting power register to start getting sesnor data
                bus.write_byte_data(address, power_mgmt_1, 0)

                # Setting gyroscope register to set the sensitivity
                # 0,8,16 and 24 for 131,65.5,32.8 and 16.4 sensitivity respectively
                bus.write_byte_data(address, gyro_config, 8)

                gyro_xout = read_word_2c(gyro_xout_h) #We just need to put H byte address
                gyro_yout = read_word_2c(gyro_yout_h) #as we are reading the word data
                gyro_zout = read_word_2c(gyro_zout_h)

                gyro_xout_scaled = gyro_xout / 65.5 #According to the sensitivity you set
                gyro_yout_scaled = gyro_yout / 65.5
                gyro_zout_scaled = gyro_zout / 65.5

                return gyro_xout_scaled, gyro_yout_scaled, gyro_zout_scaled
        except Exception as e:
                return 'gyro_xout_scaled', 'gyro_yout_scaled', 'gyro_zout_scaled'
```

```
"""
        Magnetometer Data
"""
def magnetometer():
        try:

                #initialize the power registers to wake up sensor
                bus.write_byte_data(address, power_mgmt_1, 0)

                #disable master i2c mode of sensor
                bus.write_byte_data(address, usr_cntrl, 0)

                # enable bypass mode to read directly from magnetometer
                bus.write_byte_data(address, int_pin_conf, 2)

                # setup magnetic sensors for contiuously reading data
                bus.write_byte_data(address_mag, 0x0a, 18)

                mag_xout = read_word_2c_mag(address_mag, mag_xout_l)
                mag_yout = read_word_2c_mag(address_mag, mag_yout_l)
                mag_zout = read_word_2c_mag(address_mag, mag_zout_l)

                mag_xout_scaled = mag_xout * 0.6 #From data sheet, there is only
                mag_yout_scaled = mag_yout * 0.6 #one sensitivity level for
                mag_zout_scaled = mag_zout * 0.6 #Magnetometer in MPU 9255 (Multiply as described in datasheet)

                return mag_xout_scaled, mag_yout_scaled, mag_zout_scaled
        except Exception as e:
                return 'mag_xout_scaled', 'mag_yout_scaled', 'mag_zout_scaled'

"""
        GPS Data
"""
def gps():
        try:

                # Connect to the local gpsd
                gpsd.connect()

                # Get gps position
                packet = gpsd.get_current()
                #pos = packet.position()
                [lat, lng ]=packet.position()

                #Available Data from the packet
                #print(packet.position())
                #print(packet.altitude())
                #print(packet.movement())
                #print(packet.speed_vertical())
                #print(packet.speed())
                #print(packet.position_precision())
                #print(packet.map_url())
                #print(packet.time())
                return lat, lng
        except  Exception as e:
                return 'lat', 'lng'

"""
        Data Collection and Logging Module
"""
String_head = "Time" + "\t" + \
                "Position_lat" +"\t" + \
                "Position_lng" +"\t" + \
                "Accelerometer_x" +"\t" + \
                "Accelerometer_y" +"\t" + \
                "Accelerometer_z" +"\t" + \
```

```
                        "Gyroscope_x" +"\t" + \
                        "Gyroscope_y" +"\t" + \
                        "Gyroscope_z" +"\t" + \
                        "Magnetometer_x" + '\t' + \
                        "Magnetometer_y" + '\t' + \
                        "Magnetometer_z" + '\n'

datafile = datetime.datetime.now().strftime("%Y%m%d-%H%M")
with open(datafile + '.txt','a',) as e:
                        e.write(String_head)
print ("Start Getting data")

while True:
        try:
                        PIN = 25
                        GPIO.setup(PIN, GPIO.OUT)
                        GPIO.output(PIN, False)
                        time = datetime.datetime.now()
                        [lat, lng] = gps()
                        [Accelerometer_x, Accelerometer_y, Accelerometer_z] = acceleromter()
                        [Gyroscope_x, Gyroscope_y, Gyroscope_z] = gyroscope()
                        sleep(0.005)            #Checking Magnetometer response if wait
                        [Magnetometer_x, Magnetometer_y, Magnetometer_z] = magnetometer()

