

**ODOMETRY ERROR REDUCTION IN WHEELCHAIR USING MORE
THAN ONE SENSOR**

Daniel Boey Mun Weng

**A project report submitted in partial fulfilment of the
requirements for the award of Bachelor of Engineering
(Hons.) Mechanical Engineering**

**Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman**

January 2019

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : _____

Name : DANIEL BOEY MUN WENG

ID No. : 13015182

Date : 26 APRIL 2019

APPROVAL FOR SUBMISSION

I certify that this project report entitled “**ODOMETRY ERROR REDUCTION IN WHEELCHAIR USING MORE THAN ONE SENSOR**” was prepared by **DANIEL BOEY MUN WENG** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons.) Mechanical Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : _____

Supervisor : Prof. Dr. Goh Sing Yau

Date : 26 APRIL 2019

Signature : _____

Co-Supervisor : Danny Ng Wee Kiat

Date : 26 APRIL 2019

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2019, Daniel Boey Mun Weng. All right reserved.

ABSTRACT

Autonomous wheelchair promises a safer and convenient mobility for disabled and senior citizens. Odometry is to estimate position change over time. It uses data from one or more sensors such as encoder attached to wheel and IMU. Odometry is important for navigation of wheelchair. Odometry via wheel rotary encoder is prone to random error such as wheel slip on slippery or uneven surface, and inaccurate measurement of wheelbase and wheel diameter use to calculate position. Meanwhile, IMU data are noisy and once integrated to obtain position and orientation, their values drift. The IMU comprises of 3 separate sensors: accelerometer which measures acceleration and gyroscope which measures angular velocity and magnetometer which measures direction of magnetic north. The IMU outputs acceleration, angular velocity and magnetic field values based on the orientation of the sensor which is referred to as sensor coordinate system. In order to compute meaningful position of the wheelchair, the sensor coordinate system has to be aligned with the wheelchair coordinate system. Rotation matrix is applied to the IMU data to transform the IMU data. IMU data that are transformed is then filtered to reduce noise. When the sensor is stationary, the output data after the exponential filter still fluctuates between ± 0.01 degree/s. Over time, the integrated reading of the gyro sensor will drift due to the fluctuation. Since the fluctuation is very small, it can be assumed to be zero to reduce drift. Next, the data from encoder, accelerometer and gyroscope are combined together with Kalman filter. Test was performed to obtain position from encoder, IMU and sensor fusion output and the position results were compared to the truth. The resulting fused position reduced error by 76.5%.

TABLE OF CONTENTS

TABLE OF CONTENTS		vi
LIST OF TABLES		viii
LIST OF FIGURES		ix
LIST OF SYMBOLS / ABBREVIATIONS		x
LIST OF APPENDICES		xii
CHAPTER		
1	INTRODUCTION	1
1.1	General Introduction	1
1.2	Importance of the Study	2
1.3	Problem Statement	2
1.4	Aims and Objectives	2
1.5	Scope and Limitation of the Study	3
2	LITERATURE REVIEW	4
2.1	IMU Data Transformation	4
2.1.1	Rotation Matrix	5
2.1.2	Quaternion	8
2.2	Filter	9
2.2.1	Exponential Filter	9
2.2.2	Assume Zero	10
2.3	Sensor Fusion	11
2.3.1	Kalman Filter	11
2.4	Summary	14
3	METHODOLOGY, WORK PLAN & PRELIMINARY TEST	15
3.1	Hardware	15
3.2	Software	16
3.3	Preliminary Test	17

3.4	Summary	17
4	RESULTS	19
4.1	Introduction	22
4.1	Rotation Matrix	22
4.1	Assume Zero Filter	23
4.1	Sensor Fusion	24
4.4	Summary	26
5	PROBLEMS AND RECOMMENDED SOLUTIONS	27
5.1	Problems encountered	27
5.2	Recommended Solutions	27
	REFERENCES	28
	APPENDICES	29

LIST OF TABLES

Table 3.1: Transformed acceleration for various orientation	18
Table 3.2: Filtered gyroscope data over a duration of time while stationary	20
Table 3.3: Angle after 45° (anti-clockwise) rotation at various initial orientation	20
Table 4.1: Transformed acceleration	22
Table 4.2: Transformed angular displacement	23
Table 4.3: Assume zero filter (Accelerometer)	23
Table 4.4: Assume zero filter (Gyroscope)	24
Table 4.5: End position of wheelchair	24
Table 4.6: Error in end position of wheelchair	26

LIST OF FIGURES

Figure 2.1: Wheelchair coordinate frame	5
Figure 2.2: Sensor coordinate frame	7
Figure 2.3: Rotations ϕ , θ , ψ between coordinate frames	7
Figure 2.4: Exponential diminishing weightage	10
Figure 2.5: Normal distribution curve of estimates and measurement	11
Figure 2.6: Kalman filter prediction and update cycle	12
Figure 3.1: Rotary Encoder	15
Figure 3.2: MPU-9250	15
Figure 3.3: ROS visual map	16
Figure 3.4: Simulation with Gazebo	17
Figure 3.5: Arduino Prototype	18
Figure 4.1: Measurement of Wheelchair Position	25
Figure 4.2: Measurement of Wheelchair Position (IMU)	25

LIST OF SYMBOLS / ABBREVIATIONS

IMU	inertial measurement unit
GPS	global positioning system
ROS	robotics operating system
DOF	degree of freedom
LIDAR	laser imaging detection and ranging
3D	three dimensions

LIST OF APPENDICES

APPENDIX A: Tables	29
APPENDIX B: Program	32
APPENDIX C: Log book	
APPENDIX D: Supervisor's Comments on Originality Report	
APPENDIX E: Poster	
APPENDIX F: Presentation Slide	

CHAPTER 1

INTRODUCTION

1.1 General Introduction

Odometry is to estimate position change over time. It is commonly used in robot, smartphone, drone and autonomous driving technologies to estimate their position relative to the starting location. Odometry uses data from one or more sensors such as encoder attached to wheel and IMU. Odometry is important for navigation of robots and autonomous driving technologies. Odometry does not provide absolute position but position estimate relative to a known point, usually the starting point. Odometry is commonly used alongside sensors which provide absolute position such as GPS, LIDAR, camera vision. Sensors used for odometry have higher data rate and are lower cost compared to absolute position sensors.

A rotary encoder which measures rotation in degrees may be attached to the wheel of the wheelchair through a pulley and belt. By knowing the diameter of the wheel, the distance travelled by each wheel can be calculated. Simple trigonometry is then used to calculate the 2D position and orientation of the wheelchair relative to the starting location.

A 9-DOF IMU comprises of 3 different sensors in one module: accelerometer, gyroscope and magnetometer. However, only the accelerometer and gyroscope were used in this project. The accelerometer outputs 3D linear acceleration and can be integrated twice to obtain 3D position. Next, the gyroscope outputs angular velocity and needs to be integrated once to obtain 3D orientation: pitch, roll, yaw.

Odometry is prone to error due to the characteristic of the sensor and also the environment and application of the sensor. For rotary encoder, a slippery surface or objects on the ground may reduce accuracy. Whereas, IMU data are noisy and integrating the raw data to obtain position and orientation will amplify the error. It results in position and orientation drift even though the wheelchair is stationary.

Each sensor provides different types of information about the tracked object position with differing accuracies especially in different conditions (Wilbur, 2017). They can be used together in order to obtain a more accurate odometry. Sensor fusion combines two or more sensors where each sensor has different contribution weightage to the resulting odometry depending on their accuracy.

1.2 Importance of the Study

Sensor fusion is the combination of measurements from multiple sensors to obtain a more accurate measurement. It is applied in many automated processes such as airplane autopilot system, drones and autonomous vehicles. Sensors are prone to inaccuracy in varying conditions. When one sensor has high uncertainty in certain condition, other sensors may compensate for it and still output an accurate measurement.

The position of a subject whether a vehicle, drone or wheelchair is important for navigation. Position acquisition can be differentiated into two types, relative and absolute position. Relative position is synonym to odometry while absolute position is the exact position which is acquired from systems such as GPS, LIDAR, vision. Sensors to obtain absolute position has a lower measurement rate (< 50 Hz) and are costlier compared to odometry. Hence, odometry is often used alongside absolute position in acquiring accurate position.

1.3 Problem Statement

Odometry is based on relative position and the accuracy of the odometry is affected by the sensor's limitation and odometry method in varying conditions. Odometry via wheel rotary encoder is prone to random error such as wheel slip on slippery or uneven surface, and inaccurate measurement of wheelbase and wheel diameter use to calculate position. Meanwhile, IMU data are noisy and once integrated to obtain position and orientation, their values drift. Drifting position values will show that the wheelchair is moving when in actual, it is stationary. However, IMU sensor is not prone to inaccuracy due to varying surface condition unlike motor encoder.

Errors in odometry compound on each other since they are based on relative position, and not absolute position. A small initial error will cause a large error after a duration even with no additional error after the initial error. In conclusion, position acquisition through odometry cannot be done accurately with only one sensor.

