MARMARA UNIVERSITY
FACULTY OF ENGINEERING

AN EXPERIMENTAL APPROACH TO ESTIMATE THE SHIP CENTER OF GRAVITY BASED ON ACCELERATIONS AND ANGULAR VELOCITIES

İsmail Cemre Öksüz

GRADUATION PROJECT REPORT

Department of Mechanical Engineering

Supervisor

Dr. Ömer Haluk BAYRAKTAR

ISTANBUL, 2023

MARMARA UNIVERSITY

FACULTY OF ENGINEERING

An Experimental Approach To Estimate The Ship Center Of Gravity Based On Accelerations And Angular Velocities

by

İsmail Cemre Öksüz

February, 2023, Istanbul

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING INPARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF BACHELOR OF SCIENCE AT

MARMARA UNIVERSITY

Signature of Author(s)..... İsmail Cemre ÖKSÜZ

Department of Mechanical Engineering

Certified By..... Dr. Haluk BAYRAKTAR

Project Supervisor, Department of Mechanical Engineering

Accepted by..... Prof. Dr. Bülent Ekici

Head of the Department of Mechanical Engineering

ACKNOWLEDGEMENT

I would like to thank my supervisor Dr. Ömer Haluk BAYRAKTAR, for the valuable guidance and advice on preparing this thesis and for giving me moral and material support.

February 2023 **İsmail Cemre ÖKSÜZ**

CONTENTS

ABBREVATIONS

IMU - Inertial Measurement Unit

MEMS - Microelectromechanical systems

I2C - Inter-Integrated Circuit

FFT – Fast Fourier Transform

SYMBOLS

G - Center of gravity

B – Center of buoyancy

M - Metacenter

GM – Metacentric Height

GZ – Righting arm

W – Weight of the ship

θ – Heeling angle

F_B - Buoyancy Force

v – Linear Velocity

s – Linear Position

ω – Angular Velocity

g – Gravitational Acceleration

b – beam length

r = pendulum rod lentgh

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

“If a solid lighter than a fluid be forcibly immersed in it, the solid will be driven upwards by a force equal to the difference between its weight and the weight of the fluid displaced.”[1]

The Archimedes' Law, which has been in existing for 22 centuries and may be one of the most well-known principles of physics, states that a body submerged in a fluid experiences a force equal to the weight of the fluid that has been displaced while effecting in the opposite direction of gravity. This force is known as "Archimedes' force" or "buoyancy force".

When a body is submerged, a pressure differance occurs at different depths of the submerged body due to weight of the overlying fluid. Buoyancy force is caused by the pressure differance between the different depths of the submerged body. This force is always upwards directional and magnitude is proportional to the pressure difference and equivalent to the weight of the imaginary amount of fluid contained within the volume of the submerged body.[6]

In the process of designing a ship, step 1 should be the estimaton of weight of the ship in order the predict buoyancy force required, then to predict hull size and confirm the weight based on hull size.

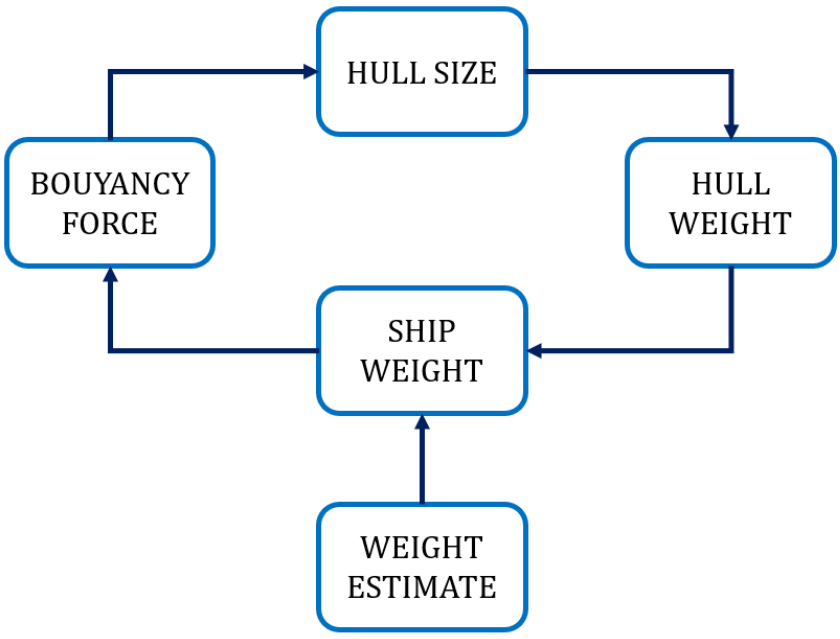

Figure 1: Ship Design Flow

While estimating the ship weight itself, determining the location of center of gravity is a crucial component of the stability calculation in the ship design.

Ship stability is basically depends on the interaction of two forces:

- Weight of the ship
- Buoyancy force

and also design of the ship floating.

When a ship is inclined by an angle θ by an external force such as wind or waves, it is said to be heeled.

When a ship is inclined by an angle θ by forces within the ship such as shifting a weight within the ship in transverse plane, it is said to be listed.

If a floating body experinces an exteral force thus heeling, since there is no change in the mass or distrubition of mass, center of gravity G will remain constant on body frame but since the shape of the area under water or submerged body or say shape of the imaginary water displaced changed. Thus location of the center of buoyancy has been changed.

The verticals lines through the centres of buoyancy at two consecutive angles of heel intersect at a point called the metacentre M. For angles of heel up to about 15° the vertical through the centre of buoyancy may be considered to intersect the centre line at a fixed point called the initial metacentre.[2] This implies that for small angles of heeling, metacenter is the center of rotation.

Vertical distance between metacenter M and center of gravity G is called metacentric height GM. If G is above the M, ship is has negative metacentric height, and if G is below M, ship has postive metacentric height. [2]

If ship has a postive metacentric height, it is called stable equilibrium. The world stable is used because a moment created by weight of the ship called ‘Moment of Statical Stability’ tries to bring the ship to the balanced postion which is called ‘righting’.This moment is caused by the force weight and the distance GZ which is called righting arm. If positive metacentric height is relatively too large then this Moment of Statical Stability is also large and righting occurs fast which may give passengers discomfort so engineers has to be take this into account when designin the ship.

In contrast to postive metacentric height, if ship has a negative metacentric height, it is called unstable equilibrium. In this situation moment created by weight of the ship tries to heel over the ship which is crucial.

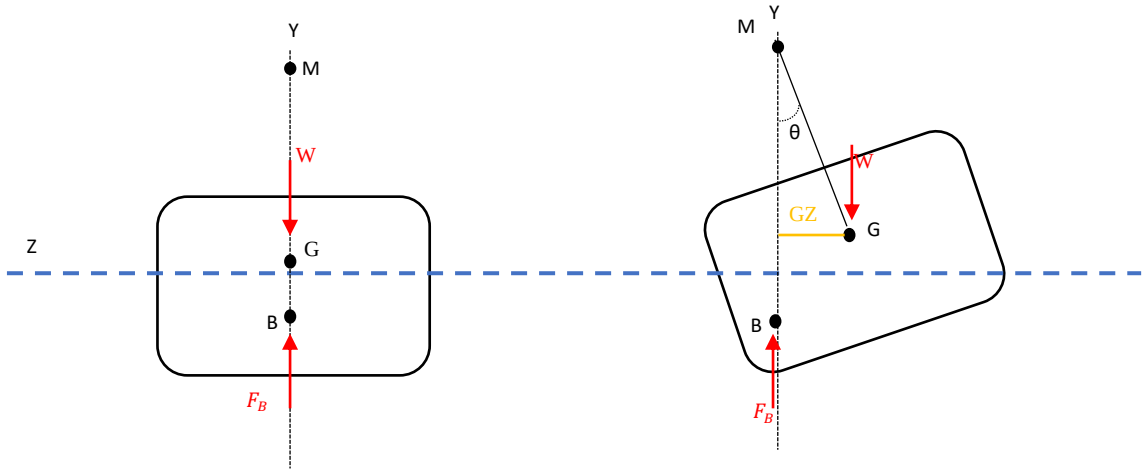

Figure 2: Positive Metacentric Height

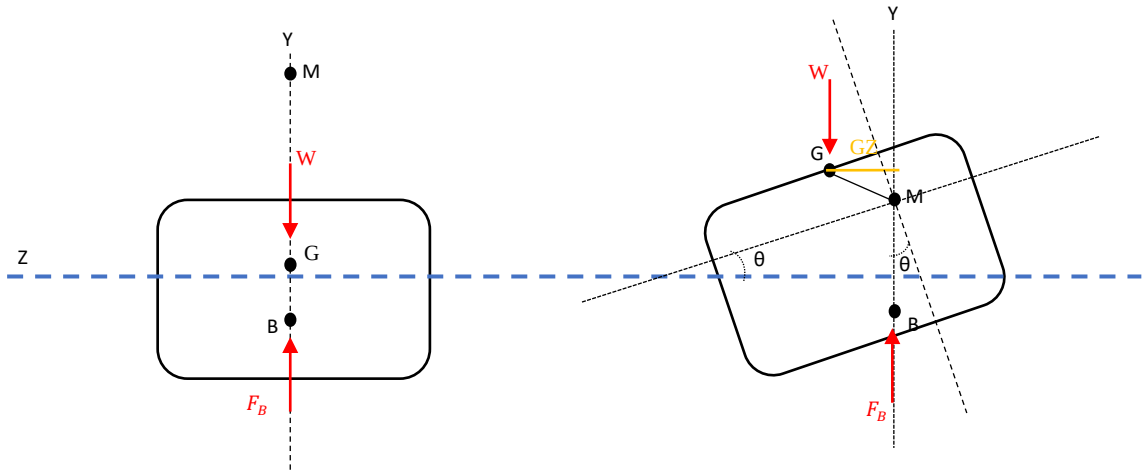

Figure 3:Negative Metacentric Height

It's worth noting that the metacentric height can change during the voyage due to cargo loading, fuel consumption and other factors, so it's important to periodically check the metacentric height of a ship to ensure the safety of the vessel and the crew.

And also determining the location of the G an important aspect of ship design. Even if 3D solid models or other computer aided methods are used to determine G, it is still an estimate and has to be verfied.

An experiment called heeling experiment is carried out to correct this estimated G.[3]

Heeling experiment is a method used to measure the stability of a ship by intentionally tilting it to a specific angle and then measuring the time it takes for the vessel to return to its upright position. This test is typically performed in a controlled environment, such as a shipyard or dock, and is used to evaluate the design and construction of the vessel, as well as its ability to resist capsizing or other forms of instability.

In this Project a method based on the inertial measurements of the ship is proposed to determine the location of the G. This method requires to measure the heeling or say rolling movement of the ship while floating, using inertial measurement sensors (IMU). This sensors has to be able to measure the angular velocity around the x,y and z axises and also linear accelerations in the x,y and z axises.

Measuring linear acceleration and angular velocities provide information about the heeling movement of the vessel. Once movement path is known, rotation center which is metacenter in the ship heeling application can be determined.

Also acceleration measurements provide information about ships orientation which is related to heeling angle.

And from angular velocity measurements we can calculate the metacentric height,

This method claimed to be able to determine location of G independent of ship parameters and is applicable for large type of floating vessels. No input from other sytems or ships command is necessary. Also being cost effective and gives exact results compared to estimations. Further this method can be used to predict ship's movement and avoid dangerous cituations.[1]

2. METHOD AND MATERIALS

2.1 Proposed Method

As we discussed in the introduction part, this method claims to be able to determine location of G without ship parameters.

In order to do that IMU measurements has to be done to extract the movement parameters such as linear acceleration and angular velocity.

With the help of the linear acceleration measurements orientation of the ship can be calculated:[3]

$$\arctan(\theta_x) = \frac{acc\ Z}{acc\ Y} \tag{1}$$

$$\arctan(\theta_y) = \frac{acc\ Z}{acc\ X} \tag{2}$$

Further the postion of the sensors during this heeling can be calculated since acceleration is the rate of change of velocity and velocity is rate of change of position, numerically double integrating the acceleration provides position of the sensor.

$$v = \int_{t_0}^{t_1} a(t)dt + v_0 \tag{3}$$

$$s = \int_{t_0}^{t_1} v(t)dt + s_0 \tag{4}$$

Since up to 15 degrees metacenter is the center of rotation, from initial and final postion of 2 different points on the ship while it is heeling, center of rotation which is metacenter can be geometrically calculated.[3]

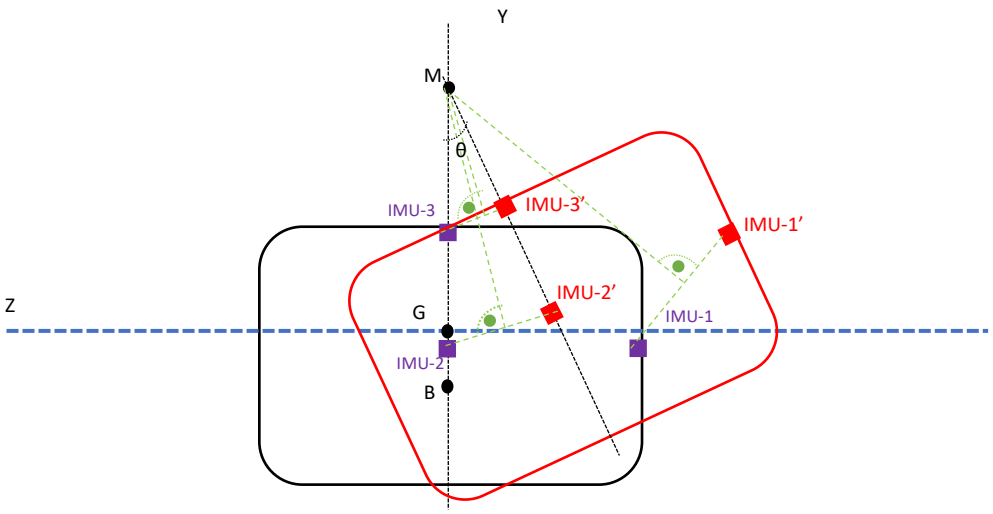

Figure 4:Metacenter Calculation

Also the metacentric height can be calculated using measured angular velocities:[3]

$$GM = \frac{0.16 \times \omega^2 \times b^2}{g} \tag{5}$$

Where ω is the angular velocity around rotation axis, g is the gravitational acceleration and b is the beam length.