                        '''String = "Time:" + "\t" + str(time) +"\t" + \
                                        "Position:" + str(pos) +"\t" + \
                                        "Accelerometer:" + str(acc) +"\t" + \
                                        "Gyroscope:" + str(gyro) +"\t" + \
                                        "Magnetometer:" + str(mag) + '\n'
                        '''
                        String_body = str(time) + "\t" + \
                                str(lat) +"\t" + \
                                str(lng) +"\t" + \
                                str(Accelerometer_x) +"\t" + \
                                str(Accelerometer_y) +"\t" + \
                                str(Accelerometer_z) +"\t" + \
                                str(Gyroscope_x) +"\t" + \
                                str(Gyroscope_y) +"\t" + \
                                str(Gyroscope_z) +"\t" + \
                                str(Magnetometer_x) + '\t' + \
                                str(Magnetometer_y) + '\t' + \
                                str(Magnetometer_z) + '\n'

                        with open(datafile + '.txt','a') as e:
                                e.write(String_body)
                        GPIO.setup(PIN, GPIO.OUT)
                        GPIO.output(PIN, True)
                        print ('Data is being recorded')
                        sleep(0.005)            #Wait for next iteration
                        #print (String)
                        #sleep(2)
        except KeyboardInterrupt:
                        GPIO.cleanup()
                        exit()
```

Appendix B. MATLAB CODE - IMU data from MPU-9255 and Raspberry Pi 3

```
clear all
close all

% File_name in Raspberry Pi 3 for logged data of MPU9255
filename = '20171113-1705.txt';

if(~exist(filename,'file'))
    if(~exist('mypi','var'))
        %Raspberry Pi IP, usernmae and password
        mypi = raspi('IP_address','username','password');
    end
    raspLocation = '/home/pi/mpuu9255/MPU9255/';
    srcFile = [raspLocation,filename];
    % For importing file from Raspberry Pi
    getFile(mypi,srcFile);

    if(exist('mypi','var'))
        clear mypi;
    end
end
%% Initialize variables.
delimiter = '\t';
startRow = 2;
%% Read columns of data as text:
formatSpec = '%s%*s%*s%s%s%s%s%s%s%s%[^\n\r]';
%% Open the text file.
fileID = fopen(filename,'r');
%% Read columns of data according to the format.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'TextType', 'string',
'HeaderLines' ,startRow-1, 'ReturnOnError', false, 'EndOfLine', '\r\n');
%% Close the text file.
fclose(fileID);
%% Convert the contents of columns containing numeric text to numbers.
% Replace non-numeric text with NaN.
raw = repmat({''},length(dataArray{1}),length(dataArray)-1);
for col=1:length(dataArray)-1
    raw(1:length(dataArray{col}),col) = mat2cell(dataArray{col},
ones(length(dataArray{col}), 1));
end
numericData = NaN(size(dataArray{1},1),size(dataArray,2));

for col=[2,3,4,5,6,7]
    % Converts text in the input cell array to numbers. Replaced non-numeric
    % text with NaN.
    rawData = dataArray{col};
    for row=1:size(rawData, 1)
        % Create a regular expression to detect and remove non-numeric prefixes and
        % suffixes.
        regexstr = '(?<prefix>.*?)(?<numbers>([-]*(\d+[\,]*)+[\.]{0,1}\d*[eEdD]{0,1}[-
+]*\d*[i]{0,1})|([-]*(\d+[\,]*)*[\.]{1,1}\d+[eEdD]{0,1}[-+]*\d*[i]{0,1}))(?<suffix>.*)';
        try
            result = regexp(rawData(row), regexstr, 'names');
            numbers = result.numbers;

            % Detected commas in non-thousand locations.
            invalidThousandsSeparator = false;
            if numbers.contains(',')
                thousandsRegExp = '^\d+?(\,\d{3})*\.{0,1}\d*$';
                if isempty(regexp(numbers, thousandsRegExp, 'once'))
                    numbers = NaN;
                    invalidThousandsSeparator = true;
                end
            end
            % Convert numeric text to numbers.
```