1.4 Aims and Objectives

The aim of this project is to reduce odometry error in wheelchair by using more than one sensor. The objective of this project is to:

- a) Compute and process raw data from accelerometer and gyroscope to obtain accurate position and orientation information of the wheelchair.

- b) Apply sensor fusion method to combine data from wheel rotary encoder, accelerometer and gyroscope.

1.5 Scope and Limitation of the Study

The scope of this project is to compute position and orientation information from IMU. Then, sensor fusion is applied to combine data of the IMU and motor encoder. The programming language to be used is C++. ROS will be used to integrate all of the program components together and also as a robot visualisation tool. Arduino is also use to quickly test sensor fusion methods or various filters.

In this project, the data from sensors to be fused together are limited to accelerometer, gyroscope and wheel rotary encoder. The project covers data processing of IMU to obtain position and orientation. However, position and heading derived from wheel rotary encoder was previously done by another student for his final year project. Hence, only derivation of position and orientation from IMU and not wheel rotary encoder will be covered in this project.

CHAPTER 2

LITERATURE REVIEW

2.1 IMU Data Transformation

The IMU comprises of 3 separate sensors: accelerometer which measures acceleration and gyroscope which measures angular velocity and magnetometer which measures direction of magnetic north. These three components in the IMU allows the 3-dimensional (x, y, z-axis) position of a subject to be compute.

The IMU outputs acceleration, angular velocity and magnetic field values based on the orientation of the sensor which is referred to as sensor coordinate system. In order to compute meaningful position of the wheelchair, the sensor coordinate system has to be aligned with the wheelchair coordinate system.

The orientation of the sensor can be aligned with the wheelchair by two methods: physically or by software calibration. The former requires a custom-made precision jig/mount to physically secure the sensor to the wheelchair. Designing and fabricating mount for the sensor is both time consuming and costly. In addition, it is difficult to install the sensor in perfect orientation with the wheelchair.

Therefore, software calibration may be performed so that the sensor can be installed freely in different wheelchairs. An algorithm is used to automatically calibrate and normalise the sensor values in order to have accurate readings. The algorithm will transform the sensor values in the sensor coordinate frame into the wheelchair coordinate system.

There are two methods to transform the coordinate frames: rotation matrix and quaternion.

2.1.1 Rotation Matrix (Euler Angle)

Euler angle is used to represent 3-dimensional orientation of the wheelchair using a combination of three rotations about the x, y, z-axes. In the wheelchair coordinate frame, the x-axis is pointing out towards the front of the wheelchair, y-axis towards the right side and z-axis downwards. Roll (ϕ) is along the x-axis, pitch (θ) is along the y-axis and yaw (ψ) is along z-axis. Figure 2.1 shows the wheelchair coordinate frame.



Figure 2.1: Wheelchair coordinate frame

The acceleration values in wheelchair coordinate frame is computed through the product of rotation matrix, R and acceleration values from the sensor, a_s as shown in the Equation 2.1 below.

$$\begin{bmatrix} a_{x,w} \\ a_{y,w} \\ a_{z,w} \end{bmatrix} = R \cdot \begin{bmatrix} a_{x,s} \\ a_{y,s} \\ a_{z,s} \end{bmatrix} \quad (2.1)$$

where

a_w = acceleration in wheelchair coordinate frame, g

a_s = acceleration in sensor coordinate frame, g

$$R = R_{\phi}^x \cdot R_{\theta}^y \cdot R_{\psi}^z \quad (2.2)$$

$$R_{\phi}^x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \quad (2.3)$$

$$R_{\theta}^y = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad (2.4)$$

$$R_{\psi}^z = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.5)$$

$$R = \begin{bmatrix} c(\theta)c(\psi) & -c(\theta)s(\psi) & -s(\theta) \\ -s(\phi)s(\theta)c(\psi) + c(\phi)s(\psi) & s(\phi)s(\theta)s(\psi) + c(\phi)c(\psi) & -s(\phi)c(\theta) \\ c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi) & -c(\phi)s(\theta)s(\psi) + s(\phi)c(\psi) & c(\phi)c(\theta) \end{bmatrix} \quad (2.6)$$

where

$c = \cos$

$s = \sin$

For the angular velocity, it can be transformed using the transformation matrix as shown in Equation 2.7 below.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} p + q \sin(\phi) \tan(\theta) + r \cos(\phi) \tan(\theta) \\ q \cos(\phi) - r \sin(\phi) \\ q \sin(\phi) / \cos(\theta) + r \cos(\phi) / \cos(\theta) \end{bmatrix} \quad (2.7)$$

where

$p = x$ -axis gyro sensor output

$q = y$ -axis gyro sensor output

$r = z$ -axis gyro sensor output

In the rotation matrix, ϕ , θ , ψ represent the angles of rotation between the sensor coordinate frame and wheelchair coordinate frame.

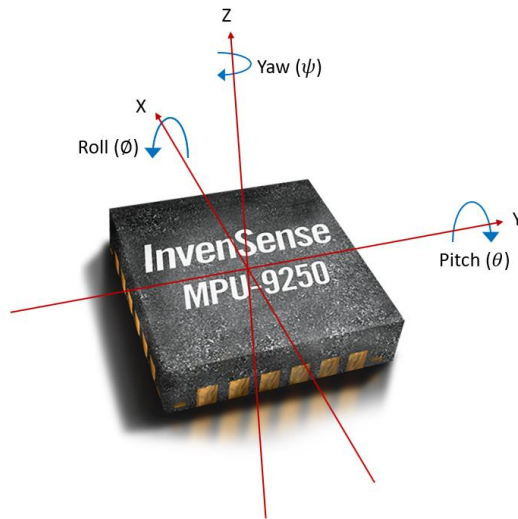


Figure 2.2: Sensor coordinate frame

The angle of rotation between the sensor coordinate frame and wheelchair coordinate frame are calculated by measuring the gravity vector components on each axis of the accelerometer as shown in Figure 2.2 below.

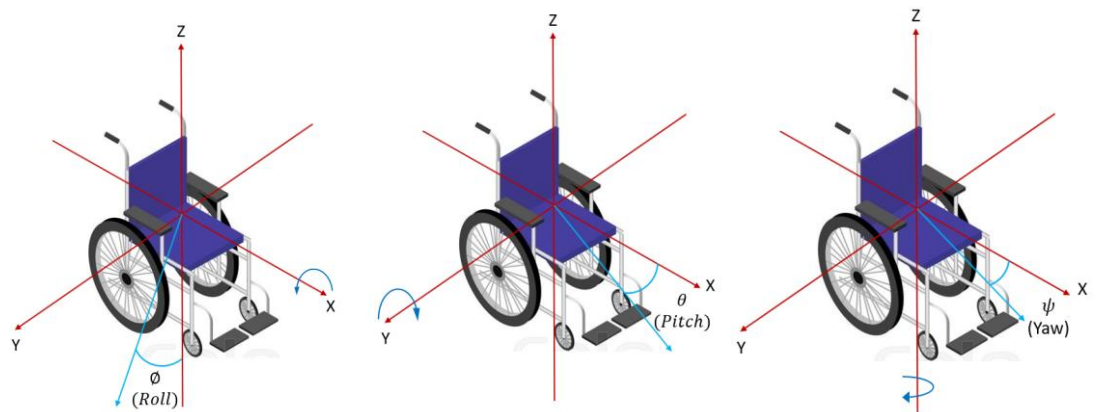


Figure 2.3: Rotations ϕ , θ , ψ between coordinate frames

For vertical orientation angles ϕ and θ , the gravity vector components on each axis of the accelerometer are measured while the wheelchair is stationary and on a flat surface.

$$\phi = \tan^{-1} \left(\frac{g_y}{g_z} \right) \quad (2.8)$$

$$\theta = \tan^{-1} \left(\frac{g_x}{\sqrt{g_y^2 + g_z^2}} \right) \quad (2.9)$$

After the vertical orientation angles ϕ and θ are obtained, the rotation matrixes R_ϕ^x and R_θ^y are used to calculate the new coordinate system where the z-axis of the sensor is aligned with the z-axis of the wheelchair.

$$a_w = R_\phi^x \cdot R_\theta^y \cdot a_s \quad (2.10)$$

The angle ψ is obtained to align the x-axis of the sensor with the wheelchair's x-axis (forward). The acceleration vector in both the x and y-axis are recorded while the wheelchair is moving forward and the angle ψ is compute with the equation below.

$$\psi = \cos^{-1} \left(\frac{a_x}{\sqrt{a_x^2 + a_y^2}} \right) \quad (2.11)$$

The calibration process of obtaining the angles between frames ϕ , θ , ψ is only needed to be performed once. The calibration values can then be stored and recalled.

The disadvantage of using rotational matrix is that it experiences gimbal lock. Gimbal lock occurs when the pitch angle is at $\pm 90^\circ$. At this angle, the yaw axis and roll axis coincide therefore the sensor is unable to track orientation.