Combining these 3 information about the ship which is location of the M, distance GM, and heeling angle θ which is also the angle between the vertical line or z axis and the GM vector.

I would like to add that in [3] this combination of information is either wrong calculated or wrong expressed because they expressed the location of M as G in the results section.

Also contrary to claims, vessel's beam length has to be known in order to calculate GM distance.

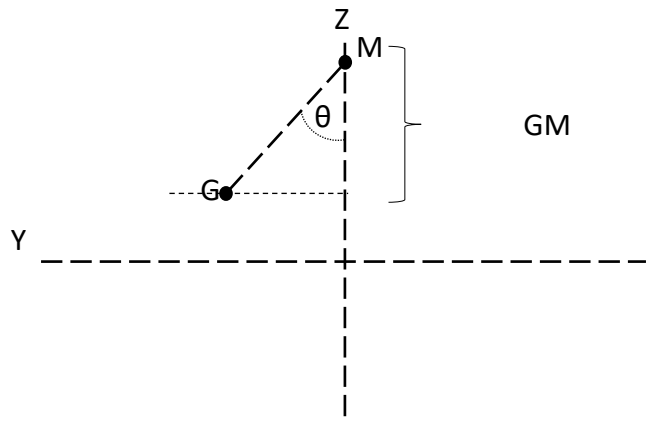

Figure 5:Combination of Parameters to Find G

2.2 Pendulum Validation

In order to verify the rotation center calculation which relies on position of sensors, a pendulum is used, which rotation center's location is already known relatively to sensor locations.

A Rolling bearing is used to ease the movement of pendulum. A stiff wooden rod is used as pendulum rod in order to fully transmit the movement. Measuring the acceleration of the sensors which will be mounted on the tip of the rod, we can detect location thus can calculate the rotation center, which will verify the propesed method's approach.

Sensors mounted in the orientation that rotation is around x axis and gravity is in z axis as minus g.

2.3 Simplified Model Vessel

Next step would be the verification of G. Thus a simplified model vessel was manufactured using a 3D printer.

First 3D solid model is created using SOLIDWORKS student edition software. In order to easliy print the model, a basic square model is used. Seats for sensors in different locations are added to the model for easily accommodate them.

As Printer, Ender 3 V2 PRO is used. Material used is Polylactic acid (PLA) is a biodegradable and bioactive thermoplastic aliphatic polyester derived from renewable resources, such as corn starch, sugarcane, or cassava. It is one of the most commonly used materials in 3D printing. PLA has a relatively low melting point, making it easy to extrude through a 3D printer's nozzle. It has good dimensional stability, minimal warping, and does not emit toxic fumes when heated.

3D printed model vessel brings one important feature, already known center of gravity to verify calculated center of gravity using proposed method.

2.4 Metacenter Calculation

In order to calculate metacenter from calculated position data we need two different positions of same sensor during the movement.[3] As demonstrated in figure-4 from initial position to final position, a line needed to be drawn for each sensor.I named those lines as LINE in order to prevent confusion. Then from midpoints of the each 3 LINE a perpendicular line must be drawn which I named VERTICAL. Intersection of VERTICALS for two different sensor must give center of rotation.[3] Metacenter is calculated for each 2 different pairs of sensors which means 3 metacenter points. Ideally, those 3 metacenters must coincidence, but due to measurement and calculation errors, they can be different points.

I wrote a script to verify this approach. I created a circular path as positions. Ideally result should have been the center of circle.

First LINES must be drawn. I selected 3 different points on the circle as initial locations of sensors and their corresponding final location is calculated. Then slope of the LINES are calculated from initial and final position and equation of LINES are determined using midpoint of LINE.

$$m_L = \frac{z_f - z_i}{y_f - y_i} \tag{6}$$

$$z_{midpoint} = \frac{z_f + z_i}{2} \tag{7}$$

$$y_{midpoint} = \frac{y_f + y_i}{2}$$

$$C_L = z_{midpoint} + y_{midpoint} \times m_L \tag{8}$$

Next, slope of the VERTICALS are determined considering slope of two perpencular lines are equal to -1. Since slope and one of the points are know(midpoint), equation of line of VERTICALS are determined.

$$m_V = -\frac{1}{m_L} \tag{9}$$

$$C_V = z_{midpoint} + y_{midpoint} \times m_V$$

Then using Symbolic Algebra Toolbox of MATLAB, intersection points of VERTICALS are calculated. Results plotted in figure-6.

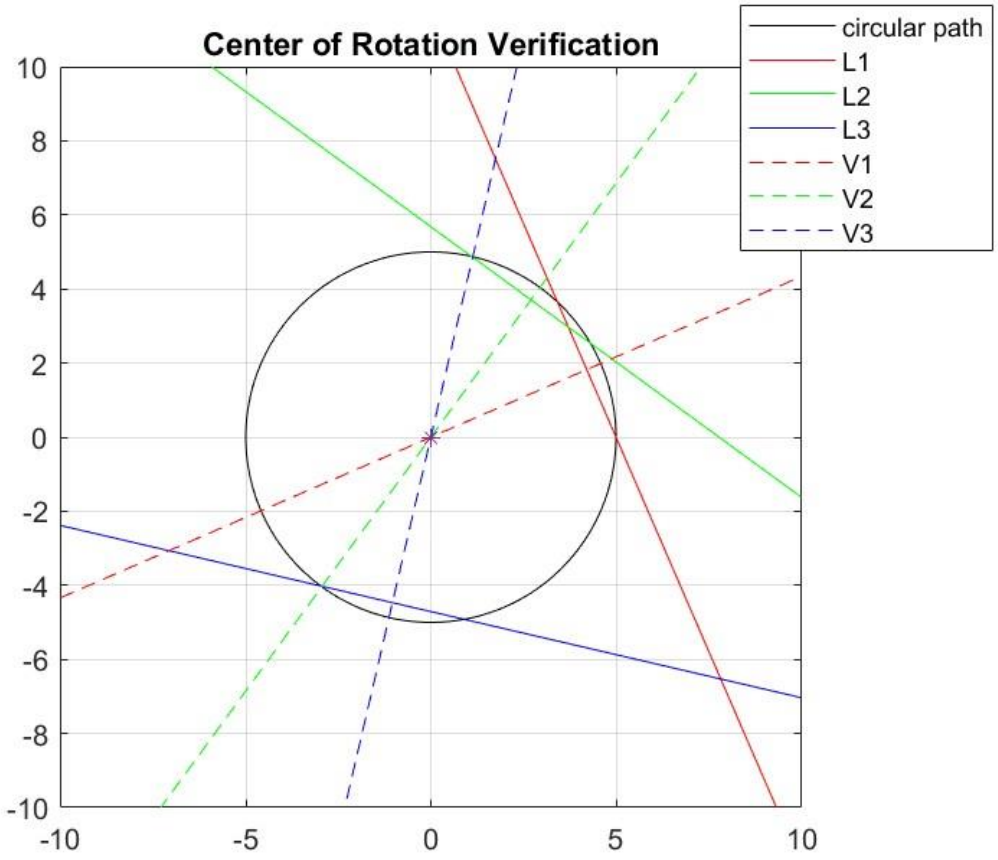

Figure 6:Metacenter Calculation Verification

2.5 SENSORS

For the IMU sensor required MPU6050 is selected for this project which is a microelectromechanical systems (MEMS) sensor. MEMS sensor is a type of sensor that utilizes miniaturized mechanical and electro-mechanical elements that are integrated with electronics on a small silicon chip. These elements, known as micro-mechanisms, can be used to detect and measure various physical phenomena such as acceleration, angular rate, pressure, temperature, and more.

MEMS sensors are commonly used in a wide range of applications including smartphones, automobiles, industrial automation, medical devices, and many more. In automobiles, MEMS

sensors can be used in airbags, anti-lock braking systems, and stability control systems.MEMS technology is continuously evolving and expanding its application field.

MEMS IMU is a type of sensor that combines multiple MEMS sensors in order to provide information about the orientation and movement of an object. An IMU may include accelerometer, gyroscope, and sometimes a magnetometer. The accelerometer measures linear acceleration along the three axes of the device. The gyroscope measures angular velocity, allowing for the detection of rotation and changes in orientation. The magnetometer measures the strength and direction of the magnetic field, allowing for the detection of the direction of north. Combining the output of the accelerometer, gyroscope, and magnetometer MEMS IMU can provide a more complete picture of the movement and orientation of the object it is attached to. The sensor used in this project(MPU6050) only consist of 3 axis accelerometer and a 3 axis gyroscope.

Accelerometer consists of a small proof mass, which is suspended on a spring-like structure. When the device is subjected to linear acceleration, the proof mass is displaced, causing a change in capacitance. This change in capacitance is then converted into a voltage, which can be read by the accelerometer's electronics to determine the acceleration.

Gyroscope works by measuring angular velocity, which is the rate of change of orientation. It consists of a small rotor, which is suspended on a spring-like structure. When the device is subjected to angular velocity, the rotor begins to rotate. The rotation of the rotor causes a change in capacitance, which is then converted into a voltage that can be read by the gyroscope's electronics to determine the angular velocity.

The MPU6050 also has a built-in temperature sensor, which can be used to measure the temperature of the device and compensate for any temperature-related effects on the accelerometer and gyroscope. Also this implies that sensor readings depend on temprature.

2.6 Arduıno Uno and I2C Protocol

In order to use this sensor I used Arduino UNO. Arduino Uno is a microcontroller board based on the ATmega328P microcontroller. It is an open-source platform that allows for easy programming and interaction with sensors and other devices. The board includes a variety of input and output pins that can be used to interface with a wide range of sensors, actuators, and other devices, as well as a USB connection for programming and power. The Arduino Uno has

14 digital input/output pins, 6 of them can be used as PWM outputs and 6 analog pins(A0 to A5) which I used.

The Arduino Uno uses the Arduino IDE (Integrated Development Environment) for programming, it is a free software that allows users to write and upload code to the board. The Arduino IDE supports C and C++ programming languages also.

The communication between an Arduino Uno and an MPU6050 can be achieved using the I2C (Inter-Integrated Circuit) protocol and was developed by Philips Semiconductors (now NXP Semiconductors) in the early 1980s. It is a type of serial communication protocol that allows multiple devices to communicate with each other over a single pair of wires. In I2C, there are two main types of devices: master and slave devices. A master device controls the communication on the bus and initiates data transfers, while a slave device responds to the master's requests and sends or receives data. The I2C bus uses two wires to transmit data: SDA (Serial Data) and SCL (Serial Clock). The SDA line carries data in both directions between the master and slave devices, while the SCL line carries the clock signal, which is used to synchronize the data transfers.

Since we intended to use 3 MPU6050, we needed to use an I2C multiplexer which is a device that allows me to use multiple slave devices which share the same I2C address. It works by routing the communication between the I2C master and the various slave devices connected to it, based on the slave address. An I2C multiplexer has several channels, each of which is connected to a different slave device. The master can select which channel to communicate with by sending a command to the multiplexer, specifying the address of the desired slave device. Once the channel is selected, the master can communicate with the corresponding slave device as if it were the only device on the bus. I used to TCA9548A multiplexer.

Wiring is done by from all sensors to blackboard 3.3 volt and grounding are connected using jumper cables. Also Arduino Uno and TCA9548A are connected with 3.3 volts and groundings using corresponding lines on the blackboard.

From Arduino Uno(A4 pin) to TCA9548A SCA pin and Arduino Uno(A5 pin) to TCA9548A SCL pins are connected using jumper cables. TCA9548A consists 8 different I2C lines.3 different I2C lines to 3 sensors, SDA and SCL lines are connected using jumper cables.

In the software part inside the void loop one TCA9548A line are opened and after corresponding sensor readings are made and printed to the serial port monitor, after another line

is opened for the other sensor to make readings and print on serial port monitor. This process is done for 3 sensors.

2.7 NUMERICAL INTEGRATION

To be able to integrate data, I used trapezoidal rule which is a numerical integration method used to approximate the definite integral of a function. It is a simple and efficient method that can be used to approximate the area under a curve. The basic idea behind the trapezoidal rule is to approximate the region under the curve as a series of trapezoids, and then add up the areas of these trapezoids to get an estimate of the total area.

The trapezoidal rule works by dividing the region under the curve into a set of small intervals, called subintervals. The x-coordinates of the subintervals are chosen such that they define a set of points on the curve. The height of each trapezoid is determined by the y-coordinate of the function at the two endpoints of the subinterval. The area of each trapezoid is then calculated by multiplying the height by the average of the lengths of the two bases (the subinterval). The areas of all trapezoids are added up to give an estimate of the total area under the curve. However, the accuracy of the trapezoidal rule depends on the number of subintervals used, as the more subintervals, the more accurate the approximation will be.

Trapezodial rule has applied using a built-in MATLAB function called “cumtrapz”, result is a vector of the cumulative integral at each data point.

2.8 Pre Processing

2.8.1 Sensor Bias

After measurements are done, data is printed on Arduino’s Serial Port Monitor then transfered MATLAB envoriment and stored as. mat file.

Since MPU6050 is a low cost IMU, measurements consist bias. Sensor bias refers to the systematic error that is present in a sensor's measurement. It is a deviation from the true value of the quantity being measured. The bias error can be caused by various factors such as temperature variations, mechanical misalignment, or electronic noise.

For example, in an accelerometer, bias can be caused by the accelerometer's mechanical alignment or the accelerometer's electronic circuit. In this case, bias can cause the accelerometer to read a non-zero acceleration when it is not moving (stationary) and this error is called a "zero-g bias".

In a gyroscope, bias can be caused by the temperature of the device, electronic noise or mechanical misalignment. This bias can cause the gyroscope to read a non-zero angular velocity when it is not rotating (stationary) and this error is called a "bias drift".