```
            if ~invalidThousandsSeparator
                numbers = textscan(char(strrep(numbers, ',', '')), '%f');
                numericData(row, col) = numbers{1};
                raw{row, col} = numbers{1};
            end
        catch
            raw{row, col} = rawData{row};
        end
    end
end

% Convert the contents of columns with dates to MATLAB datetimes using the
% specified date format.
try
    dates{1} = datetime(dataArray{1}, 'Format', 'yyyy-MM-dd HH:mm:ss.SSSSSS',
'InputFormat', 'yyyy-MM-dd HH:mm:ss.SSSSSS');
catch
    try
        % Handle dates surrounded by quotes
        dataArray{1} = cellfun(@(x) x(2:end-1), dataArray{1}, 'UniformOutput', false);
        dates{1} = datetime(dataArray{1}, 'Format', 'yyyy-MM-dd HH:mm:ss.SSSSSS',
'InputFormat', 'yyyy-MM-dd HH:mm:ss.SSSSSS');
    catch
        dates{1} = repmat(datetime([NaN NaN NaN]), size(dataArray{1}));
    end
end

anyBlankDates = dataArray{1} == '';
anyInvalidDates = isnan(dates{1}.Hour) - anyBlankDates;
dates = dates(:,1);

%% Split data into numeric and string columns.
rawNumericColumns = raw(:, [2,3,4,5,6,7]);

%% Exclude rows with non-numeric cells
I = ~all(cellfun(@(x) (isnumeric(x) || islogical(x)) && ~isnan(x),rawNumericColumns),2);
%Find rows with non-numeric cells
K = I | anyInvalidDates | anyBlankDates;
NumOfRowsExcluded = sum(K)
[~,rowsExcludedIndex] = max(K)
dates = cellfun(@(x) x(~K,:), dates, 'UniformOutput', false);
rawNumericColumns(K,:) = [];

%% Create output variable
Untitled = table;
Untitled.Time = dates{:, 1};
Untitled.Accelerometer_x = cell2mat(rawNumericColumns(:, 1));
Untitled.Accelerometer_y = cell2mat(rawNumericColumns(:, 2));
Untitled.Accelerometer_z = cell2mat(rawNumericColumns(:, 3));
Untitled.Gyroscope_x = cell2mat(rawNumericColumns(:, 4));
Untitled.Gyroscope_y = cell2mat(rawNumericColumns(:, 5));
Untitled.Gyroscope_z = cell2mat(rawNumericColumns(:, 6));
```

Appendix C. MATLAB CODE - IMU data from the Jackal Robot

```
close all
clear all

tic
bagFilename = 'imu_1311_1.bag'; %Filename to be imported
if(~exist(bagFilename,'file'))
    if(~exist('rosDevice1','var'))
        %Jackal Robot IP, usernmae and password
        rosDevice = rosdevice('IP_address','username','password');
    end
    getFile(rosDevice,['~/', bagFilename]);
    toc
    disp('File Received');
end
rosBagAll = rosbag(bagFilename);
disp('File Imported');
% For getting IMU data
imuBag = select(rosBagAll,'Topic','/imu/data');
imurawBag = select(rosBagAll,'Topic','/imu/data_raw');
cmdvelBag = select(rosBagAll,'Topic','/jackal_velocity_controller/cmd_vel');
odomfiltBag = select(rosBagAll,'Topic','/odometry/filtered');
toc
disp('Topics are selected');
imuData = readMessages(imuBag);
imurawData = readMessages(imurawBag);
toc
disp('Read IMU Data');
odomfiltData = readMessages(odomfiltBag);
toc
disp('Read odom Messages');
```

Appendix D. MATLAB CODE - IMU data from the iPhone 7 and saving in ". mat" file