However, in our application the wheelchair pitch angle will not come close to $\pm 90^\circ$ unless the wheelchair flips upwards/downwards.

Gimbal lock can be avoided by representing angle in quaternions instead of Euler angles which will be discussed next.

2.1.2 Quaternion

Angle can also be represented as quaternion for 3D transformation. Four quaternion representation are first calculated. Then a matrix is used to calculate the rotation, R. Similar to rotation matrix method, Equation 2.16 will be multiplied by acceleration component to compute the transformed acceleration.

$$q_o = \cos(\alpha/2) \quad (2.12)$$

$$q_1 = \sin(\alpha/2)A_{norm,x} \quad (2.13)$$

$$q_2 = \sin(\alpha/2)A_{norm,y} \quad (2.14)$$

$$q_3 = \sin(\alpha/2)A_{norm,z} \quad (2.15)$$

$$R = \begin{bmatrix} 1 - 2(q_2^2 + q_3^2) & 2(q_1q_2 - q_0q_3) & 2(q_0q_2 + q_1q_3) \\ 2(q_1q_2 + q_0q_3) & 1 - 2(q_1^2 + q_3^2) & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_0q_1 + q_2q_3) & 1 - 2(q_1^2 + q_2^2) \end{bmatrix} \quad (2.16)$$

The advantages of using quaternion is that it isn't affected by gimbal lock and it has fewer equations to compute.

2.2 Filter

The data obtained from IMU is inherently noisy and the values fluctuates close to the true value at high frequency. A filter may be used to obtain a cleaner data with less fluctuations.

2.2.1 Exponential Filter

An exponential filter is used to smooth out time series data. The filter's output, x , is the weighted average of the current data and previous data, with the weighting decreasing exponentially. A higher weightage would be given to the most recent data. As time passes, the weightage of older data decreases exponentially as illustrated in the Figure 2.3 below.

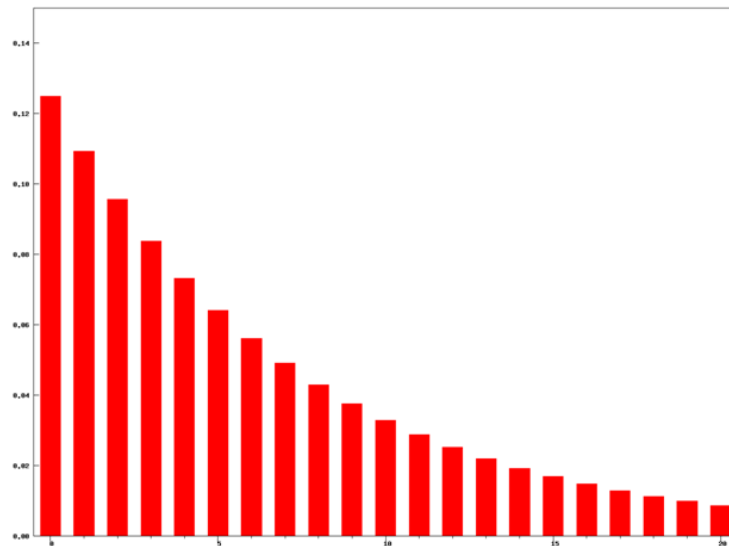


Figure 2.4: Exponential diminishing weightage

The exponential filter equation is as below.

$$x_n = \alpha \cdot y_n + (1 - \alpha) \cdot x_{n-1} \quad (2.17)$$

where

x_n = smoothed value

x_{n-1} = previous smoothed value

y_n = new measurement

α = weightage

2.2.2 Assume Zero

When the sensor is stationary, the output data after the exponential filter still fluctuates between ± 0.01 degree/s. Over time, the integrated reading of the gyro sensor will drift due to the fluctuation. Drifting occur when absolute angle is increasing or decreasing even though the sensor is stationary. Since the fluctuation is very small, it can be assumed to be zero to reduce drift. This assumption may however affect reading during actual rotation even though the value we assume to be zero is small.

2.3 Sensor Fusion

Sensor fusion algorithm is used to combine information from two or more sensors. It is used to obtain a more accurate information when accurate information cannot be obtained from just one sensor. Hence, information from several sensors are used to estimate the current state.

2.3.1 Kalman Filter

Kalman filter is used to predict state and as a smoothing filter. It also can be used to combine information from several sensors to work in unison. The states are represented with a normal distribution. Figure 2.4 shows the normal distribution curve of the predicted state estimate and measurement. Optimal state estimate is the output of the Kalman filter. Besides that, Kalman filter is efficient and requires little memory as it stores little information. The filter only requires a few matrix operations. Its application ranges from space, robotics, navigation, weather forecast to economics.

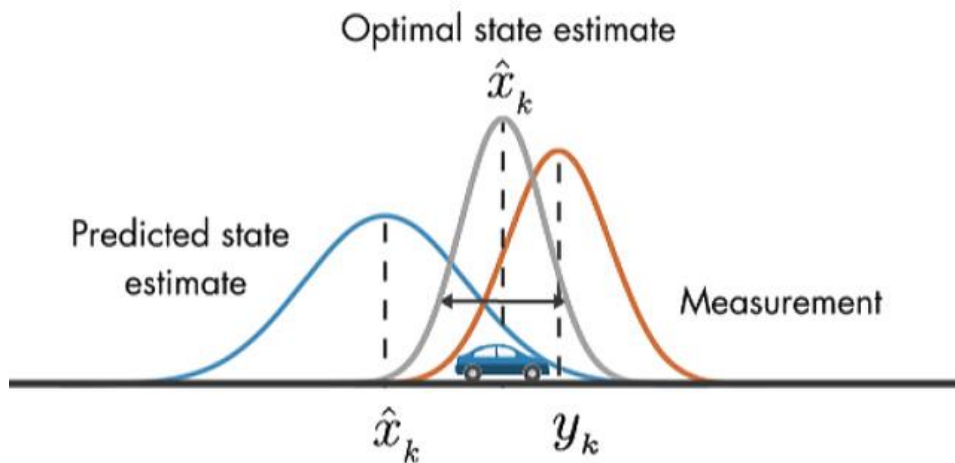


Figure 2.5: normal distribution curve of estimates and measurement

The basic Kalman filter can only be used for linear functions. An iteration of the Kalman filter, Extended Kalman filter which utilise Jacobian function and Taylor series is able to compute non-linear functions.

The equations of Kalman filter can be categorised as prediction and update. Equation 2.18 and Equation 2.19 are to predict the state and error covariance respectively in absent of sensor measurement input. Equation 2.20 is to compute the

Kalman gain which is used in equation 2.21 to update the estimate. Lastly, Equation 2.22 update the error covariance.

Prediction

$$\hat{x}_k = A\hat{x}_{k-1} + Bu_k \quad (2.18)$$

$$P_k = AP_{k-1}A^T + Q \quad (2.19)$$

Update

$$G_k = P_k C^T (C P_k C^T + R)^{-1} \quad (2.20)$$

$$\hat{x}_k = \hat{x}_k + G_k (z_k - C \hat{x}_k) \quad (2.21)$$

$$P_k = (I - G_k C) P_k \quad (2.22)$$

where

\hat{x}_k = current state estimate

A = state transition function

\hat{x}_{k-1} = previous state estimate

B = scale of control signal

u_k = control signal

P_k = prediction error or uncertainty

Q = IMU Sensor noise

G_k = Kalman gain

C = map for prediction to measurement state

R = noise

z_k = current observation (sensor measurement)

The Kalman filter works by repetitively calculating the prediction and update. However, it is not compulsory to run both prediction and update one after the other. Prediction is used whenever sensor measurement is not available. It will estimate the current state position of the wheelchair based on previous state position, \hat{x}_{k-1} and state transition function, A . Whenever sensor measurement is available, the update

equations will be calculated using the current sensor measurement, z_k . Figure 2.5 shows the cycle of the Kalman filter.