Bias is a constant error and it does not change over time, unlike random error which is caused by factors such as noise that can vary over time. Thus I removed bias by substracting stationary and right oriented measurement's mean from dynamical movement's measurement.

2.8.2 Orientation Calculation

In order to calculate the orientation of the vessel, or pendulum motion, [3] offered the use of accelerometer measurements. (2)

When I calculate orientation based on acceleration rates as equation 2 states, results are far away from being suitable for use of angle of heel.

Instead, I applied numerical integration to the angular velocity measurements which are quite accurate and can be seen on figure-7 and figure-8.

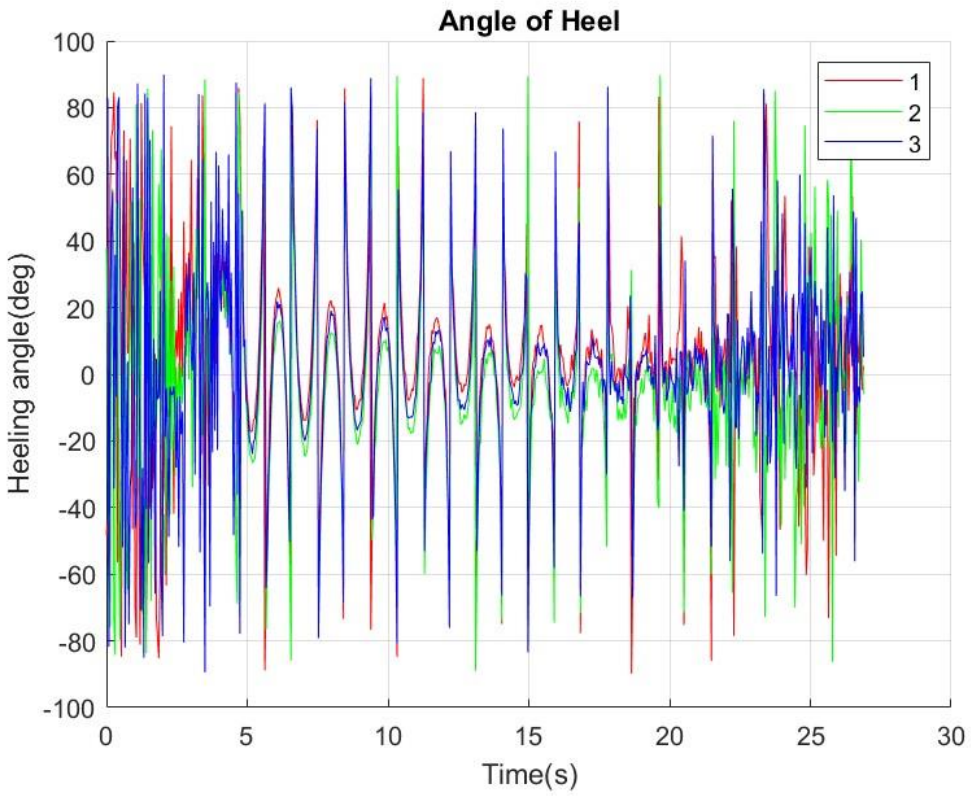

Figure 7:Accelartion based orientation on pendulum experiment

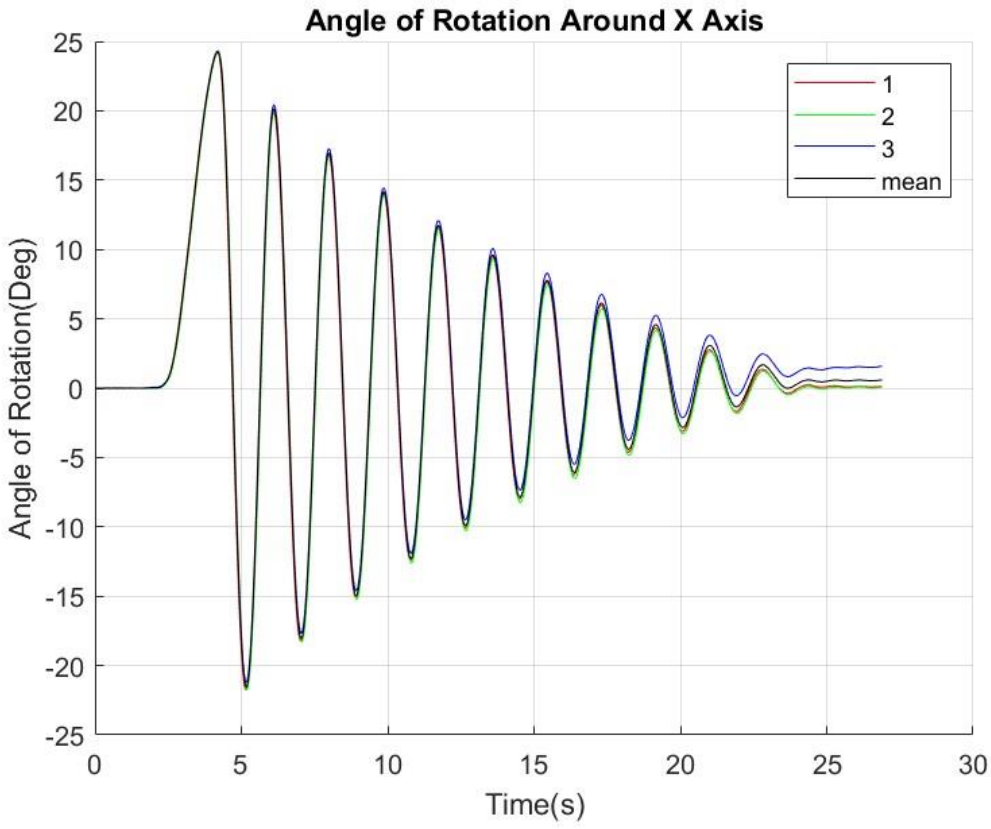

Figure 8:Integrated angular velocity based orientation on pendulum experiment

2.8.3 Axis Correction

On the experiments, sensors are mounted on the model vessel, their orientation is changing when they start rotating, but the accelerometer measures acceleration in the direction of sensor's body frame which is not true when body frame is not coincidal with global frame, thus I needed to do a coordinate transformation.

Since I already calculated orientation changes using numerical integration of angular velocity measurements, I can use rotation matrices to overcome this problem.

A rotation matrix is a matrix that is used to perform a rotation in a 3D space. It is a matrix that can rotate a vector in a two or three-dimensional space around a specific axis. Axis correction of accelerometer measurements can be done by applying a rotation transformation to the accelerometer's measurements to align them with a global reference frame.This is basically meaning the vector multipcation of corresponding acceleration measurement with rotation matrix with the corresponding orientation change.

$$[X'(i) \quad Y'(i) \quad Z'(i)] = [X(i) \quad Y(i) \quad Z(i)] \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x^i) & -\sin(\theta_x^i) \\ 0 & \sin(\theta_x^i) & \cos(\theta_x^i) \end{bmatrix} \tag{10}$$

$$[X''(i) \quad Y''(i) \quad Z''(i)] = [X'(i) \quad Y'(i) \quad Z'(i)] \times \begin{bmatrix} \cos(\theta_y^i) & 0 & \sin(\theta_y^i) \\ 0 & 1 & 0 \\ -\sin(\theta_y^i) & 0 & \cos(\theta_y^i) \end{bmatrix} \quad (11)$$

$$[X'''(i) \quad Y'''(i) \quad Z'''(i)] = [X''(i) \quad Y''(i) \quad Z''(i)] \times \begin{bmatrix} \cos(\theta_z^i) & -\sin(\theta_z^i) & 0 \\ \sin(\theta_z^i) & \cos(\theta_z^i) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (12)$$

Above, rotation transformation about 3 axises is applied to the acceleration measurements time dependent. Subscripts x, y and z in the rotation matrises specifies the rotation transformation is applied about corresponding axis and i means those transformations applied time dependent.

Also MPU6050 measures gravitational acceleration even if no real acceleration of the sensor is present, which means we have to substract gravity in the dynamical measurements of experiment.

Since rotational transformation is applied to measurements, gravity is elimnated from y or x components of data, thus only substarcting gravity from z values gives acceleration only caused from movement.

As a result of those pre-processing applications, acceleration measurements on pendulum experiment in the y and z axises are is shown in figure-9.Since there is no movement in the x axis and x axis displacement will not be used in transverse plane metacenter calculations, x axis results will not be displayed.

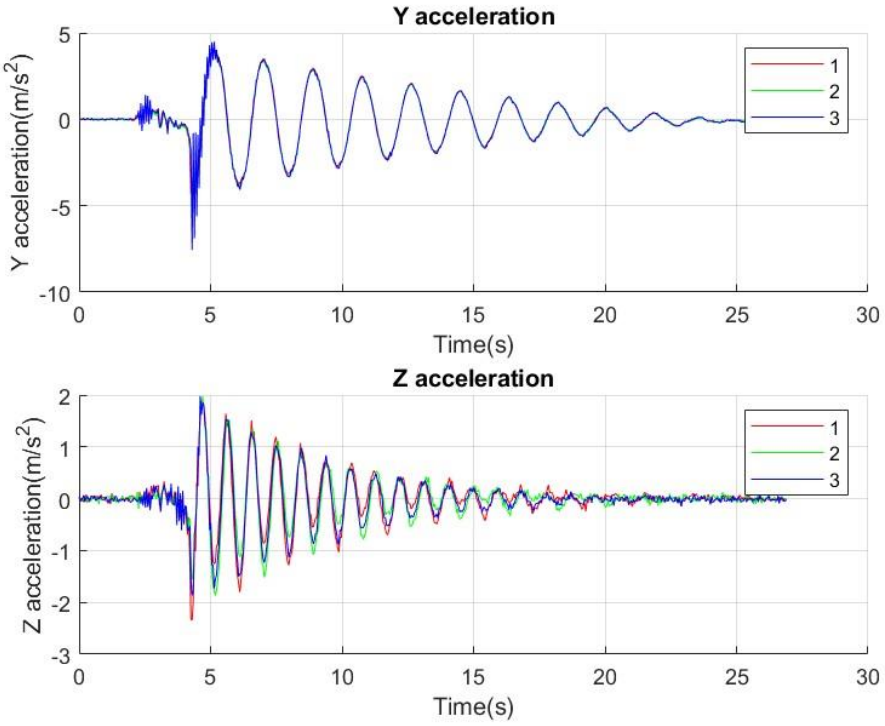

Figure 9: Raw acceleration data

Also after double numerical integration is completed, initial positions will be added to the result of integration.

In order to take effect of rotation in considiration on the initial positions, they are also multiplied with rotation matrices for during the movement.

2.9 Applying Numerical Integration to Acceleration: Integration Drift

Since metacenter calculation is verified and acceleration data are ready to process in order to find position of sensors during the movement, now we can apply double numerical integration to acceleration data to get first velocity then position data.

Results of the numerical integration showed that acceleration data after numerical integration is not suitable to calculate metacenter, integrated data is gradually deviating . Even more applying numerical inegration to deviated velocity yields more deviation results on position which is shown on Figure-10 and Figure-11.