```
clear
close all
if(exist('m','var') == 1)
    discardlogs(m);
else
    connector on stevens123
    m = mobiledev;
end

disp('Connect your MATLAB mobile app to the IP address given above')
disp(' ')
disp('Enable sensors and start recording')
disp(' ')
input('Pess return key when finish recording data on mobile dev');

if(m.AccelerationSensorEnabled)
    m.AccelerationSensorEnabled = 0;
    [a,t] = accellog(m);
end
if(m.AngularVelocitySensorEnabled)
    m.AngularVelocitySensorEnabled = 0;
    [av, tav] = angvellog(m);
end

if(m.MagneticSensorEnabled)
    m.MagneticSensorEnabled = 0;
    [mag,tmag] = magfieldlog(m);
end
if(m.OrientationSensorEnabled)
    m.OrientationSensorEnabled = 0;
    [o, to] = orientlog(m);
end
if(m.PositionSensorEnabled)
    m.PositionSensorEnabled = 0;
    [lat, lon, tpos, spd] = poslog(m);
end

filename = ['ADSLab_', char(datetime('now','Format','MMdd_HHmm'))];
save(filename);
```

Appendix E. MATLAB CODE – Algorithm to predict position

```
%% Section to Variables Initialization
close all;
clear

%To get data in m/s^2
gMultiplier = 0.53;
%Initialzing weight factor for complimentory filter for Angular velocity
weightGyro = 5;

%% Section for MPU 9255 data
% Make sure to uncomment the following line or run it once to get data
%from the raspberry pi or have Untitled variable containing data in workspace

MPU9255_data
aRaw0 = [Untitled.Accelerometer_x,Untitled.Accelerometer_y,Untitled.Accelerometer_z];
avRaw0 = [Untitled.Gyroscope_x,Untitled.Gyroscope_y,Untitled.Gyroscope_z];

[lenA,~] = size(avRaw0);
[lenAv,~] = size(aRaw0);
len = min(lenA,lenAv);
t = Untitled.Time(1:len);

%To get the same length of data for both accelerometer and gyroscope
aRaw0 = aRaw0(1:len,:);
avRaw0 = avRaw0(1:len,:);
t = t(1:len);

timeInS = zeros(len,1,'double');
%Converting absolute time to time series in seconds for MPU9255
for i = 2:len
    if(exist('a','var') == 1)
        timeInS(i) = timeInS(i-1) + (t(i) - t(i-1));
    else
        timeInS(i) = timeInS(i-1) + seconds(t(i) - t(i-1));
    end
end

%Thershold for zero velocity on Angular velocity
avRawSumThresh = 0.34;

%flag to know the device type
selection_flag = 1

%% Section for the Jackal Data
% Make sure to uncomment the following line to run to get data
%from the Jackal Robot or  have imuData in workspace

%{
Jackal_Data
[len,~] = size(imuData);
aRaw0 = zeros(len,3,'double');
avRaw0 = zeros(len,3,'double');
t = zeros(len,1,'double');

for i = 1:len
    aRaw0(i,:) = [imuData{i}.LinearAcceleration.X,...
        imuData{i}.LinearAcceleration.Y, imuData{i}.LinearAcceleration.Z];
    avRaw0(i,:) = [imuData{i}.AngularVelocity.X,...
        imuData{i}.AngularVelocity.Y, imuData{i}.AngularVelocity.Z];
    t(i) = imuData{i}.Header.Stamp.Sec +...
        imuData{i}.Header.Stamp.Nsec/1e09;
end

t = t(1:len);
```

```
%Converting absolute time to time series in seconds for the Jackal Robot
timeInS = zeros(len,1,'double');
for i = 2:len
    timeInS(i) = timeInS(i-1) + (t(i) - t(i-1));
end

%Thershold for zero velocity on Angular velocity
avRawSumThresh = 0.34;

%flag to know the device type
selection_flag = 2
%}
%% Section for the iPhone 7 Data

%{

load('Lib311_0912_1716.mat')

aRaw0 = a;
aRawNormal = normr(a);
avRaw0 = rad2deg(av);

[lenA,~] = size(avRaw0);
[lenAv,~] = size(aRaw0);
len = min(lenA,lenAv);

%To get the same length of data for both accelerometer and gyroscope
aRaw0 = aRaw0(1:len,:);
avRaw0 = avRaw0(1:len,:);
t = t(1:len);

timeInS = zeros(len,1,'double');
for i = 2:len
    if(exist('a','var') == 1)
        timeInS(i) = timeInS(i-1) + (t(i) - t(i-1));
    else
        timeInS(i) = timeInS(i-1) + seconds(t(i) - t(i-1));
    end
end

%Thershold for zero velocity on Angular velocity for iPhone 7
avRawSumThresh = 0.5;