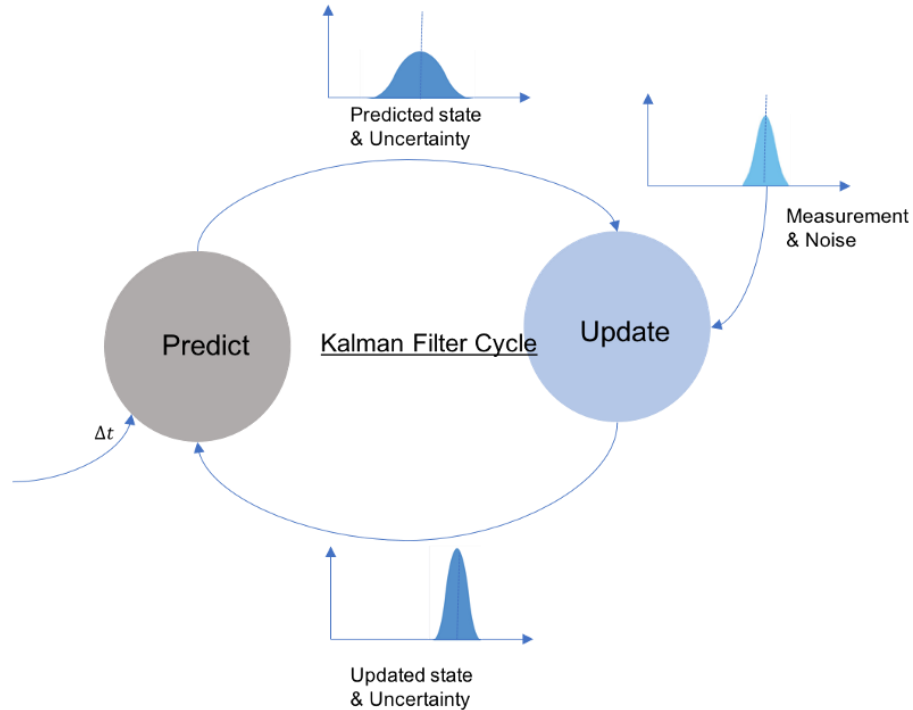


Figure 2.6: Kalman filter prediction and update cycle

The Kalman gain, G_k affects how much weightage is given to the sensor measurement, z_k or previous state estimate, \hat{x}_{k-1} . If the gain is 0, the previous state estimate is the current estimate. If the gain is 1, the current estimate would be the current sensor measurement. The Kalman gain, G_k is obtained from the prediction error, P_k . If prediction error is 0, the Kalman gain, G_k will be 0 resulting in an unchanged current state estimate. Which makes sense, because the state estimate should not be changed if the prediction is accurate.

The prediction error, P_k is calculated recursively from the previous prediction error, P_{k-1} . when $G_k=0$, we have $P_k = P_{k-1}$. So, just as with the state estimation, a zero gain means no update to the prediction error (Levys, 2016). On the other hand, when $G_k = 1$, $P_k = 0$, maximum gain corresponds to zero prediction errors. Hence, the current observation alone is used to update the current state.

In the case of this project, the current state estimate, \hat{x}_k outputs a matrix of position and rotation information $\{x, y, \text{yaw}, \text{pitch}, \text{roll}\}$. However, each sensor used

may only output a certain information, for example $\{x, y\}$ only. Hence, the constant C is used to map the prediction state to the measurement state.

2.4 Summary

IMU data transformation is needed to transform from sensor coordinate frame to wheelchair coordinate frame. The methods are rotation matrix and Quaternion. Filter is used to clean out noisy data and obtain a more accurate position and orientation information from the IMU. Sensor fusion fuses data from two or more sensors. Kalman fusion involves five prediction and update equation which is used to combine data from multiple sensors.

CHAPTER 3

METHODOLOGY, WORK PLAN & PRELIMINARY TEST

3.1 Hardware

The wheelchair used in this study is driven by two DC motors. The axles of the motors are directly connected to front wheels. At the rear, there are two smaller caster wheels which are free to rotate in the z-axis when the wheelchair is steered, similar to a shopping mall trolley. The wheelchair steers by differential power output to the front wheels.

The motorised wheelchair is equipped with rotary encoders to measure rotation of the front wheels. On both front wheels, belt and pulleys translate the rotations of the wheels to the rotary encoder. The rotary encoder brand and model are ESB Electronics Industries, type B 106 23850. Its operating voltage is between 5V to 24V and a current draw of around 120mA. The resolution of the rotary encoder is 500 pulse per rotation at a maximum measurement rate of 100kHz. Figure 3.1 shows the rotary encoder used.



Figure 3.1: Rotary Encoder

IMU sensor used in the wheelchair is an MPU-9250 9-axis device. It includes a gyroscope, accelerometer and magnetometer within a 3×3×1mm package. The MPU-9250 consumes a current of only 9.3μA. The gyroscope may be set to measure 3-axis angular rate at 3 different scale range of ± 250 , ± 500 , ± 1000 and $\pm 2000^\circ/\text{sec}$.

Meanwhile, the accelerometer output can be set to a range of $\pm 2g$, $\pm 4g$, $\pm 8g$ and $\pm 16g$. Figure 3.2 shows the MPU-9250 used.

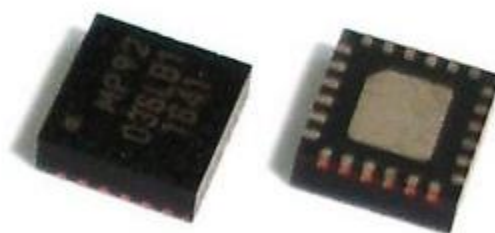


Figure 3.2: MPU-9250

3.2 Software

The programming language that is used is C++. Additionally, Robotics Operating System (ROS) is used as a wrapper for the programs. ROS offers a message passing interface that provides inter-process communication. Program components of a programming project can be linked to work together easily with ROS. ROS represents each program components as nodes and each node are able to publish and subscribe to each other to send and receive data. ROS also features a visual map to view the network of nodes as shown in Figure 3.3.

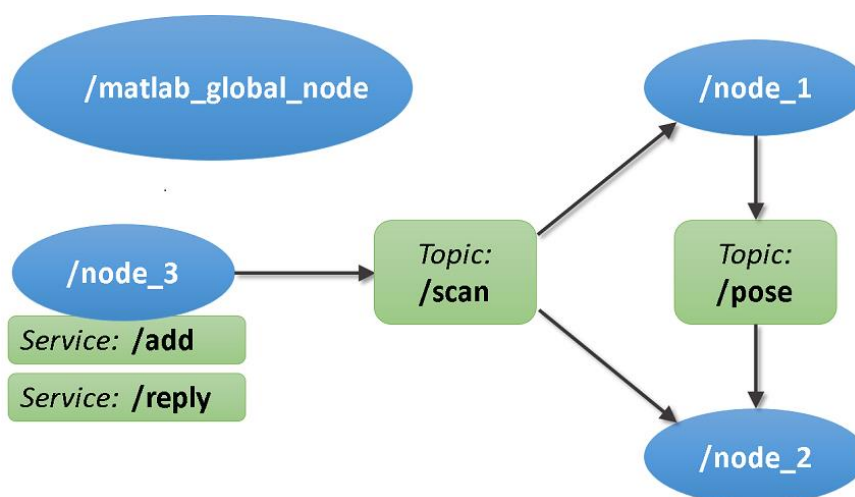


Figure 3.3: ROS visual map

ROS only runs on Linux and MacOS and supports C++ and Python. The IDE used to program the wheelchair is g++. However, it is open source and no license fee is required for commercial usage. Another advantage of ROS is the ability to run simulation of robots through Gazebo. An example of simulation with Gazebo is shown in Figure 3.4.

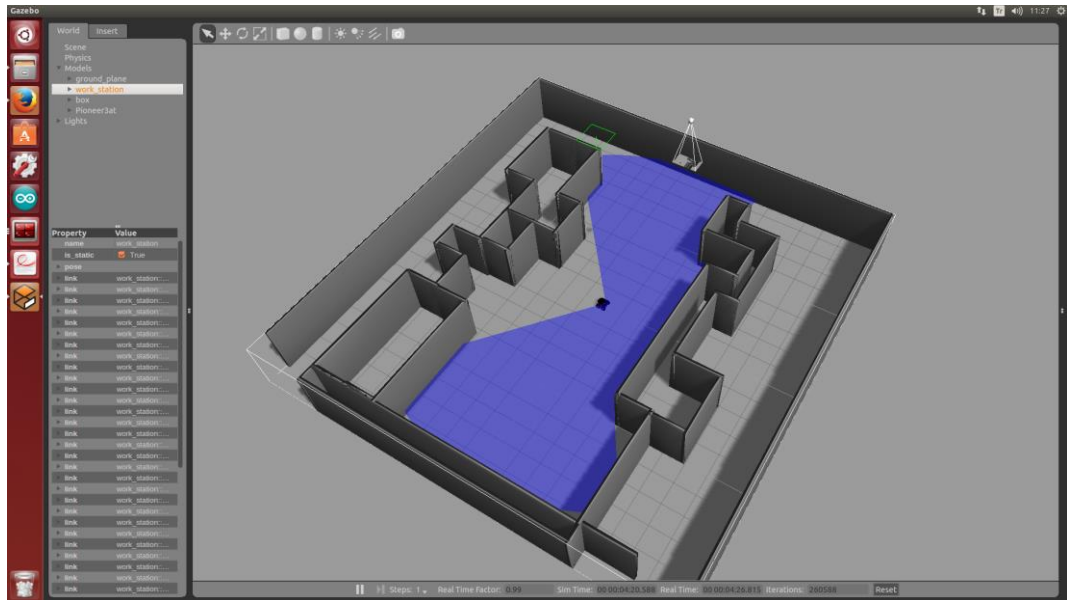


Figure 3.4: Simulation with Gazebo

3.3 Preliminary Test

In order to validate the methods to transform the IMU data and several filters, a prototype was made with an Arduino Uno, MPU6050 6-DOF IMU sensor and OLED display. It is programmed in Arduino's C-based programming language. The rotation matrix method, exponential filter and assume zero was tested.