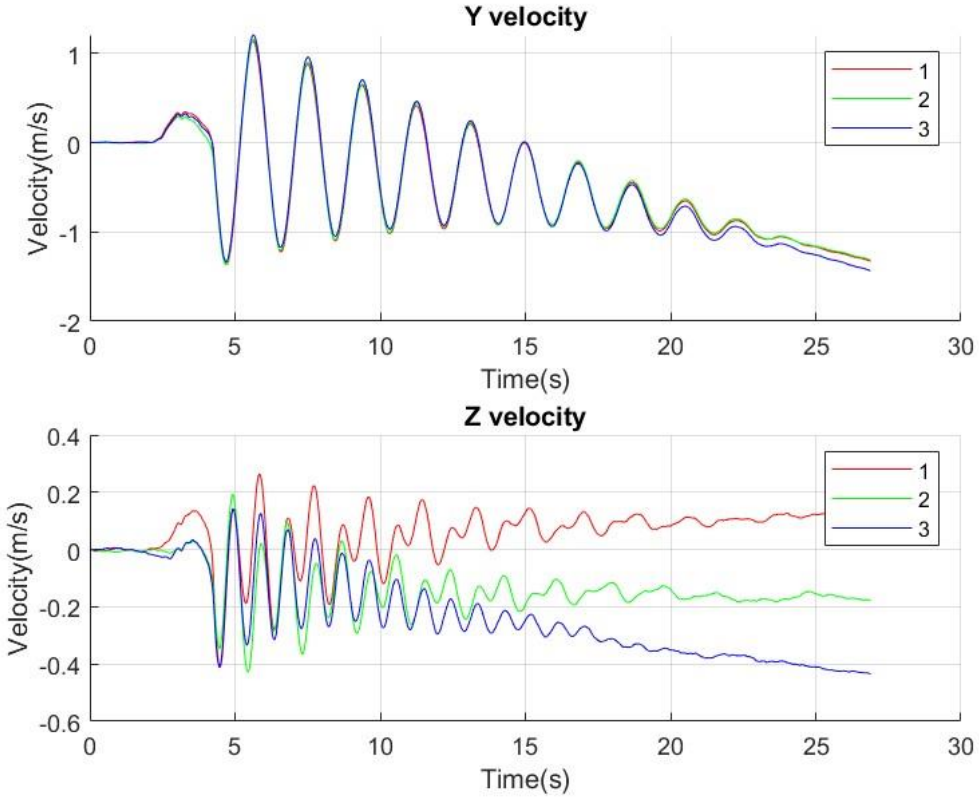

Figure 10:Linear Velocity Integrated From Raw Acceleration Data

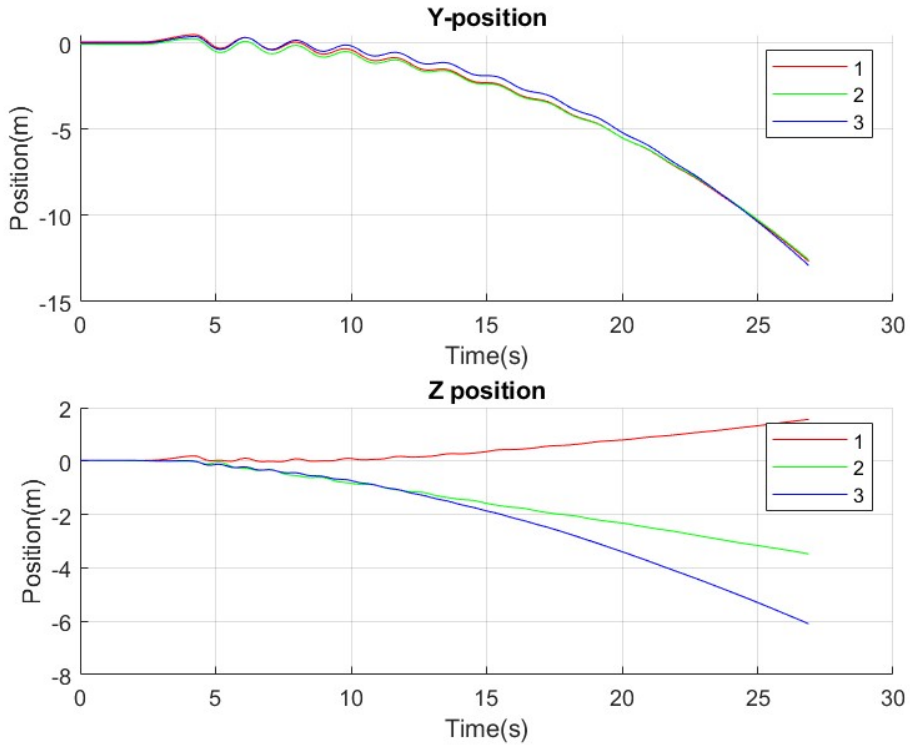

Figure 11:Linear Position Double Integrated From Raw Acceleration Data

This gradual deviation is called integration drift. When integrating accelerometer data to determine position, we are essentially calculating the displacement of an object over time by summing up the accelerations measured by the accelerometer. However, accelerometers are not perfect sensors and can have various sources of error such as noise or bias. Nevertheless I tried to overcome, these errors can cause small inaccuracies in the accelerometer measurements, which can accumulate over time and lead to a significant deviation from the true position. This deviation is known as integration drift.

In order to overcome this phenomena I tried couple of methods such

- Removing noise from the acceleration data which predicts to overcome drift when integrated.
- Highpass filtering integrated data which predicts to remove drifting.
- Substracting the drifting components from integrated data.

2.9.1 Fast Fourier Transform (FFT)

Fast Fourier Transform (FFT) is an algorithm that is used to compute the discrete Fourier transform of a sequence. Discrete Fourier transform is a mathematical operation that transforms a discrete-time signal into a representation in the frequency domain. The FFT is a more efficient method for computing the DFT, as it reduces the computational complexity.

FFT is a tool that providing the opportunity to see the different frequencies that make up a signal. It can be used to analyze the frequency content of a signal, and it's a powerful tool for signal processing and analysis. It is widely used in various fields such as audio and image processing, telecommunications, control systems and many more.

While applying FFT, frequency of the data determined using Nyquist-Shannon sampling theorem which states that if a signal is sampled at a rate of at least twice the highest frequency present in the signal, the original signal can be reconstructed from the samples without loss of information. Therefore, the highest frequency present in the signal is equal to half of the sampling frequency, also known as the Nyquist frequency.

In order to apply filters, I needed to apply FFT to data to able to determine cutoff frequencies.

2.9.2 Lowpass Filtering Acceleration Measurements

One way to remove noise from measurement is to applying a lowpass filter. A low-pass filter is a signal processing element that is used to remove high-frequency components from a signal,

while allowing low-frequency components to pass through. It is commonly used in a wide range of applications, such as audio and image processing, control systems, and telecommunications.

Lowpass filters can be divided into 2 parts, digital lowpass filters and analog lowpass filters considering application method. Key difference is that an analog low-pass filter can be implemented in the time domain using electronic circuits, while a digital low-pass filter is typically implemented in the frequency domain using mathematical equations or algorithms.

I used digital Butterworth-zero phase lowpas filter to the acceleration measurements.

Butterworth filters are the filters that have a smooth transition between the passband and the stopband, and are characterized by a constant gain in the passband and a monotonically decreasing gain in the stopband [4]. Butterworth filters are commonly used in applications such as audio processing and control system.

Other charactistic of filter is being zero phase in order to prevent phase shift. Zero phase filters are type of filters that preserves the phase relationship between the input and output signals, which means that the output signal is not shifted in time with respect to the input signal.

Also order of the filter was selected as low as possible in order to prevent data distortion.

Outcome of FFT to accelerometer measurements are plotted for all 3 sensors.Y axis represents power spectrum and X axis represents frequency. Cutoff frequencies are determined using those graphs.

2.9.3 Moving Avarage Filter

Another method to cancel noise is applying a weighted moving avarage filter which is simple filter that takes the average of a set of consecutive samples in a signal. It works by computing the average of the past "n" samples of the signal, where "n" is the number of samples used in the moving average calculation. This average is then used as the filtered output.

Applying this filter is done by using a built-in MATLAB function called ‘smoothdata’. Inside the function, We can adjust length of the sliding window of data smoothed.

2.9.4 Kalman Filter as a Noise Canceller

In [5], Kalman filters is used to reduce noise on signals. The Kalman filter is a mathematical algorithm that is used to estimate the state of a system based on a series of noisy and uncertain measurements. It is a recursive algorithm that uses a combination of predictions and measurements to produce an estimate of the true state of the system, or it can be used in sensor

fusion algorithms in order to estimate state of the system based on different sources of sensors. Kalman filter is used in a wide range of applications such as navigation, control systems, and signal processing.

Kalman filter algorithm is typically composed of two main steps: prediction and correction.

Prediction: The prediction step uses the system's model and the previous state estimate to predict the current state of the system. The prediction step is based on the assumption that the system's dynamics can be modeled by a set of linear equations.

Correction: The correction step uses the current measurement to update the predicted state estimate. The measurement is used to correct the prediction and produce a more accurate estimate of the true state of the system.

The Kalman filter algorithm uses: state transition matrix, measurement matrix, process noise covariance matrix and the measurement noise covariance matrix. State transition matrix describes how the state of the system changes over time, while the measurement matrix describes the relationship between the state of the system and the measurements.Process noise covariance and the measurement noise covariance matrices describe the uncertainty in the system's dynamics and the uncertainty in the measurements, respectively.

Kalman filter can also be extended to handle non-linear systems by using variants such as the extended Kalman filter and the unscented Kalman filter, but [5] states that for noise reduction, linear Kalman filter is sufficient.

Kalman filter equations provided in[5] or many more resources as:

Predict:

$$\widehat{x_{t|t-1}} = F_t x_{t-1|t-1} + B_t u_t \tag{13}$$

$$P_{t|t-1} = F_t P_{t-1|t-1} F_t^T + Q_t \tag{14}$$

Update:

$$\widehat{x_{t|t}} = \widehat{x_{t|t-1}} + K_t\left(y_t - H_t\widehat{x_{t|t-1}}\right) \tag{15}$$

$$K_t = P_{t|t-1}H_t^T\left(H_tP_{t|t-1}H_t^T + R_t\right)^{-1} \tag{16}$$

$$P_{t|t} = (I - K_tH_t)P_{t|t-1} \tag{17}$$

$F_t = 1$ because there is no state transition. Thus, reducing the system's input component B_t because the used system does not have any input u_t.

$H_t = 1$ since the sensor data that will be filtered is only consisted of one sensor reading.

When this adjustmenst done Kalman filter flow chart is shown on figure-12

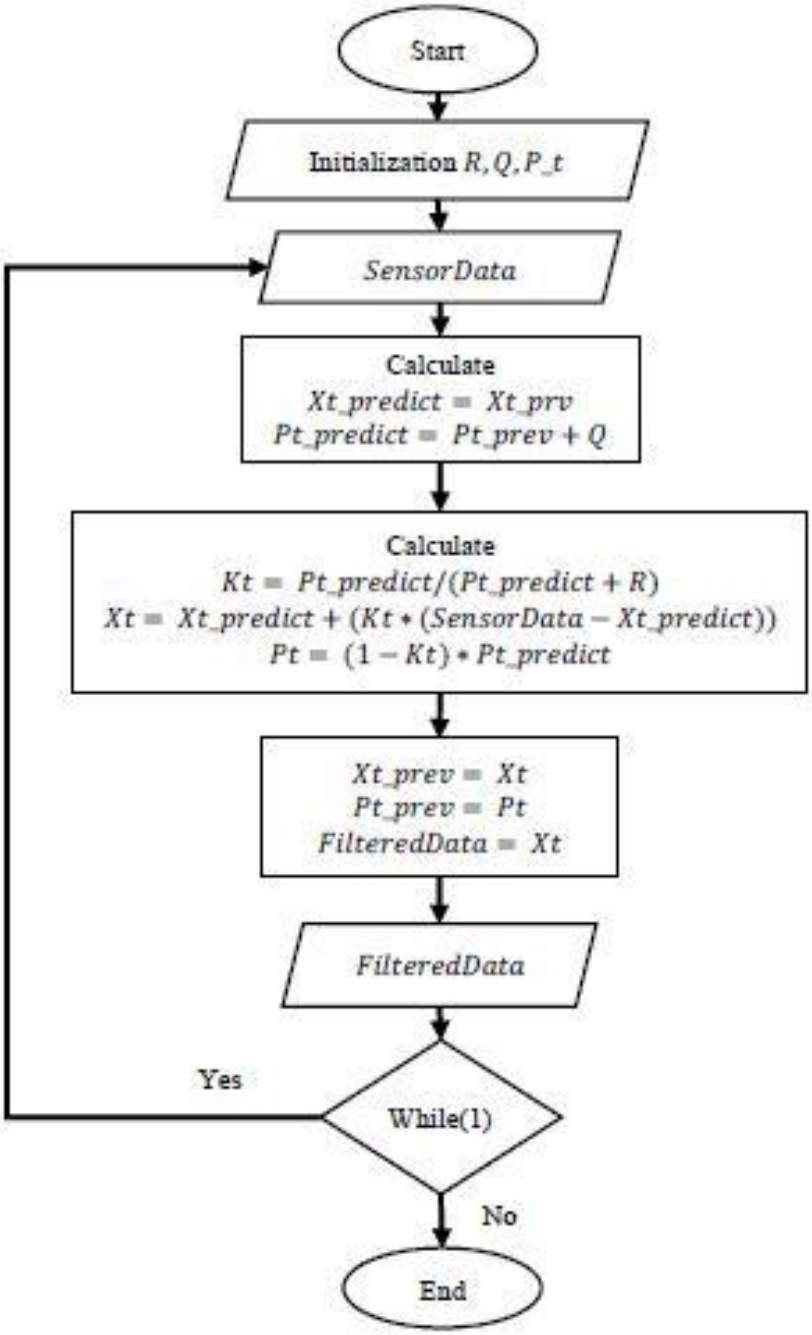

Figure 12:Kalman Filter Flowchart [5]

2.9.5 Applying Highpass Filter

A high-pass filter is a type of filter that allows high-frequency signals to pass through while blocking low-frequency signals. It is the opposite of a low-pass filter, which allows low-frequency signals to pass through while attenuating or blocking high-frequency signals.

In the integrated data(linear velocity and linear position), drift is stored in low frequency signals, so using a higpass filter predicts to overcome integration drift.

As filter, again I used a zero phase Butterworth Higpass filter, with the cutoff frequencies determined from results of the FFT applied to the integrated velocity and position.

2.9.6 Detrend Function

In MATLAB, a built-in function exists to remove polynomial trend on a curve. Inside the function we can adjust order of the trend, also this function can be applied to discontinuous trends.

Goal is the use this function to remove integration drift, by applying it to the integrated results.

3. RESULTS

Data collected from the pendulum experiment is used to test previously discussed methods of overcoming integration drift.

3.1 Butterworth Lowpass Filter

As stated before, FFT is applied to the acceleration measurements. Results shown in figure-13, 14 and 15 for different sensor's measurements on frequency domain with selected cutoff frequencies.