%flag to know the device type
selection_flag = 3
%}
%% Filtering Section

avRaw = avRaw0;
aRaw1 = aRaw0;
aRaw = aRaw0;

%Plotting Raw data (Accelerometer) without any changes to data
figure (1);
subplot(3,2,1);
if(selection_flag == 2)
        plot(timeInS,aRaw0/9.8);
    else
        plot(timeInS,aRaw0);
end
title('Raw Acceleration');
legend('x','y','z');
xlabel('Time (sec)')
ylabel("Acceleration(g's)");
ylim([-2 2]);

figure (1);
```

```
subplot(3,2,2);
plot(timeInS,avRaw0);
title('Raw Angular Velocity');
legend('x','y','z');
xlabel('Time (sec)')
ylabel("Angular Velocity(rad/s)");
ylim([-50 50]);

% Loop for filtering all axis of raw data
for i = 1:3
    % De-noise noisy signal using minimax threshold with
    % a multiple level estimation of noise standard deviation.
    avRaw(:,i) = wden(avRaw0(:,i),'modwtsqtwolog','s','mln',8,'sym4');

    figure (1);
    subplot(3,2,4);
    plot(timeInS,avRaw);
    title('De-noised Angular Velocity');
    legend('x','y','z');
    xlabel('Time (sec)')
    ylabel("Angular Velocity(rad/s)");
    ylim([-50 50]);

    order = 6; % 6th Order Filter
    [b1,a1] = butter(order, 0.01, 'low');
    aRaw(:,i) = filtfilt(b1 ,a1 , aRaw1(:,i));

    figure (1);
    subplot(3,2,3);
    if(selection_flag == 2)
        plot(timeInS,aRaw/9.8);
    else
        plot(timeInS,aRaw);
    end
    title('Low Pass Butterworth Filter (Acceleration)');
    legend('x','y','z');
    xlabel('Time (sec)')
    ylabel("Acceleration(g's)");
    ylim([-2 2]);

    aRaw1(:,i) = medfilt1(aRaw1(:,i),100);
    aRaw1(1,i) = aRaw0(1,i);

    figure (1);
    subplot(3,2,5);
    if(selection_flag == 2)
        plot(timeInS,aRaw1/9.8);
    else
        plot(timeInS,aRaw1);
    end
    title('Median Filter (Acceleration)');
    legend('x','y','z');
    xlabel('Time (sec)')
    ylabel("Acceleration(g's)");
    ylim([-2 2]);

    avRaw(:,i) = avRaw(:,i) - mean(avRaw(1:300,i));
    avRaw(:,i) = medfilt1(avRaw(:,i),100);

    figure (1);
    subplot(3,2,6);
    plot(timeInS,avRaw);
    title('Median Filter (Angular Velocity)');
    legend('x','y','z');
    xlabel('Time (sec)')
    ylabel("Angular Velocity(rad/s)");
    ylim([-50 50]);
```

```
end
%% Orientation, Gravity Vector, and Dynamic accleration Calculation
gSum = zeros(1,'double');
for i = 1:300
    gSum = gSum + norm(aRaw0(i,:));
end
gMean = gSum / 300

aEst = zeros(len,3,'double');
anglesPrev = [rad2deg(atan2(aRaw(1,1), aRaw(1,3))),...
    rad2deg(atan2(aRaw(1,2), aRaw(1,3)))];
aEst(1,:) = aRaw(1,:);


for i = 2:len
    if(i < 25)
        avPreviousRaw = avRaw(i-1,:);
    else
        avPreviousRaw = mean(avRaw(i-20:i-1,:));
    end

    if(selection_flag ~= 1)
        timePeriod = (t(i) - t(i-1));
    else
        timePeriod = seconds(t(i) - t(i-1));
    end

    [aEst(i,:),anglesPrev] = findEstimate(avRaw(i,:), aEst(i-1,:),...
        avPreviousRaw, anglesPrev, gMean, timePeriod);
end

aAdjusted = zeros(len,3,'double');
weightGyro2 = zeros(len,1,'double');