Figure 3.5 shows the MPU5060 IMU (right) and OLED (left) attached to the prototype shield. The prototype shield was connected to the Arduino Uno below it. The OLED displays accelerometer data at the top row and absolute angle of the gyroscope below.

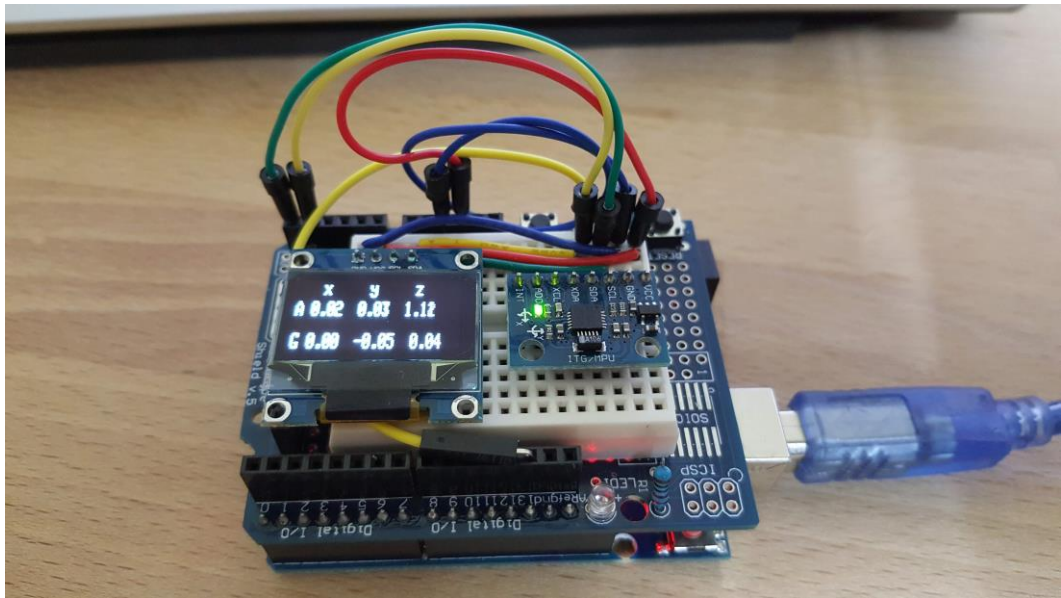


Figure 3.5: Arduino Prototype

3.3.1 Rotation Matrix

The rotation matrix method to transform accelerometer and gyroscope values discussed in chapter 2 was tested. The rotation matrix transformation was tested with various sensor orientation. To calibrate the heading, one side of the sensor is tilted downwards to simulate positive acceleration. The side that is tilted downwards is the new x-axis.

The gravitation acceleration is not removed in the accelerometer reading. Therefore, the transformation method can be proven to be correct when the z-axis has an acceleration of 1.00g while the x-axis and y-axis are 0.00g when orientation is unchanged after calibration.

The acceleration test results for various orientation are tabulated in Table 3.1 below.

Table 3.1: Transformed acceleration for various orientation

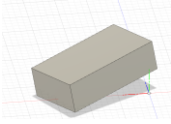
Sensor orientation	x-axis acceleration (g)	y-axis acceleration (g)	z-axis acceleration (g)
	0.19	-0.27	1.06

Table 3.1: (Continued)

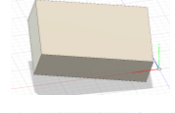
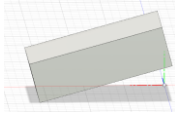
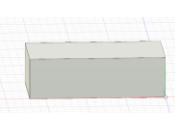
	-0.05	0.27	1.08
	-0.07	-0.30	1.03
	-0.01	0.05	1.12

Table 3.1 shows that the accelerations are transformed but there were errors in the x-axis and y-axis acceleration when orientation of sensor is not horizontal.

The rotation matrix was also tested to transform the gyroscope data. To validate whether the rotation matrix had correctly transformed the gyroscope data, it was needed to be integrated into absolute angle. Hence, it will be discussed in the next sub-topic 3.3.2.

3.3.2 Integrating & Filtering Gyroscope Data

After transforming the gyroscope data, it has to be integrated to obtain absolute angle. When the raw gyro data is integrated, the angle will drift. Drifting occurs when absolute angle is increasing or decreasing even though the sensor is stationary. It is due to noisy raw gyro data. The gyro data which was noisy was cleaned with an exponential filter with weightage, α of 0.6 when the angular velocity is greater than 0.2 degree/s. When the angular velocity is below 0.1 degree/s, a stronger filter is used by decreasing the weightage to 0.1. Two different weightages were used to obtain very clean values (reduce drift) at low angular velocity but have a good response at higher angular velocity. If a low weightage is used even at high angular velocity, the absolute angle calculated would not be accurate.

Even with an exponential filter, the angular velocity data showed a small angular velocity even though the sensor was stationary. Although the angular velocity when stationary was small, it cannot be ignored as integrating it to obtain the absolute value will cause drifting. However, the angular velocity is small enough to assume as zero without affecting reading accuracy during actual rotation. Hence, in the program, any angular velocity below $0.01^\circ/\text{s}$ is considered to be 0.

Table 3.2 shows the transformed and filtered gyroscope data over a duration of time while stationary.

Table 3.2: Filtered gyroscope data over a duration of time while stationary

Rotation axis	0 minute	1 minute	10 minutes	30 minutes
	Absolute angle (degrees, °)			
x-axis	0.00	0.00	0.00	0.00
y-axis	0.00	-0.01	-0.03	-0.06
z-axis	0.00	-0.01	-0.02	0.22

The results in Table 3.2 shows that the absolute angle after filtering the raw data from the gyroscope produces small error which is less than 0.25° .

The transformation of gyro data using rotation matrix discussed in 3.3.1 was verified after integration and filtering. The sensor with various initial orientation was rotated along the x-axis, y-axis, and z-axis by 45° (anti-clockwise) along the z-axis with the help of a protractor and turntable. If there isn't any error during rotation, the x-axis and y-axis angle should remain at 0.00° while the z-axis at 45° . The resulting angle after rotation is tabulated in Table 3.3.

Table 3.3: Angle after 45° (anti-clockwise) rotation at various initial orientation

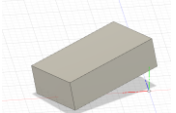
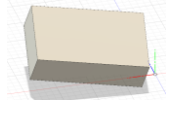
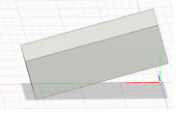
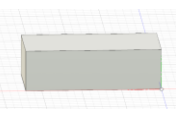
Initial Sensor orientation	x-axis angle (°)	y-axis angle (°)	z-axis angle (°)
	-5.12	-2.07	41.05
	-4.12	-3.63	40.05
	-5.05	-2.35	42.05
	7.05	-1.26	40.31

Table 3.3 shows that both x-axis and y-axis angle are not 0.00° . There were also errors in the z-axis angle for all orientation.

3.4 Summary

The wheelchair in this study are driven by 2 DC motors at the front. It steers through differential power between the 2 wheels. The sensors used for odometry are rotary encoders and MPU-9250. C++ is used as the programming language and ROS as a wrapper to integrate the programs together and add functionality. The quaternion method for IMU data transformation and Kalman fusion will be tested next.

The rotation matrix method is able to transform the acceleration data from sensor coordinate frame to sensor coordinate frame. However, the acceleration data had an error within ± 0.31 g for acceleration.

The gyroscope was transformed with rotation matrix method. Next, the data were filtered with exponential filter and assume zero for angular velocity below $0.01^\circ/\text{s}$. The resulting angle drifted by a maximum of 0.22° after 30 minutes. The rotation test for the gyroscope have a maximum error of 7.05° when rotated 45° .

CHAPTER 4

RESULTS

4.1 Introduction

The programming for the wheelchair was done in sequence to test each component and validate that they are working before continuing with other components. The calibration of the IMU sensor was first tried out. Then the IMU data are filtered before fusing with encoder measurements.

4.2 Rotation Matrix

The rotation matrix discussed in chapter 2 was used to transform accelerometer and gyroscope values. To calibrate the heading, the front of the wheelchair was tilted downward to simulate positive acceleration.

To test the performance of the rotation matrix for accelerometer, the wheelchair is pushed forward for 2m after calibration. The average distance for the first meter is tabulated in Table 4.1 below.

Table 4.1: Transformed acceleration

Test no.	Average x-axis acceleration (m/s ²)	Average y-axis acceleration (m/s ²)	Average z-axis acceleration (m/s ²)
1	0.13840	0.00133	9.80591
2	0.28451	0.00175	9.81421
3	0.17944	0.00124	9.80259

Acceleration components in Table 4.1 shows that acceleration is measured in x-axis and not in y-axis while wheelchair is pushed forward. z-axis acceleration is close to gravitational acceleration.