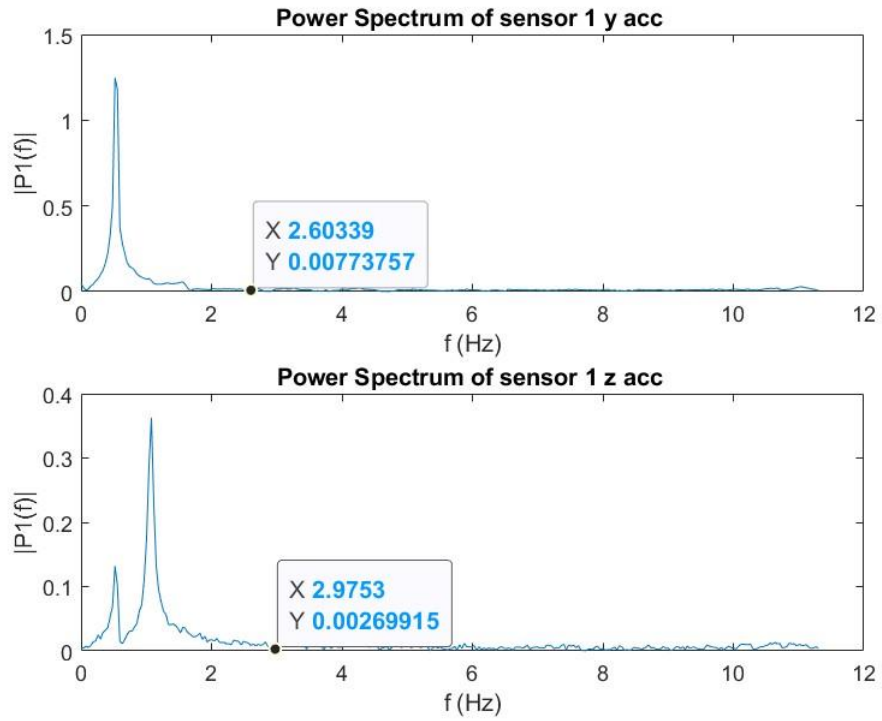

Figure 13:FFT Results of sensor 1 acceleration on pendulum experiment

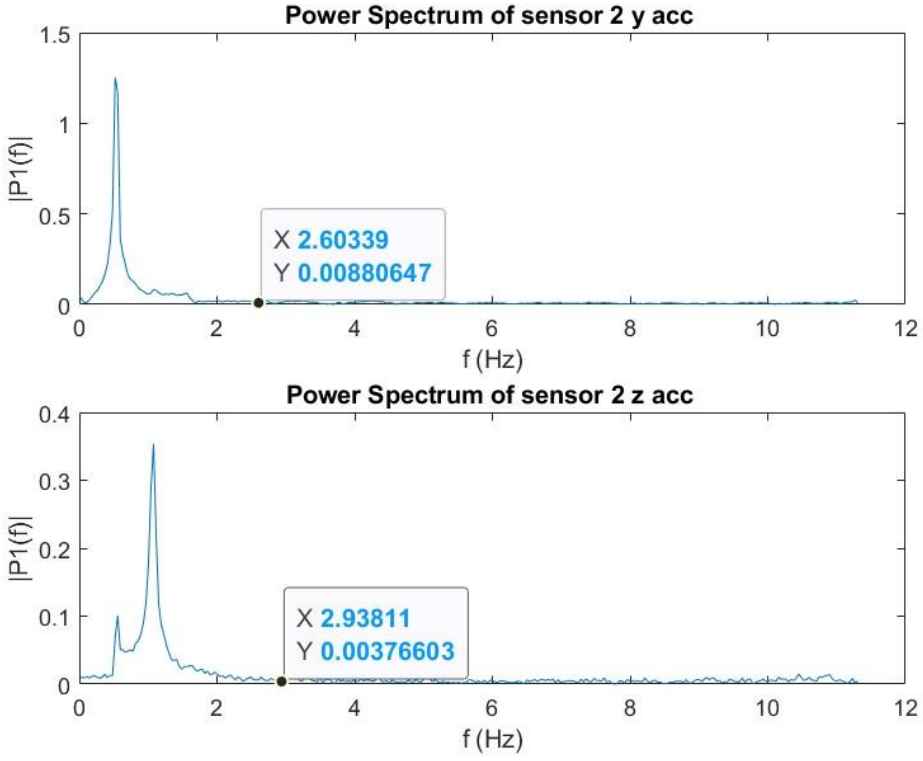

Figure 14:FFT Results of sensor 2 acceleration on pendulum experiment

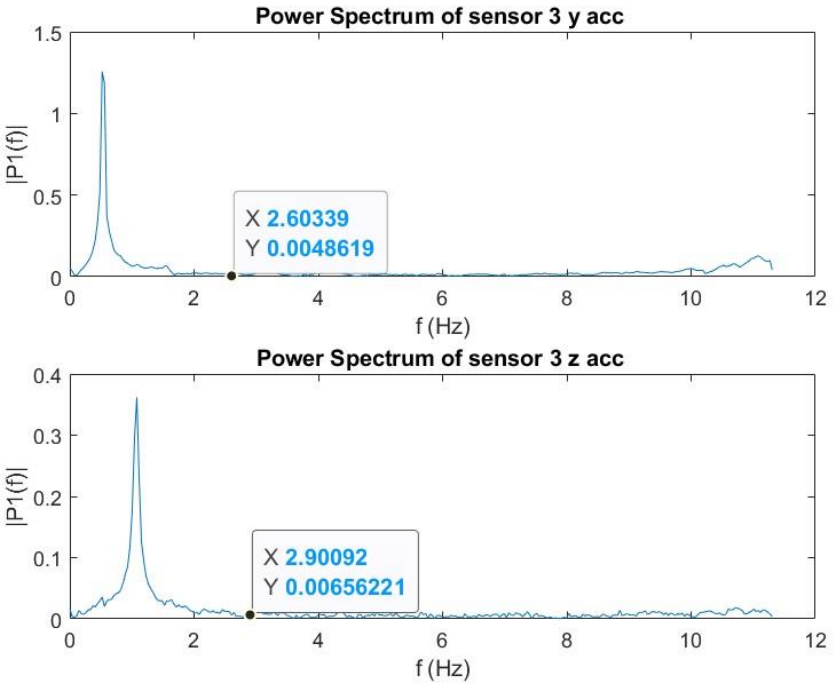

Figure 15:FFT Results of sensor 3 acceleration on pendulum experiment

Cutoff frequencies are selected considering preventing data distortion, then zero phase digital Butterworth lowpass filter is applied.Results show that noise has cancelled and can be seen on figure-16.

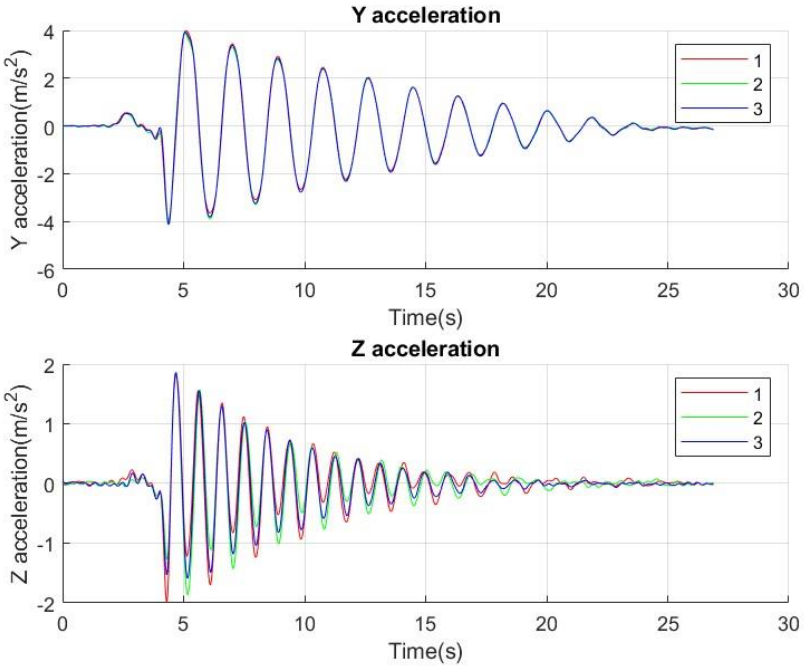

Figure 16:Lowpass Filtered Acceleration Results

It can be observed that noise has cancelled after the application of filter, but drift still exists in the integrated velocity and position.

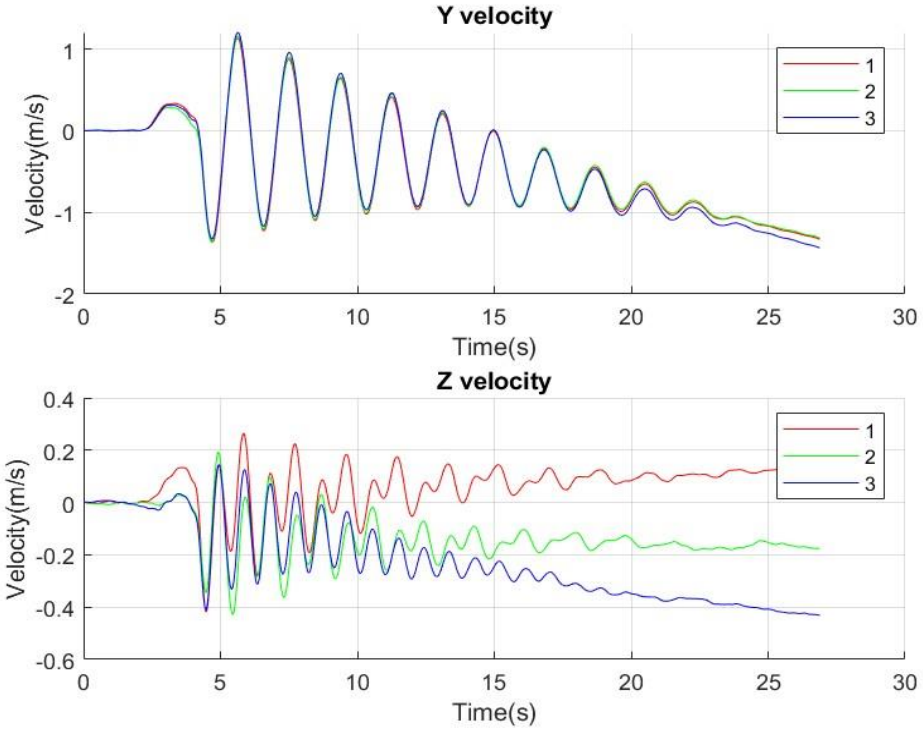

Figure 17:Velocity derived from lowpass filtred acceleration

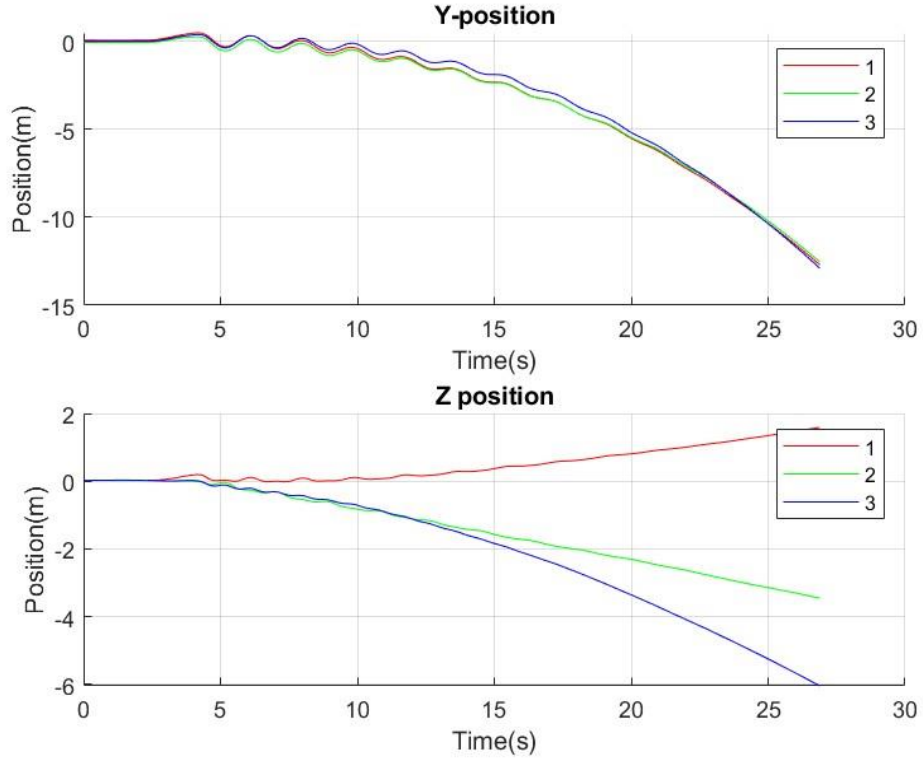

Figure 18:Position derived from lowpass filtered acceleration

Also angle of heel calculated from filtered acceleration measurements, results was better compared to unfiltered, but not still even close to the angle of heel calculated by integration of angular velocity.

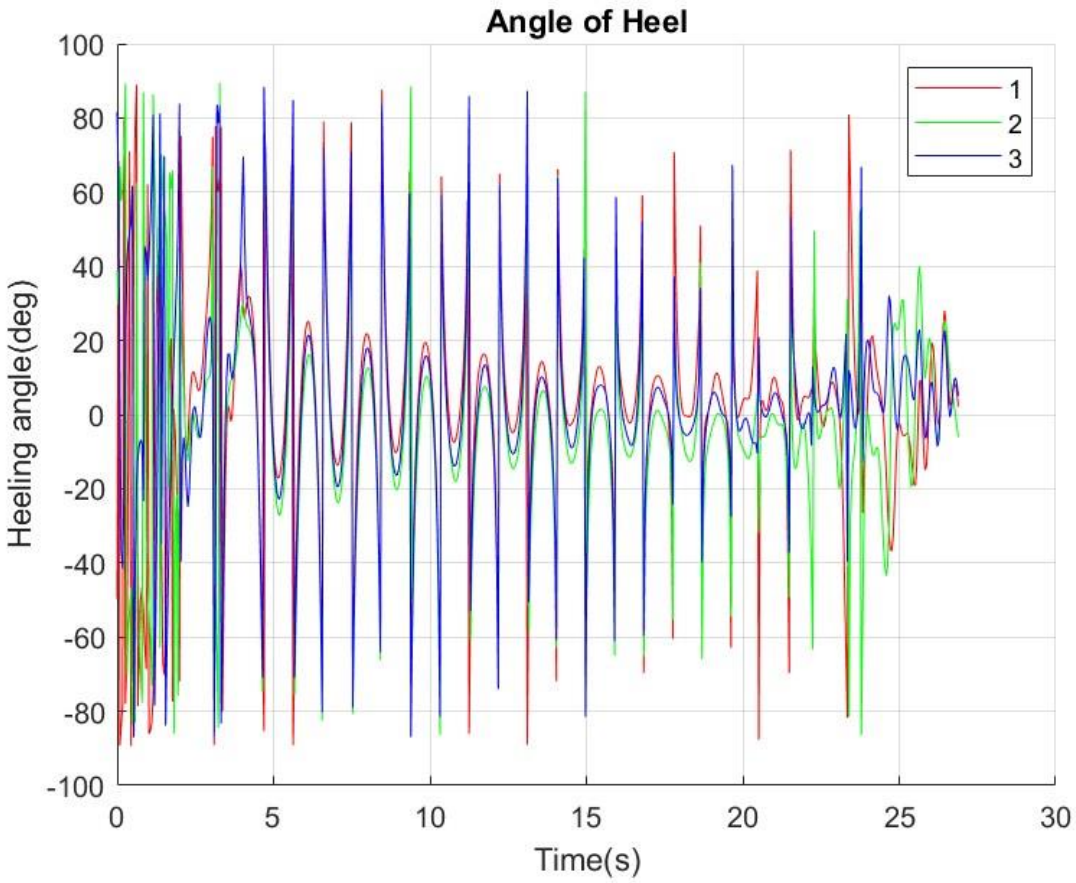

Figure 19:Angle of heel derived from filtered acceleration

3.2 Moving Avarage Filter

In the same way as Lowpass filter, moving avarage filter was able to cancle noise, but far away from being able to remove drift from integrated velocity and position.