%Thershold on gyroweightage according to anuglar velocity
for i = 1:len
    if(selection_flag == 2)
        weightGyro2(i) = weightGyro;
    else
        if(sum(abs(avRaw(i,:))) < avRawSumThresh)
            weightGyro2(i) = 0;
        else
            weightGyro2(i) = weightGyro;
        end
    end
    aAdjusted(i,:) = (aRaw(i,:) + aEst(i,:) * weightGyro2(i)) / (1 + weightGyro2(i));
end

figure (2);
subplot(3,2,1);
if(selection_flag == 2)
    plot(timeInS,aAdjusted/9.8);
else
    plot(timeInS,aAdjusted);
end
title('Acc Gyro Weight Adjusted');
legend('x','y','z');
ylim([-0.5 1.5]);
ylabel("Unit Vector (g's)")
xlabel('Time (sec)')

subplot(3,2,2);
if(selection_flag == 2)
    plot(timeInS,aEst/9.8);
else
    plot(timeInS,aEst);
end
```

```
title('Gyro Estimated DCs');
ylim([-0.5 1.5]);
legend('x','y','z');
ylabel("Unit Vector (g's)")
xlabel('Time (sec)')

gSph = zeros(len,3,'double');
gSphDegree = zeros(len,3,'double');
[gSph(:,1),gSph(:,2),gSph(:,3)] = cart2sph(aAdjusted(:,1),aAdjusted(:,2),aAdjusted(:,3));
%Radians to Degrees
gSphDegree(:,1:2) = rad2deg(gSph(:,1:2));

subplot(3,2,3);
plot(timeInS,gSphDegree(:,1:2));
title('Theta Phi of DCs');
legend('Azimuth Angle','Polar Angle');
xlabel('Time (sec)')
ylabel({"Azimuth and"; "Polar Angles"; "(Degree's)"});

gVector = zeros(len,3,'double');
gSph(:,3) = gMean;

%Gravity vector
[gVector(:,1),gVector(:,2),gVector(:,3)] = sph2cart(gSph(:,1),gSph(:,2),gSph(:,3));

%Calculating Dynamic motion from filtered acceleration
aMotion = aRaw - gVector;
anglesDC = rad2deg(abs(acos(aEst)));

subplot(3,2,4);
if(selection_flag == 2)
    plot(timeInS,gVector/9.8);
else
    plot(timeInS,gVector);
end

title('Gravity Vector Cartesian');
ylim([-0.5 1.5]);
legend('x','y','z');
xlabel('Time (sec)')
ylabel("Acceleration(g's)");

subplot(3,2,5);
plot(timeInS,anglesDC);
title('Angles of DCs with each axis');
legend('x','y','z');
xlabel('Time (sec)')
ylabel({"Euler Angles ";"(Degree's)"});

subplot(3,2,6);
if(selection_flag == 2)
    plot(timeInS,aMotion/9.8);
else
    plot(timeInS,aMotion);
end
title('Dynamic Motion');
legend('x','y','z');
xlabel('Time (sec)')
ylabel("Acceleration(g's)");

DisplacementmagNoG = findDisplacement(aMotion, weightGyro2, timeInS,gMultiplier);

if(selection_flag == 2)
        [lenPos,~] = size(odomfiltData);
    pos1 = [odomfiltData{1}.Pose.Pose.Position.X,...
        odomfiltData{1}.Pose.Pose.Position.Y,odomfiltData{1}.Pose.Pose.Position.Z];
    pos2 = [odomfiltData{lenPos}.Pose.Pose.Position.X,...
```

```
odomfiltData{lenPos}.Pose.Pose.Position.Y,odomfiltData{lenPos}.Pose.Pose.Position.Z];
    odomDistance = norm([pos1;pos2])
end