For gyroscope, the raw angular velocities were also transformed using rotation matrix, Equation 2.7. The wheelchair was rotated 90° clockwise and the angular displacement in all 3-axis were recorded as shown in Table 4.2.

Table 4.2: Transformed angular displacement

Test no.	Average x-axis angular displacement (°)	Average y-axis angular displacement (°)	Average z-axis angular displacement (°)
1	0.0020	0.0038	90.01
2	0.0052	0.0042	89.98
3	0.0019	0.0012	90.03

Table 4.2 shows that angular displacements were close to 0° for x-axis and y-axis. For z-axis, displacement is close to 90°. The gyroscope angular displacement measurements are accurate and correctly transformed.

After the rotation matrix component of the program was tested, it was programmed to run when the overall wheelchair program was first launched. On launching, the program will first calibrate the orientation of IMU in the z-axis with the wheelchair stationary. Then, the program will prompt the user to tilt the front of the wheelchair downwards before calibrating in the x-axis and y-axis. Once completed, the calibration is completed.

4.3 Assume Zero Filter

The accelerometer and gyroscope measurements were recorded when the wheelchair was stationary. The maximum values were used as a threshold. When the IMU measurements were smaller than the threshold, the measurements were considered to be zero. For accelerometers, measurements between -0.005 m/s^2 and 0.005 m/s^2 were assumed to be zero Table 4.3 and Table 4.4 show the comparison between no filter and with filter when the wheelchair is stationary.

Table 4.3: Assume zero filter (Accelerometer)

Test no.	Without filter		With Filter	
	Average x-axis acceleration (m/s²)	Average y-axis acceleration (m/s²)	Average x-axis acceleration (m/s²)	Average y-axis acceleration (m/s²)
1	0.00095	0.00145	0.0000	0.0000

Table 4.3: (Continued)

2	0.00129	0.00190	0.0000	0.0000
3	0.00104	0.00131	0.0000	0.0000

For gyroscope, measurements in z-axis between -0.0005 °/s and 0.0005 °/s were assumed to be zero. Table 4.4 shows unfiltered and filtered measurements for gyroscope in the z-axis.

Table 4.4: Assume zero filter (Gyroscope)

Test no.	Without filter	With Filter
	Average z-axis angular velocity (°/s)	Average z-axis angular velocity (°/s)
1	0.00025	0.00000
2	0.00029	0.00000
3	0.00004	0.00000

4.4 Sensor Fusion

Once the IMU measurements were calibrated and filtered, they were fused with encoder measurements. Kalman filter was used to combine all the measurements together.

The wheelchair was pushed in a rectangular shaped route around concourse area of KB 7-floor of UTAR. The route measured 5.4 m in length and 1.8 m in width. Table 4.5 shows the end position of the wheelchair from encoder measurement, IMU measurement, fused position, and truth.

Table 4.5: End position of wheelchair

Truth (m, m)	Encoder (m, m)	IMU (m, m)	Fused (m, m)
(0, 0)	(0.3515, 0.7276)	(4.7480, -28.0132)	(-0.1381, -0.1306)

Figure 4.1 shows the position of the wheelchair from encoder measurement, IMU measurement, fused position, and truth.

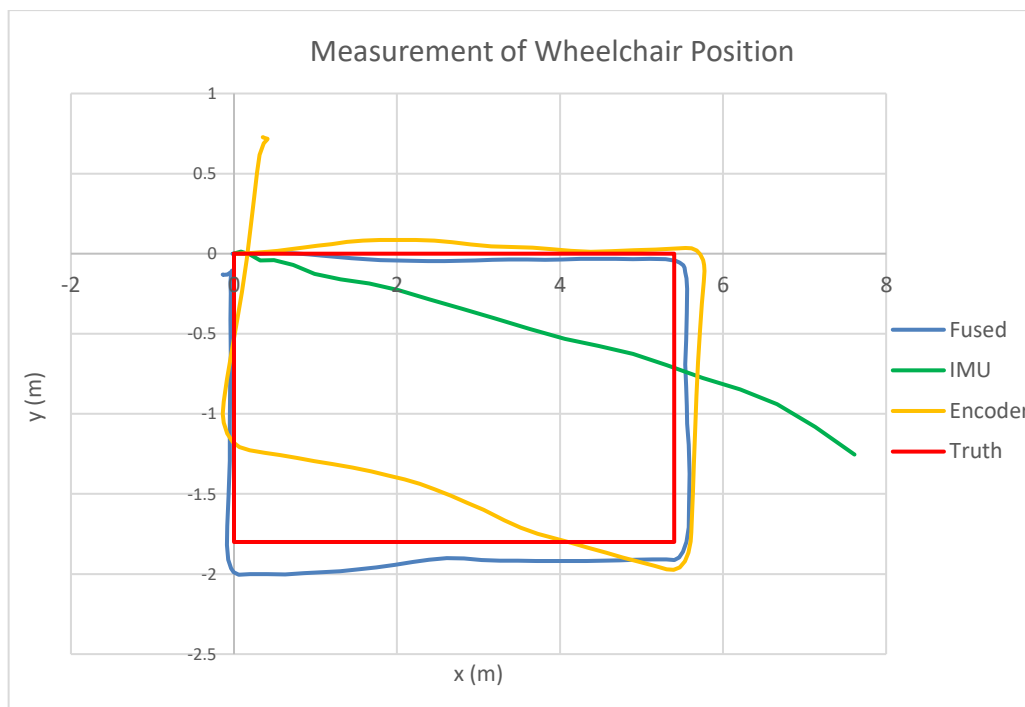


Figure 4.1: Measurement of Wheelchair Position

The position derived from IMU sensor is inaccurate as shown in Figure 4.2. The orientation of the IMU sensor needs to remain the same throughout measurement. When the sensor tilts, the accelerometer will include a fraction of gravitational acceleration depending on how much it tilts in a given axis. Sensor may tilt due to uneven floor or when the sensor mount is insecure.

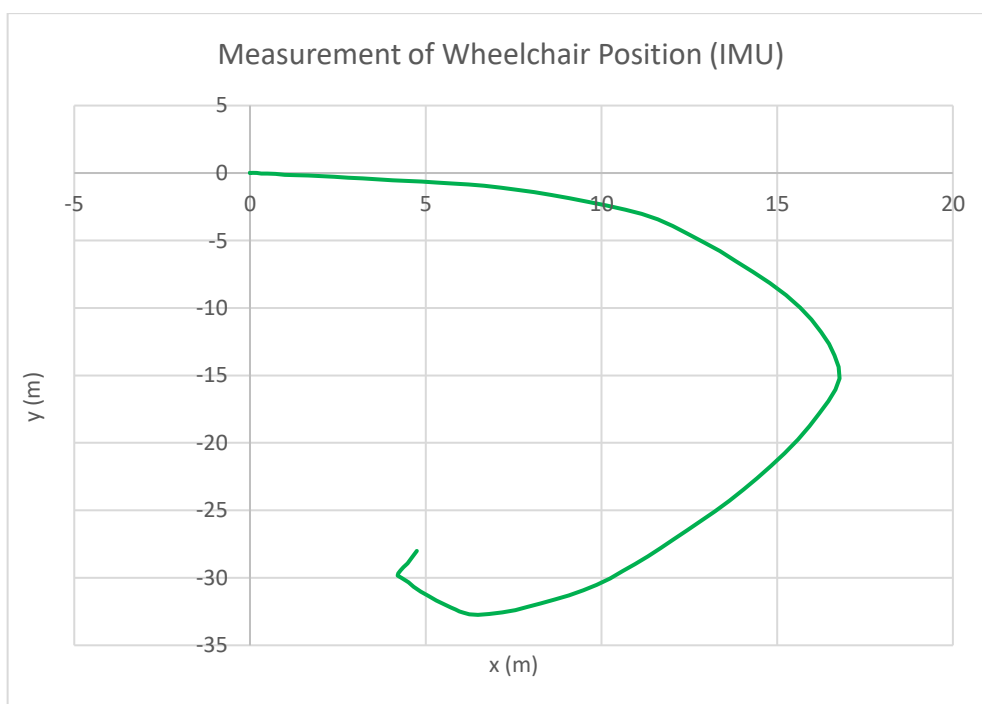


Figure 4.2: Measurement of Wheelchair Position (IMU)

The error in distance between truth and encoder, IMU and fused end position are shown in Table 4.6 below.

Table 4.6: Error in end position of wheelchair

Truth (m)	Encoder (m)	IMU (m)	Fused (m)
0	0.80802	28	0.19005

Figure 4.1 shows that measurement of position from accelerometer has very large error. However, the measurement of angular displacement from the gyroscope is accurate. The encoder produces accurate displacement measurements but when the wheelchair rotates, the encoder angular displacement measurements are inaccurate. Sensor fusion combines sensor measurements that are higher in accuracies together to output an overall more accurate position. From table 4.5 sensor fusion reduced the error of encoder position by 76.5%.