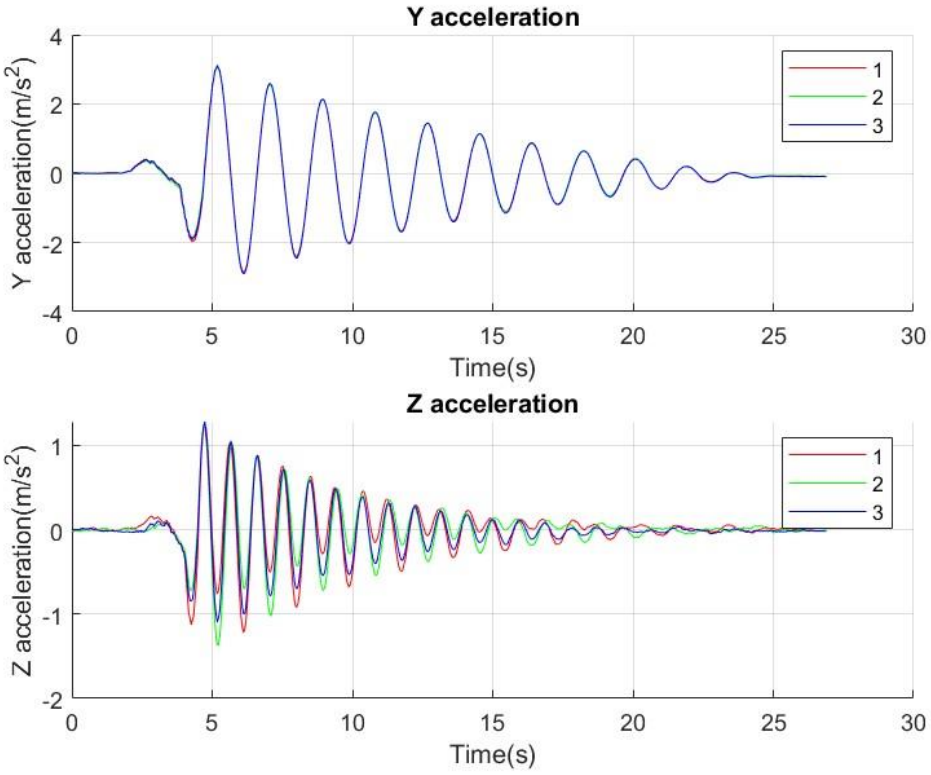

Figure 20:Moving avarage filter application on noisy acceleration

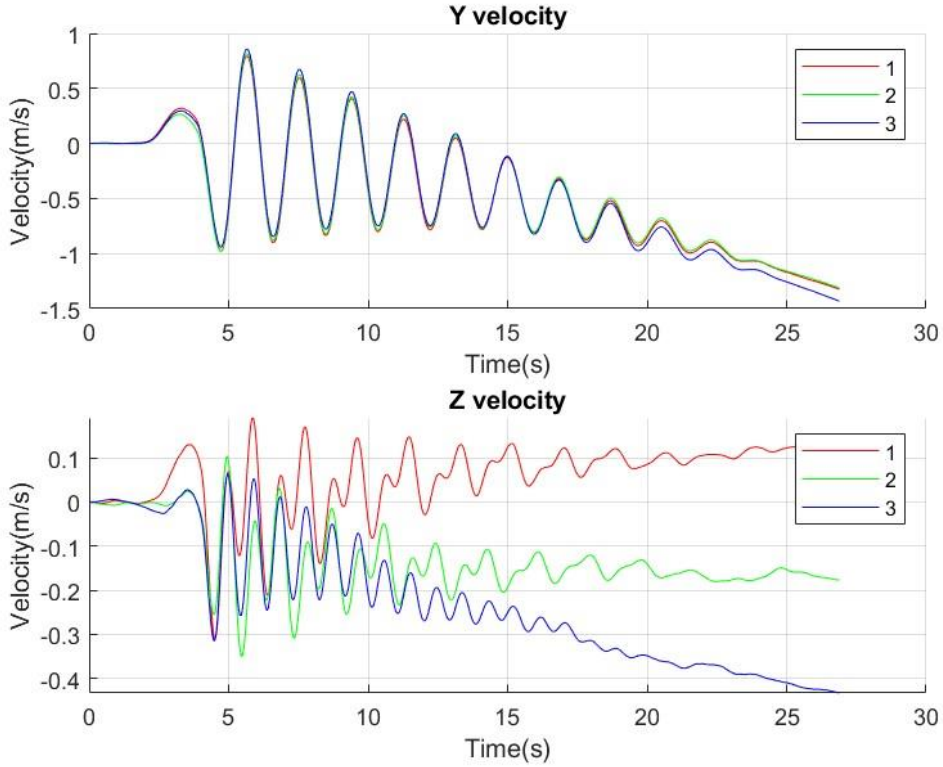

Figure 21:Velocity after moving avarage filter application

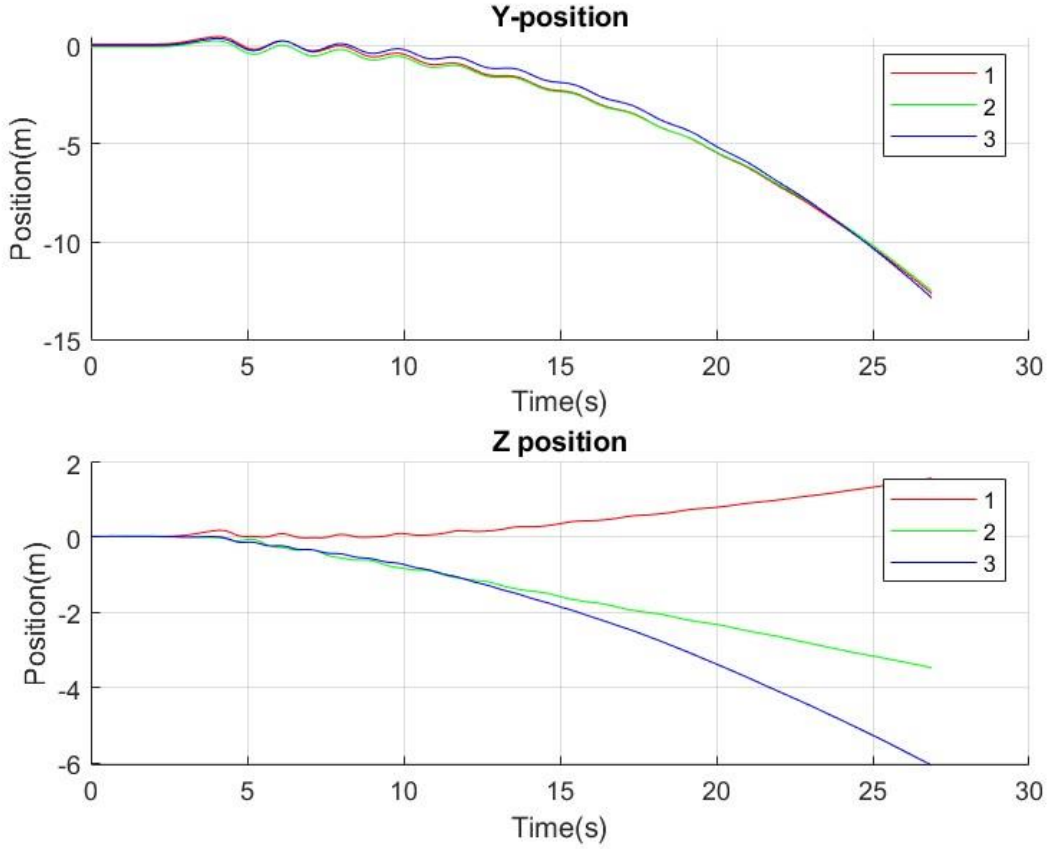

Figure 22:Position after moving avarage filter application

Compared to Butterworth lowpass filter there was data distortion in a way that amplitude of accelerations were smaller.

3.3 Applying Kalman Filter as a Noise Reducer

Another method has been tried in order to cancel noise and overcome integration drift was applying a Kalman Filter.As Kalman Filter variables [5] offered process variant constant Q to be equal to 1 and measurment constant R to be equal to 100.

With this selections, phase shift occured and data is distorted a large amount. Instead i selected Q to be equal to 10 and R to be equal to 60.Results was better this time such as phase shift was considerably smaller and data distortion was not present. Results shown in figure-23. Butc compared to lowpass filter and moving avarage filter, noise exists more.

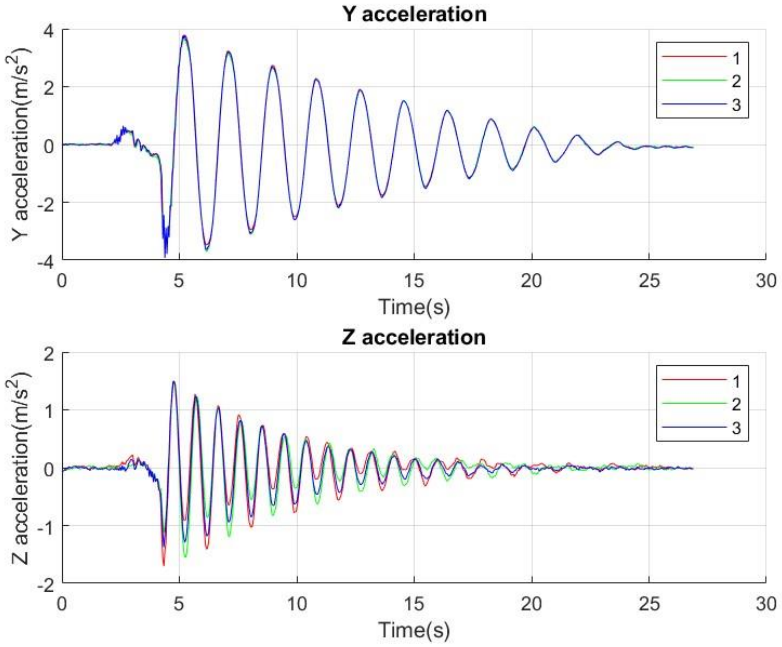

Figure 23:Acceleration data after applying Kalman Filter

Just like other two methods of cancelling noise on acceleration, applying Kalman filter could not achieve to prevent integration drift. Integrated velocities and positions are kept drifting.

Also while cancelling noise small amount of phase shift occured small amount of data amplitude changes.

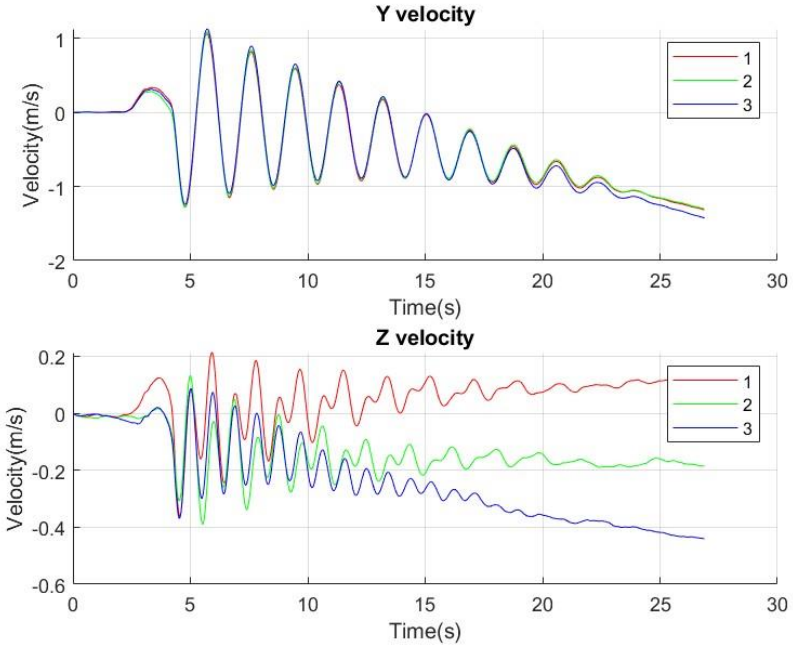

Figure 24:Velocity data after applying Kalman Filter

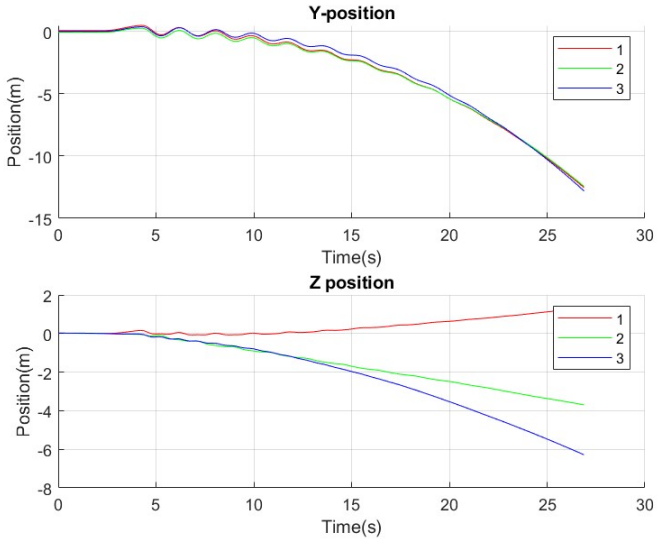

Figure 25:Position data after applying Kalman Filter

3.4 Highpass Filter

A different method has been tried to overcome drift was applying zero phase Butterworth higpass filter to the integrated results of velocity and position.

Lowpass filtered acceleration measurements used to integrate since it gave the best results of cancelling noise without causing data distortion and phase shift. After integration velocity data translated to the frequency domain using FFT method and cutoff frequencies are selected.