%Function to estimatet the Orientation and Gravity vector
function [aEst,angles] = findEstimate(avCurrentRaw, aPreviousEst,...
        avPreviousRaw, anglesPrev, gMean, timePeriod)
    aEst = zeros(1,3,'double');
    avXZAvg = mean([avCurrentRaw(2),avPreviousRaw(2)]);
    angleXZcurr = anglesPrev(1) + avXZAvg * timePeriod;
    avYZAvg = mean([avCurrentRaw(1),avPreviousRaw(1)]);
    angleYZcurr = anglesPrev(2) + avYZAvg * timePeriod;
    angles = [angleXZcurr,angleYZcurr];

    aEst(1) = sind(angleXZcurr) / sqrt(1 + (cosd(angleXZcurr)^2)*(tand(angleYZcurr)^2));
    aEst(2) = sind(angleYZcurr) / sqrt(1 + (cosd(angleYZcurr)^2)*(tand(angleXZcurr)^2));
    aEst(3) =  sign(aPreviousEst(3)) * sqrt(1 - aEst(1)^2 - aEst(2)^2);
end


%% Section for Velocity and displacement profile

%Function to calculate the velocity and position profiles
function DisplacementmagNoG = findDisplacement(aMotion, weightGyro2, t,gMultiplier)
    magNoG = aMotion;
    time = t;
    [dataSize,~] = size(aMotion);
    for i = 1:3
        magNoG(:,i) = wden(aMotion(:,i),'modwtsqtwolog','s','mln',8,'sym4');
        magNoG(:,i) = magNoG(:,i) - mean(magNoG(1:500,i));

        thresh = max(abs(magNoG(1:500,i)));
        for j = 1:floor(dataSize/10)
            startAcc = ((j-1)*10) + 1;
            endAcc = j*10;
            if(max(abs(magNoG(startAcc:endAcc,i))) < thresh)
                magNoG(startAcc:endAcc,i) = 0;
            end
        end
        startAcc = (j*10) + 1;
        endAcc = dataSize;
        if(max(abs(magNoG(startAcc:endAcc,i))) < thresh)
            magNoG(startAcc:endAcc,i) = 0;
        end
    end

    magNoG1 = magNoG * gMultiplier;

    figure (3);
    subplot(2,2,1);
    plot(time,aMotion);
    title('Dynamic Acceleration');
    xlabel('Time (sec)');
    ylabel('Acceleration (m/s^2)');
    legend('x','y','z');
    subplot(2,2,3);
    plot(time,magNoG1);
    title('Filtered Dynamic Acceleration');
    xlabel('Time (sec)');
    ylabel('Acceleration (m/s^2)');
    legend('x','y','z');

    % First Integration (Acceleration - Veloicty)
    velmagNoG = cumtrapz(time,magNoG);
    for i = 1:dataSize
        if(weightGyro2(i) == 0)
```

```
            velmagNoG(i,:) = [0,0,0];
        end
    end

    velmagNoG1 = velmagNoG * gMultiplier;

    subplot(2,2,2);
    plot(time,velmagNoG1);
    title('Velocity');
    xlabel('Time (sec)');
    ylabel('Velocity (m/s)');
    legend('x','y','z');

    % Second Integration (Velocity - Displacement)
    DisplacementmagNoG=cumtrapz(time, velmagNoG);
    DisplacementmagNoG1 = DisplacementmagNoG * gMultiplier;

    subplot(2,2,4);
    plot(time,DisplacementmagNoG1);
    title('Displacement');
    xlabel('Time (sec)')
    ylabel('Displacement (m)');
    legend('x','y','z');

    TotalDisplacement = norm(DisplacementmagNoG(dataSize,:));
    disp('Total Displcement(m) = ');
    disp(TotalDisplacement * gMultiplier);

    d = hypot(diff(DisplacementmagNoG(:,1)), diff(DisplacementmagNoG(:,2)));
    TotalDistance = sum(d);
    disp('Total Distance(m) = ');
    disp(TotalDistance * gMultiplier);

    %Plot for Movement Pattern on Cartesian plane
    figure(4);
    plot(DisplacementmagNoG1(:,1),DisplacementmagNoG1(:,2));
    hold on

plot(DisplacementmagNoG1(1,1),DisplacementmagNoG1(1,2),'o','MarkerSize',10,'MarkerFaceCol
or','g')
    hold on