4.5 Summary

The accelerometer and gyroscope data from the IMU sensor were transformed with rotation matrix calculation. The results showed that the measurements from each axis of the accelerometer and gyroscope were now aligned to the wheelchair coordinate frame. The assume zero filter reduced noise of filter while the wheelchair is stationary. The IMU measurements drifts lesser. Sensor fusion combined data from encoder and IMU and produced a more accurate final position of the wheelchair. Error was reduced by 76.5% through sensor fusion.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

IMU raw data was transformed with rotation matrix so that the output measurements of the accelerometer and gyroscope are aligned to the wheelchair coordinate frame. The IMU data was then filtered to reduce drift.

Sensor fusion with Kalman filter was used to combine encoder measurement and IMU data. Results of position and orientation from the encoder were accurate for displacement but not rotation. Meanwhile, IMU measurement produced a more accurate rotation measurement. The resulting fused position reduced error by 76.5%.

5.2 Recommendations

Measurement from IMU may be improved by including compensation for uneven floor. The angular displacement on the x-axis and y-axis may be used for calculation to remove gravitational acceleration when the sensor is tilted. Additionally, more than one IMU sensor may be used and the average measurements may output a more accurate position and orientation. The IMU mount may be redesigned to ensure that it is rigidly secured to the wheelchair.

Position derived through odometry are prone to build up in error over time. Sensors that outputs absolute position such as LIDAR and GPS may be fused with encoder and IMU to reduce error.

REFERENCES

Chih-Hung Wu, Wei-Zhou Hong, and Shing-Tai Pan (2015) 'Performance Evaluation of Extended Kalman Filtering for Obstacle Avoidance of Mobile Robots', Proceedings of the International MultiConference of Engineers and Computer Scientists, 1, pp. 263-267 [Online]. Available at: [Accessed: 25 June 2018].

Bin Lee, H. (2018). Sensor Fusion and Object Tracking using an Extended Kalman Filter Algorithm — Part 1. [online] Medium. Available at: <https://medium.com/@mithi/object-tracking-and-fusing-sensor-measurements-using-the-extended-kalman-filter-algorithm-part-1-f2158ef1e4f0> [Accessed 15 Jul. 2018].

De Souza, W. (2018). Sensor Fusion Algorithms For Autonomous Driving: Part 1 — The Kalman filter and Extended Kalman.... [online] Medium. Available at: <https://medium.com/@wilburdes/sensor-fusion-algorithms-for-autonomous-driving-part-1-the-kalman-filter-and-extended-kalman-a4eab8a833dd> [Accessed 4 Jul. 2018].

Van de Maele, P. (2018). Reading a IMU Without Kalman: The Complementary Filter | Pieter-Jan.com. [online] Pieter-jan.com. Available at: <http://www.pieter-jan.com/node/11> [Accessed 27 Jul. 2018].

Home.wlu.edu. (2018). The Extended Kalman Filter: An Interactive Tutorial. [online] Available at: http://home.wlu.edu/~levys/kalman_tutorial/ [Accessed 19 Jun. 2018].

Caron, F., Duflos, E., Pomorski, D. and Vanheeghe, P. (2006). GPS/IMU data fusion using multisensor Kalman filtering: introduction of contextual aspects. *Information Fusion*, 7(2), pp.221-230.

Alatise, M. and Hancke, G. (2017). Pose Estimation of a Mobile Robot Based on Fusion of IMU Data and Vision Data Using an Extended Kalman Filter. *Sensors*, 17(10), p.2164.

Chrobotics.com. (2018). AN-1006 - Understanding Quaternions. [online] Available at: <http://www.chrobotics.com/docs/AN-1006-UnderstandingQuaternions.pdf> [Accessed 15 Jul. 2018].

MegunoLink. (2018). Three Methods to Filter Noisy Arduino Measurements | MegunoLink. [online] Available at: https://www.megunolink.com/articles/3-methods-filter-noisy-arduino-measurements/?utm_referrer=https://www.google.com/ [Accessed 1 Aug. 2018].

A. Basir, O., Jamali, H., Ben Miners, W. and Toonastra, J. (2018). Method of Correcting the Orientation of a Freely Installed Accelerometer in a Vehicle. [online] Freepatentsonline.com. Available at: <http://www.freepatentsonline.com/20130081442.pdf> [Accessed 1 Aug. 2018].

Lee, J. (2016). A Two-step Kalman/Complementary Filter for Estimation of Vertical Position Using an IMU-Barometer System. *Journal of Sensor Science and Technology*, 25(3), pp.202-207.

APPENDICES

APPENDIX A: Tables

Table 3.1: Transformed acceleration for various orientation

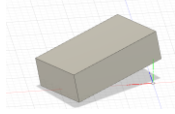
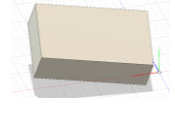
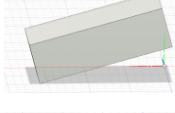

Sensor orientation	x-axis acceleration (g)	y-axis acceleration (g)	z-axis acceleration (g)
	0.19	-0.27	1.06
	-0.05	0.27	1.08
	-0.07	-0.30	1.03
	-0.01	0.05	1.12

Table 3.2: Filtered gyroscope data over a duration of time while stationary

Rotation axis	0 minute	1 minute	10 minutes	30 minutes
	Absolute angle (degrees, °)			
x-axis	0.00	0.00	0.00	0.00
y-axis	0.00	-0.01	-0.03	-0.06
z-axis	0.00	-0.01	-0.02	0.22

Table 3.3: Angle after 45° (anti-clockwise) rotation at various initial orientation

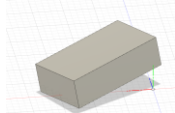
Initial Sensor orientation	x-axis angle (°)	y-axis angle (°)	z-axis angle (°)
	-5.12	-2.07	41.05

Table 3.3 (Continued)

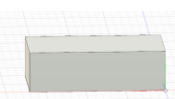
	-4.12	-3.63	40.05
	-5.05	-2.35	42.05
	7.05	-1.26	40.31

Table 4.1: Transformed acceleration

Test no.	Average x-axis acceleration (m/s ²)	Average y-axis acceleration (m/s ²)	Average z-axis acceleration (m/s ²)
1	0.13840	0.00133	9.80591
2	0.28451	0.00175	9.81421
3	0.17944	0.00124	9.80259

Table 4.2: Transformed angular displacement

Test no.	Average x-axis angular displacement (°)	Average y-axis angular displacement (°)	Average z-axis angular displacement (°)
1	0.0020	0.0038	90.01
2	0.0052	0.0042	89.98
3	0.0019	0.0012	90.03

Table 4.3: Assume zero filter (Accelerometer)

Test no.	Without filter		With Filter	
	Average x-axis acceleration (m/s ²)	Average y-axis acceleration (m/s ²)	Average x-axis acceleration (m/s ²)	Average y-axis acceleration (m/s ²)
1	0.00095	0.00145	0.0000	0.0000
2	0.00129	0.00190	0.0000	0.0000
3	0.00104	0.00131	0.0000	0.0000

Table 4.4: Assume zero filter (Gyroscope)

Test no.	Without filter	With Filter
	Average z-axis angular velocity (°/s)	Average z-axis angular velocity (°/s)
1	0.00025	0.00000
2	0.00029	0.00000
3	0.00004	0.00000

Table 4.5: End position of wheelchair

Truth (m, m)	Encoder (m, m)	IMU (m, m)	Fused (m, m)
(0, 0)	(0.3515, 0.7276)	(4.7480, -28.0132)	(-0.1381, -0.1306)

Table 4.6: Error in end position of wheelchair

Truth (m)	Encoder (m)	IMU (m)	Fused (m)
0	0.80802	28	0.19005

APPENDIX A: Program

```

typedef enum publisherStatus
{
    S_INIT,
    S_CALI_XY,
    S_CALI_Z,
    S_READY
} publisherStatus;

int x = 0;

void calibrateXY (library::Driver2Sensor sensor)
{
    ros::Time t = ros::Time::now();
    // Publish IMU Data
    sensor_msgs::Imu imu;

    imu.header.stamp = t;
    imu.header.frame_id = "base_imu";

    imu.linear_acceleration.x = sensor.accelerometer.x - accBias.x;
    imu.linear_acceleration.y = sensor.accelerometer.y - accBias.y;
    imu.linear_acceleration.z = sensor.accelerometer.z - accBias.z;

    imu.angular_velocity.x = sensor.gyroscope.x - gyroBias.x;
    imu.angular_velocity.y = sensor.gyroscope.y - gyroBias.y;
    imu.angular_velocity.z = sensor.gyroscope.z - gyroBias.z;

    //gyro XY calibrate
    r_x = atan2(imu.linear_acceleration.y, imu.linear_acceleration.z);
    accMag_yz =
    pow((pow(imu.linear_acceleration.y,2)+pow(imu.linear_acceleration.z,2)),0.5);
    r_y = atan2(imu.linear_acceleration.z, accMag_yz);