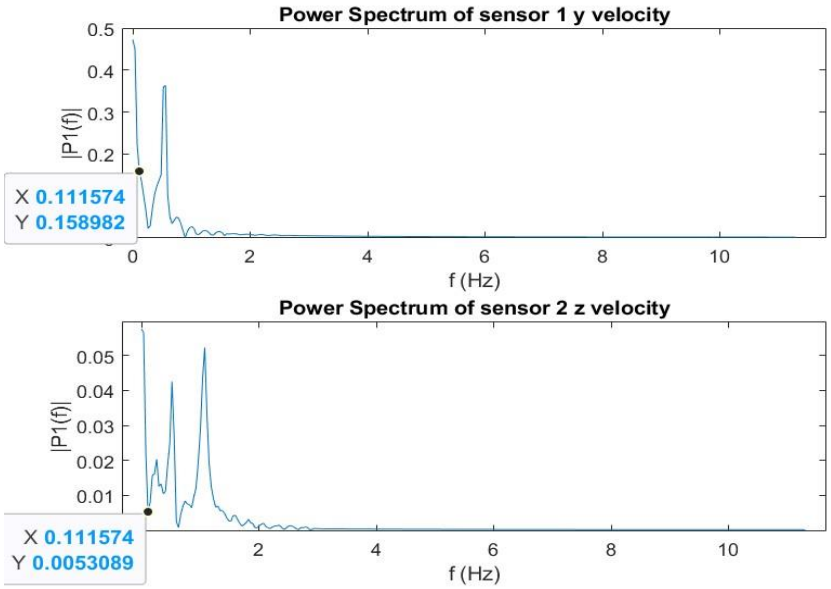

Figure 26:Sensor 1 Velocity on frequency domain

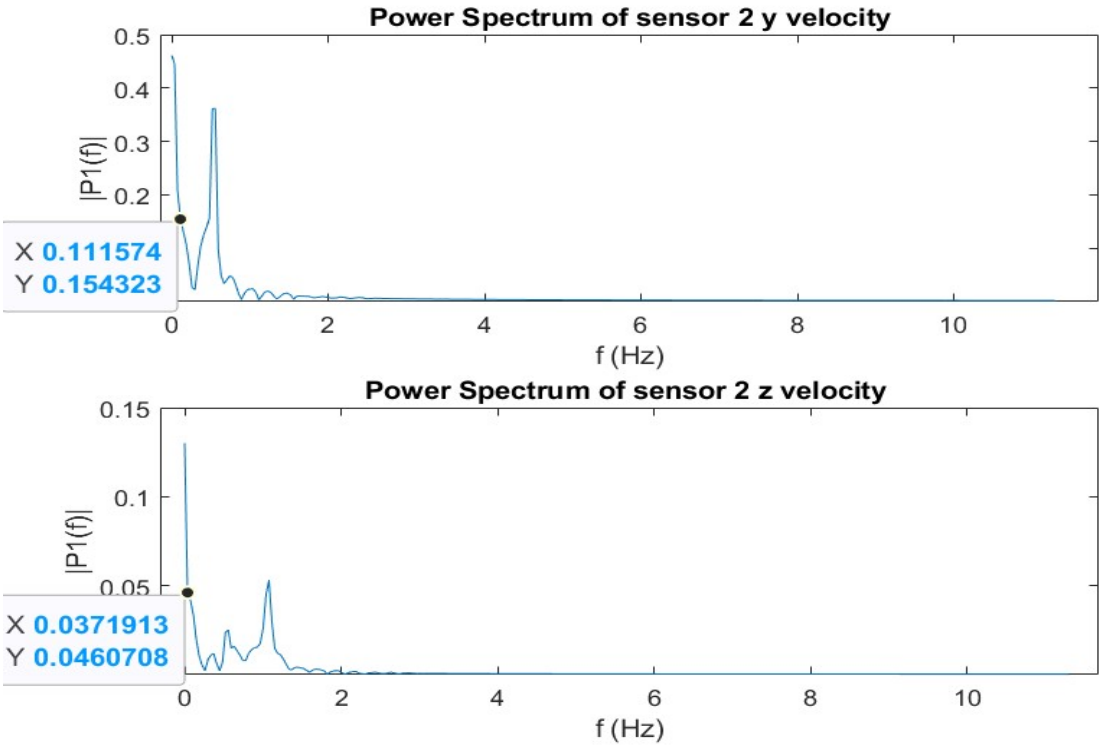

Figure 27:Sensor 2 Velocity on frequency domain

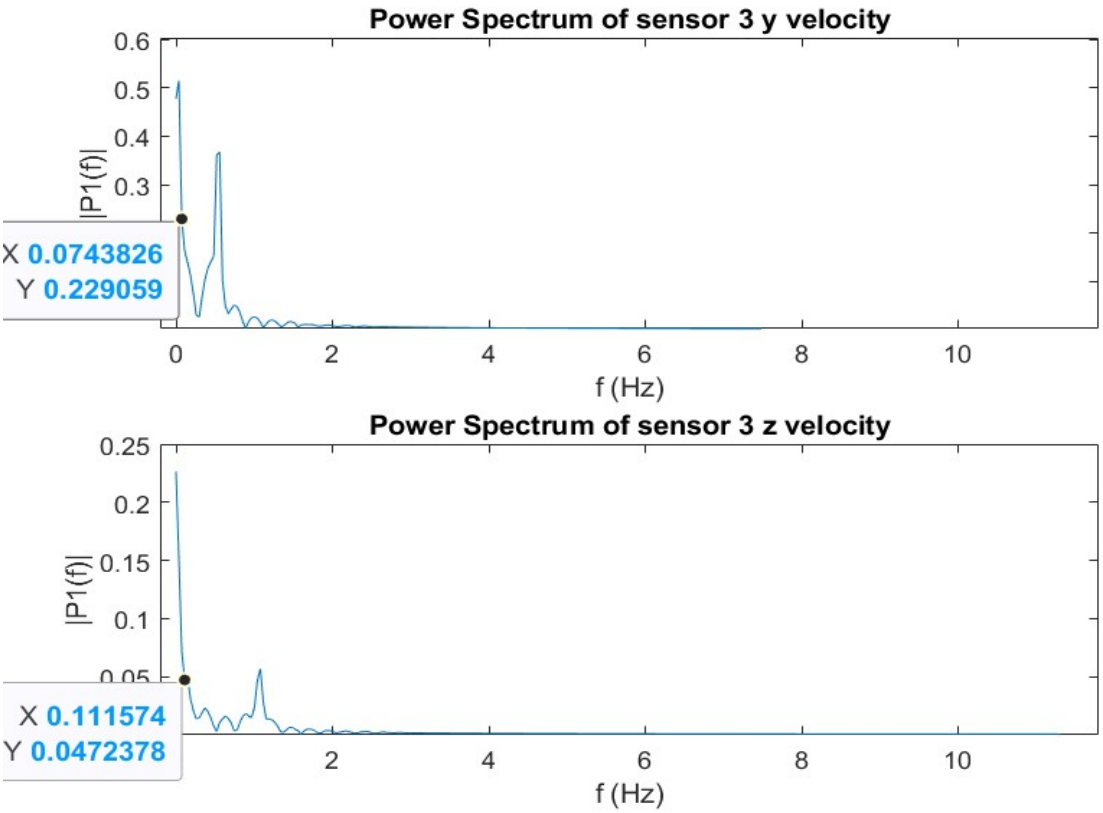

Figure 28:Sensor 3 Velocity on frequency domain

After selecting cutoff frequencies, zero pahse higpass Butterworth filter is applied to the velocity. Results shows that drift is prevented in the velocity data.

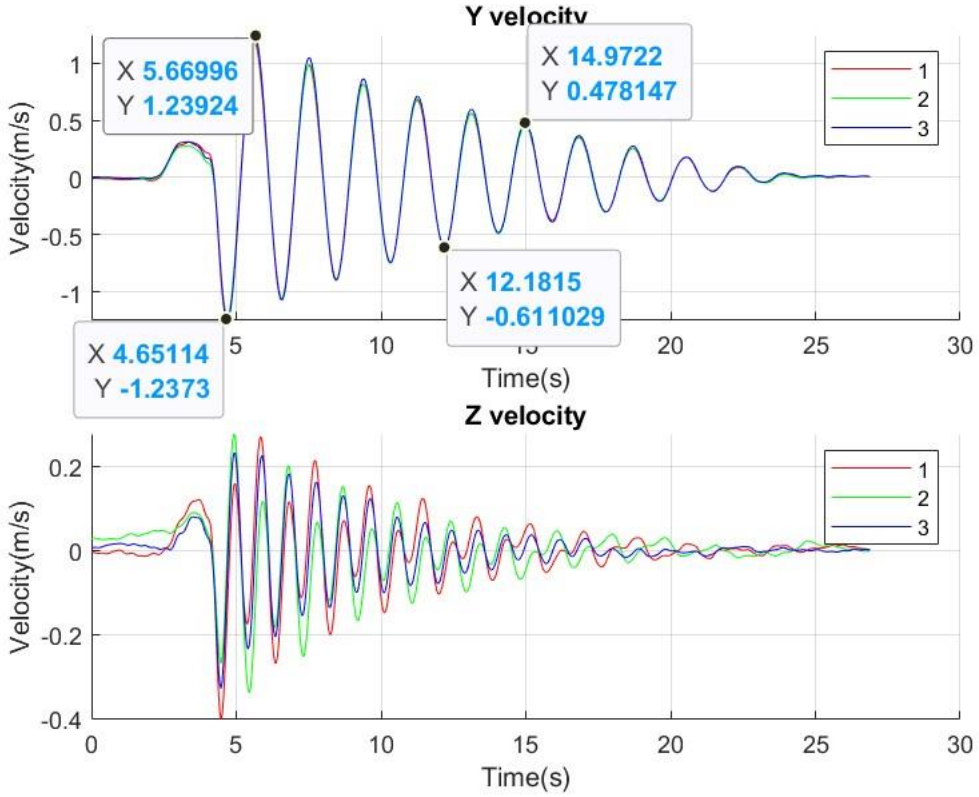

Figure 29:Higpass Filter applied Velocity Results

In order to verify results after filter, velocity results are compared to the accurate angular velocity results, since we know the radius of pendulum. Calculated linear velocity from integration has to be equal to the radius times measured angular velocity in the deadhang position, or max y velocity instant. Radius is 0.95 meters. So velocity calculated with the equation:

$$v\left(\frac{m}{s}\right) = r * \omega\left(\frac{deg}{sec}\right) * \left(2 * \frac{pi}{360}\right) \tag{18}$$

For 4 different points results are tabulated in table-1. It can be observed that result from two different sources are quite similar.

Table 1 : Velocity comparation

Time(s)	Accelerometer based y velocity (m/s)	Gyroscope based y velocity (m/s)
4.65	-1.237	-1.27
5.66	1.239	1.13
12.18	-0.611	-0.607
14.97	0.478	0.434

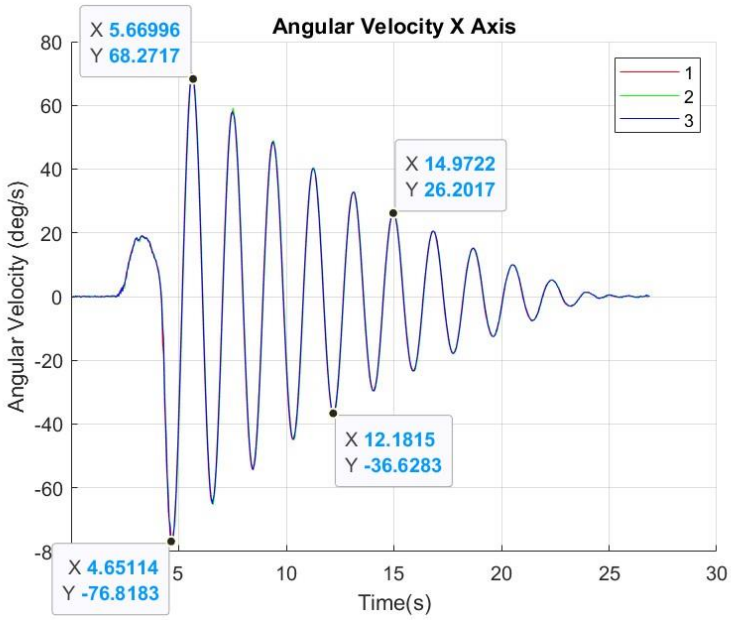

Figure 30:Angular Velocity around x axis

Further, FFT is applied to the velocity data and cutoff frequencies are selected.