plot(DisplacementmagNoG1(length(DisplacementmagNoG1(:,1)),1),DisplacementmagNoG1(length(D
isplacementmagNoG1(:,2)),2),'o','MarkerSize',10,'MarkerFaceColor','b')
    title('2D Movement');
    xlabel('X-Axis (m)')
    ylabel("Y-Axis (m)");
    legend('Movement Pattern');
    grid on
end
```

References

[1] Aleksandr Mikov, “The multi-mode inertial tracking system for unconstrained indoor positioning,” IEEE 3rd International Symposium on Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS), Offenburg, Germany, pp. 36-43, 2016.

[2] Matthias Gietzelt, Klaus H. Wolf, Michael Marschollek, and Reinhold Haux, “Automatic self-calibration of body worn triaxial-accelerometers for application in healthcare,” IEEE second International Conference on Pervasive Computing Technologies for Healthcare 2008 (PervasiveHealth 2008), pp. 177-180.

[3] Ahmed Wahdan, Jacques Georgy, Walid F. Abdel, and Aboelmagd Noureldi, “Magnetometer calibration for portable navigation devices in vehicles using a fast and autonomous technique,” *IEEE Transaction of Intelligent Transportation Systems*, Vol. 15, No. 5, pp. 2347-2352, October 2014.

[4] E.P. Herrera, R. Quirs, and H. Kaufmann, “Analysis of Kalman approach for a pedestrian positioning system in indoor environments,” European Conference on Parallel and Distributed Computing, Rennes, France, pp. 931-940, August 2007.

[5] I. Belhajem, Y. Ben Maissa and A. Tamtaoui, "A robust low-cost approach for real time car positioning in a smart city using Extended Kalman Filter and evolutionary machine learning," 4th IEEE International Colloquium on Information Science and Technology (CiSt), Tangier, Tangier, Morocco, pp. 806-811, 2016.

[6] P. Bahl and VN. Padmanabhan, “RADAR: An In-Building RF-based User Location and Tracking System,” IEEE INFOCOM, Tel Aviv, Israel, pp. 775-784, March 2000.

[7] “MPU-9250 Datasheet,” InvenSense Inc, 2016. [Online]. Available: http://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf. [Accessed: 01- Feb- 2017]

[8] “Raspberry Pi 3 Model B - Raspberry Pi,” Raspberry Pi, 2017. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed: 01- Jan- 2017]

[9] “I2C Info – I2C Bus, Interface and Protocol,” I2C Info – I2C Bus, Interface and Protocol, 2017. [Online]. Available: http://i2c.info/. [Accessed: 10- Jan- 2017]

[10] “SSH Protocol – Secure Remote Login and File Transfer | SSH.COM,” Ssh.com, 2017. [Online]. Available: https://www.ssh.com/ssh/protocol/. [Accessed: 02- Feb- 2017]

[11] “MATLAB – MathWorks,” Mathworks.com, 2017. [Online]. Available: https://www.mathworks.com/products/matlab.html. [Accessed: 01- Jan- 2017]

[12] “Fritzing,” Fritzing.org, 2017. [Online]. Available: http://fritzing.org/home/. [Accessed: 01- Feb- 2017]

[13] “Jackal UGV - Small Weatherproof Robot – Clearpath,” Clearpath Robotics, 2017. [Online]. Available: https://www.clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/. [Accessed: 16- Sep- 2017]

[14] “Jackal unmanned ground vehicle,” Generation Robots, 2017. [Online]. Available: https://www.generationrobots.com/en/402144-jackal-unmanned-ground-vehicle.html. [Accessed: 10- Nov- 2017]

[15] “MATLAB Mobile Overview,” Mathworks.com, 2017. [Online]. Available: https://www.mathworks.com/products/matlab-mobile.html. [Accessed: 10- Sep- 2017]

[16] “rosbag - ROS Wiki,” Wiki.ros.org, 2017. [Online]. Available: http://wiki.ros.org/rosbag. [Accessed: 04- Dec- 2017]

[17] “Guide to Comparing Gyro and IMU Technologies – Micro-Electro-Mechanical Systems and Fiber Optic Gyros,” Kvh.com, 2014. [Online]. Available: http://www.kvh.com/ViewAttachment.aspx?guidID={A4A8B05C-D372-41EA-B9C6-D7BCC4F40769}. [Accessed: 01- Oct- 2017]

[18] N. Chandrasekaran, “Altitude Mapping by Using LiDAR-Mounted Jackal Robot,” Stevens Institute of Technology, Hoboken, 2017.