```

```
    status = S_CALI_Z;
}

void calibrateZ (library::Driver2Sensor sensor)
{
    static int count = 0;
    static ros::Time t, prevt;

    prevt = ros::Time::now();
    t = ros::Time::now();

    if(x == 0)
    {
        ROS_INFO("Push Forward");

        while((t - prevt).toSec() < 3) //wait ##s before calibration
        {
            t = ros::Time::now();
        }
        ROS_INFO("Calibrating: Push Forward");
        x = 1;
    }

    //yaw correction angle

    // Publish IMU Data
    sensor_msgs::Imu imu;

    imu.header.stamp = t;
    imu.header.frame_id = "base_imu";
```

```
imu.linear_acceleration.x = sensor.accelerometer.x - accBias.x;
imu.linear_acceleration.y = sensor.accelerometer.y - accBias.y;
imu.linear_acceleration.z = sensor.accelerometer.z - accBias.z;
```

```
imu.angular_velocity.x = sensor.gyroscope.x - gyroBias.x;
imu.angular_velocity.y = sensor.gyroscope.y - gyroBias.y;
imu.angular_velocity.z = sensor.gyroscope.z - gyroBias.z;
```

```
acc_x = (cos(r_y)*imu.linear_acceleration.x)+(-
sin(r_y)*imu.linear_acceleration.z);
acc_y = (-
sin(r_x)*sin(r_y)*imu.linear_acceleration.x)+(cos(r_x)*imu.linear_acceleration.y)+(
-cos(r_y)*sin(r_x)*imu.linear_acceleration.z);
```

```
accC_x += acc_x;
accC_y += acc_y;
```

```
count++;
```

```
if(count == 100)
{
```

```
accC_x /= 100;
accC_y /= 100;
```

```
accMag_yz = sqrt(pow(accC_x,2)+pow(accC_y,2));
r_z = acos(accC_x/accMag_yz);
```

```
ROS_INFO("Done Calibrating");
```

```
prevt = ros::Time::now();
t = ros::Time::now();
```

```

while((t - prevt).toSec() < 5) //wait 5s after calibration
{
    t = ros::Time::now();
}

status = S_READY;
}
}

void publish(library::Driver2Sensor sensor)
{
    imu.linear_acceleration.x = sensor.accelerometer.x - accBias.x;
    imu.linear_acceleration.y = sensor.accelerometer.y - accBias.y;
    imu.linear_acceleration.z = sensor.accelerometer.z - accBias.z;

    imu.angular_velocity.x = sensor.gyroscope.x - gyroBias.x;
    imu.angular_velocity.y = sensor.gyroscope.y - gyroBias.y;
    imu.angular_velocity.z = sensor.gyroscope.z - gyroBias.z;

    imuCorrected.linear_acceleration.x =
(cos(r_z)*cos(r_y)*imu.linear_acceleration.x)+(-
cos(r_y)*sin(r_z)*imu.linear_acceleration.y)+(-sin(r_y)*imu.linear_acceleration.z);
    imuCorrected.linear_acceleration.y = (((-
cos(r_z)*sin(r_x)*sin(r_y))+cos(r_x)*sin(r_z)))*imu.linear_acceleration.x)+(((cos(r
_x)*cos(r_z))+sin(r_x)*sin(r_z)*sin(r_y)))*imu.linear_acceleration.y)+(-
cos(r_y)*sin(r_x)*imu.linear_acceleration.z);
    imuCorrected.angular_velocity.x = imu.angular_velocity.x +
(imu.angular_velocity.y*sin(r_x)*tan(r_y)) +
(imu.angular_velocity.z*cos(r_x)*tan(r_y));
    imuCorrected.angular_velocity.y = (imu.angular_velocity.y*cos(r_x)) -
(imu.angular_velocity.z*sin(r_x));

```

```

imuCorrected.angular_velocity.z =
(imu.angular_velocity.y*sin(r_x)/cos(r_y))+(imu.angular_velocity.z*cos(r_x)/cos(r_
y));

if(imuCorrected.linear_acceleration.x < 0.05 &&
imuCorrected.linear_acceleration.x > -0.05)
{
    imuCorrected.linear_acceleration.x = 0;
}

if(imuCorrected.linear_acceleration.y < 0.05 &&
imuCorrected.linear_acceleration.y > -0.05)
{
    imuCorrected.linear_acceleration.y = 0;
}

if(imuCorrected.angular_velocity.z < 0.000005 &&
imuCorrected.angular_velocity.z > -0.000005)
{
    imuCorrected.angular_velocity.z = 0;
}

static double G = 0.0;
static double Gw = 0.0;

imuCorrected.angular_velocity.x = imuCorrected.angular_velocity.x * 61.8425;
//64.6420
imuCorrected.angular_velocity.y = imuCorrected.angular_velocity.y * 72.0550;
//72.0550
imuCorrected.angular_velocity.z = imuCorrected.angular_velocity.z * 61.8425;

```

```
double dth = (imuCorrected.angular_velocity.z * dt)+ Gw * ((vth * dt)-  
(imuCorrected.angular_velocity.z * dt));
```

```
//initialise variables for Kalman Filter
```

```
static double x_kx;
```

```
static double x_ky;
```

```
static double x_ku;
```

```
static double x_kv;
```

```
static double x_kw;
```

```
static double x_kminus1x;
```

```
static double x_kminus1y;
```

```
static double x_kminus1u;
```

```
static double x_kminus1v;
```

```
static double x_kminus1w;
```

```
static double xdot_kx;
```

```
static double xdot_ky;
```

```
static double xdot_kminus1x;
```

```
static double xdot_kminus1y;
```

```
static double z_kx;
```

```
static double z_ky;
```

```
static double z_kw;
```

```
static double z_kminus1x;
```

```
static double z_kminus1y;
```

```
static double z_kminus1w;
```



```

//calculate new x and y based on steering angle
//multiply "-" to invert direction of y-axis (left = positive)
imuCorrected.linear_acceleration.y = - imuCorrected.linear_acceleration.y;
imuCorrected2.linear_acceleration.x = imuCorrected.linear_acceleration.x *
cos(imuCorrected.angular_velocity.z) - imuCorrected.linear_acceleration.y *
sin(imuCorrected.angular_velocity.z);
imuCorrected2.linear_acceleration.y = imuCorrected.linear_acceleration.y *
cos(imuCorrected.angular_velocity.z) + imuCorrected.linear_acceleration.x *
sin(imuCorrected.angular_velocity.z);

//current position and orientation (IMU)
x_kx = x_kminus1x + xdot_kminus1x*dt +
0.5*imuCorrected2.linear_acceleration.x*pow(dt,2);
x_ky = x_kminus1y + xdot_kminus1y*dt +
0.5*imuCorrected2.linear_acceleration.y*pow(dt,2);
x_ku = x_kminus1u + imuCorrected.angular_velocity.x*dt;
x_kv = x_kminus1v + imuCorrected.angular_velocity.y*dt;
x_kw = x_kminus1w + imuCorrected.angular_velocity.z*dt;

//calculate current velocity (IMU)
xdot_kx = xdot_kminus1x + imuCorrected2.linear_acceleration.x*dt;
xdot_ky = xdot_kminus1y + imuCorrected2.linear_acceleration.y*dt;

//calculate position (encoder)
z_kx = dx + z_kminus1x;
z_ky = dy + z_kminus1y;
z_kw = dth + z_kminus1w;

//fusion
x_kx = x_kx + G*(z_kx - x_kx);
x_ky = x_ky + G*(z_ky - x_ky);
x_kw = x_kw + Gw*(z_kw - x_kw);

```

```

//save current measurement as previous
xdot_kminus1x = xdot_kx;
xdot_kminus1y = xdot_ky;

x_kminus1x = x_kx;
x_kminus1y = x_ky;
x_kminus1u = x_ku;
x_kminus1v = x_kv;
x_kminus1w = x_kw;

z_kminus1x = z_kx;
z_kminus1y = z_ky;
z_kminus1w = z_kw;

static ros::Time t2, prevt2;

t2 = ros::Time::now();

if((t2 - prevt2).toSec() > 0.5) //display every XX seconds
{
    prevt2 = ros::Time::now();
    ROS_INFO( "X = %f, Y = %f, W = %f, imuX = %f, imuY = %f, imuZ = %f ",
x_kx,          x_ky,          x_kw,          imuCorrected.linear_acceleration.x,
imuCorrected.linear_acceleration.x, imuCorrected.linear_acceleration.z);
}

void onData(library::Driver2Sensor sensor)
{

    // Flip

```

```
sensor.encoder.left *= flipEncoderLeft ? -1 : 1;
sensor.encoder.right *= flipEncoderRight ? -1 : 1;
// Scalling encoder back to ppr of encoder used
sensor.encoder.left = sensor.encoder.left * 2000 / ppr;
sensor.encoder.right = sensor.encoder.right * 2000 / ppr;

switch (status)
{
case S_INIT:
    initialize(sensor);
    break;
case S_CALI_XY:
    calibrateZ(sensor);
    break;
case S_READY:
    publish(sensor);
    break;
}
}
```