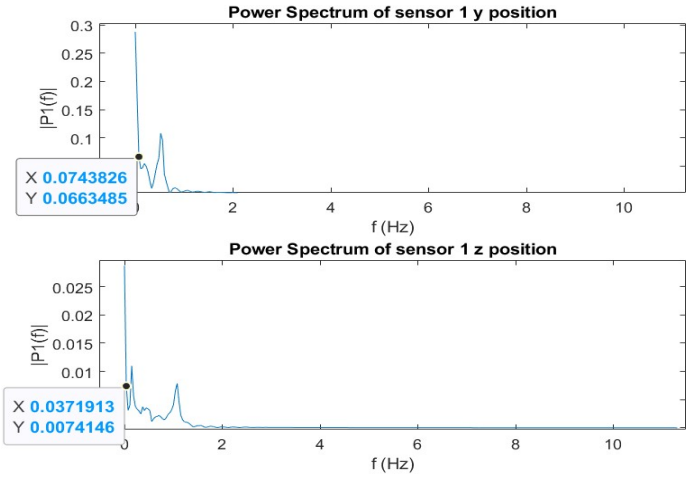

Figure 31:Position data of sensor 1 on frequency domain

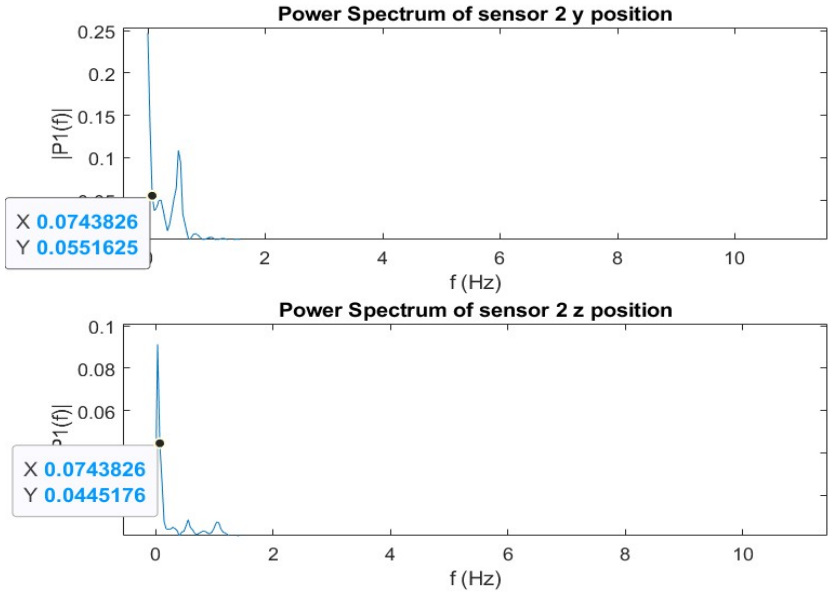

Figure 32:Position data of sensor 2 on frequency domain

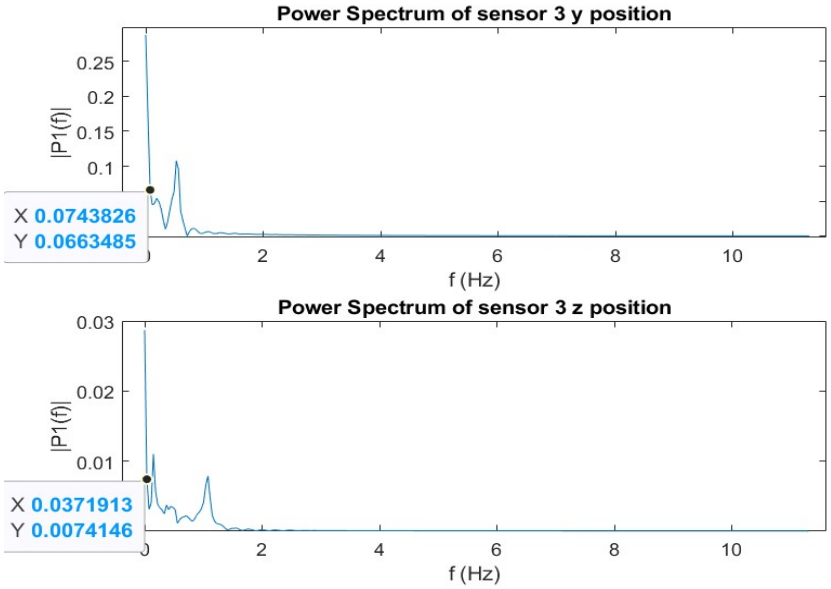

Figure 33:Position data of sensor 3 on frequency domain

After selecting cutoff frequencies, zerophase Butterworth Higpass filter is applied to the postition data. Results show that drift is removed.

In order to verify y and z position calculated numerical integration of acceleration with applied filters, I also caluclated y and z values of position by integration of angular velocities which are collected using gyroscpoe of MPU6050. This will be called gyroscpoe based positions for the rest of the thesis since angular velocities are measured using gyroscope sensor of MPU6050. Since Radius is known, y and z position would be

$$y\ position = r * \sin(angle\ of\ rotation) + initial\ y\ position$$

$$z\ position = r - (r * \cos(angle\ of\ rotation)) + initial\ z\ position$$

While calculatin z position here, i did the substraction $r - (r * \cos(angle\ of\ rotation))$ because center of rotation is not the origin , instead it's location is (0,r) in the y-z plane.

Results of the positions calculated from numerical intergation of acceleration and from gyroscpe data are shown in figure-34 and 35.Also, sensor movements in 3D space can be seen of figure-36 and figure-37.

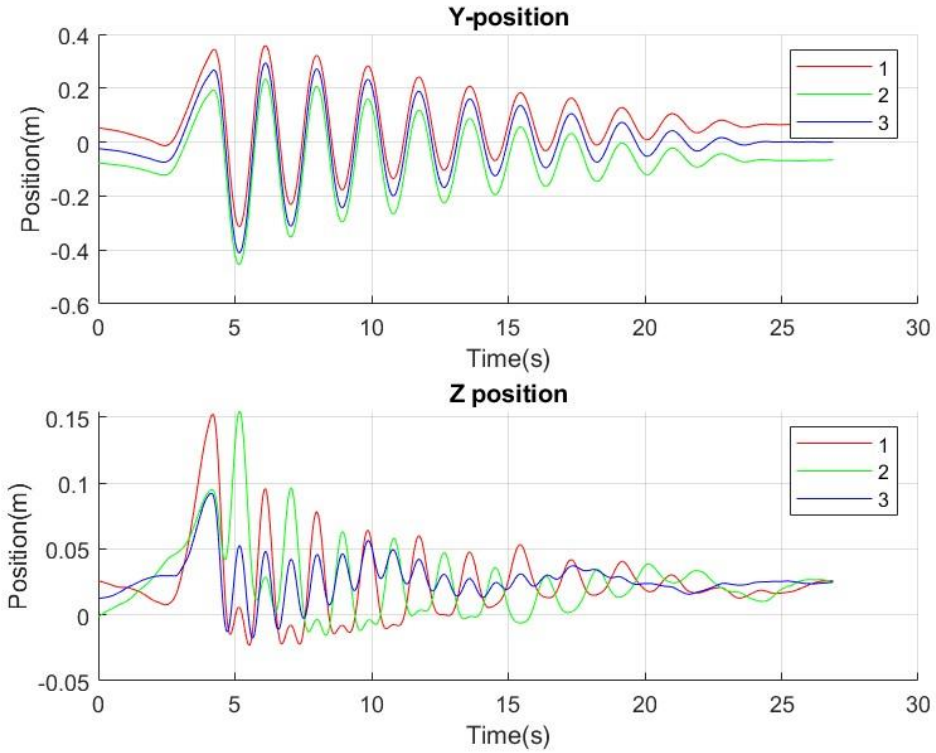

Figure 34:Highpass filter applied position data

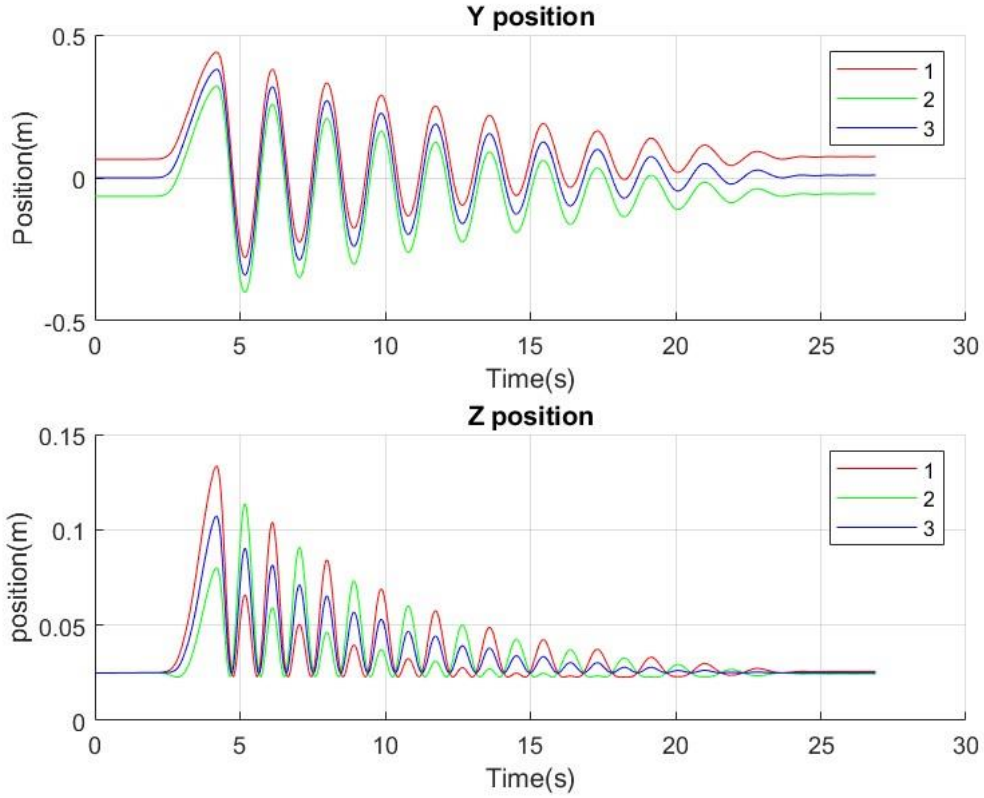

Figure 35:Position data calculted from angular position

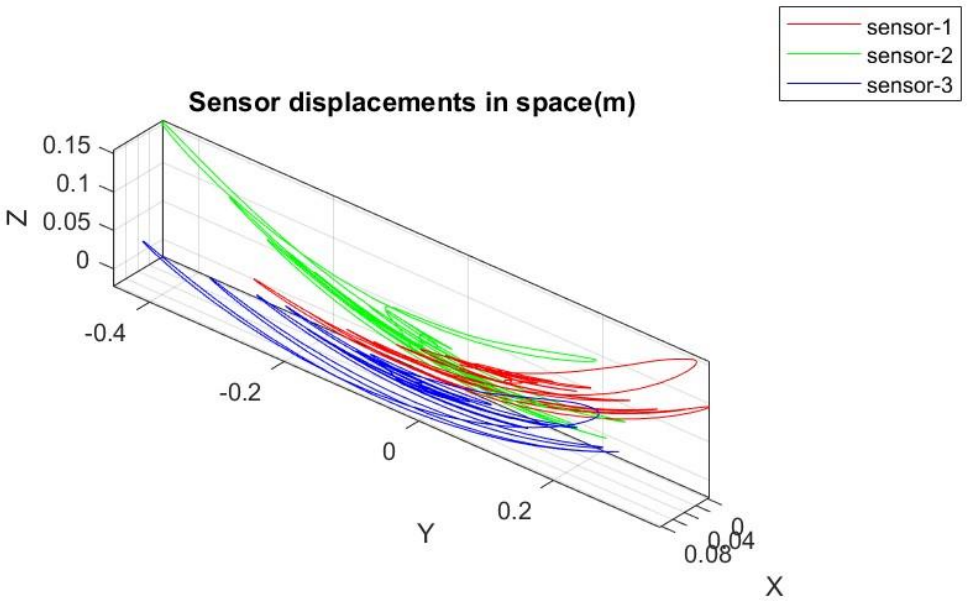

Figure 36:Sensor movements in 3D plane based on accelerometer with application of filters

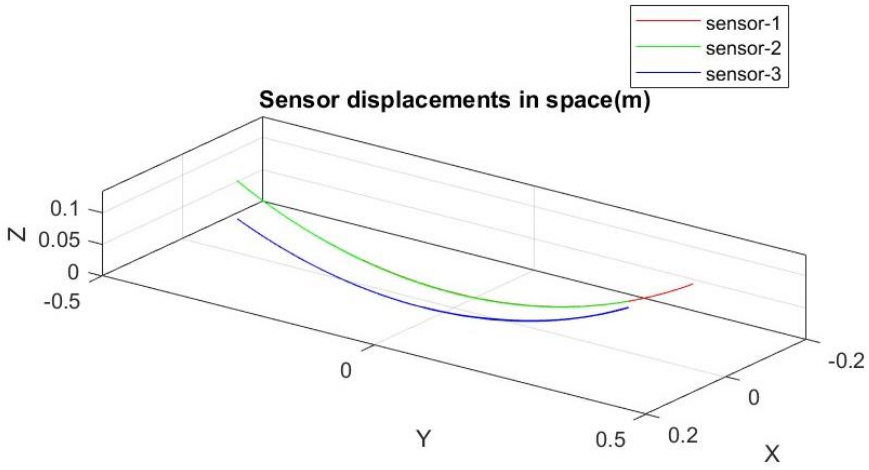

Figure 37:Sensor 3D movements calculated using angular position

It can be observed that position calculated using angular position data and Radius of pendulum, which can only be used in pendulum experiment, are very close to reality, thus will be used to verify position calculated using numerical integration of acceleration.

Differance between the two calculated positions can be seen on figure-38.

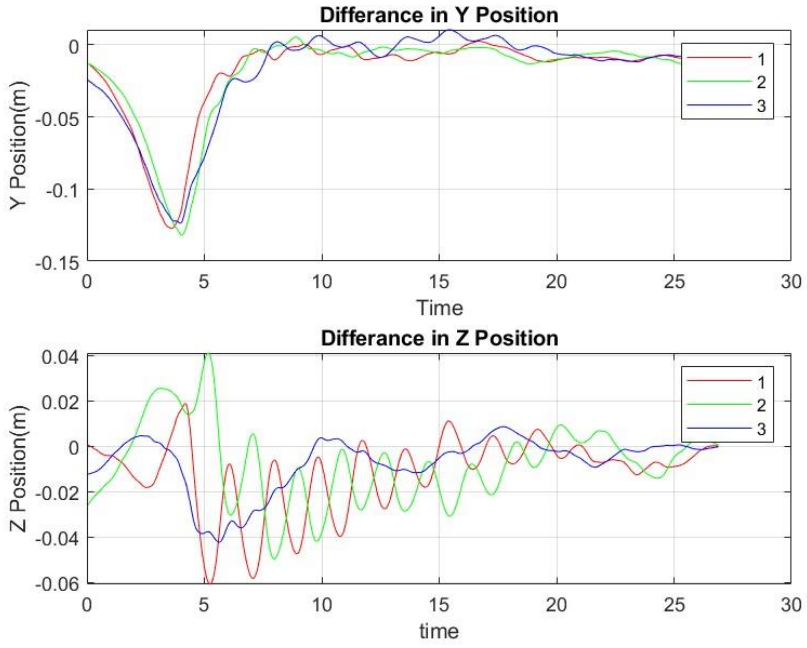

Figure 38:Differance of positions between accelerometer and gyroscope based

Also considering gyroscope based positions are true, error of positions calculated integration of accelerations are also shown dependence of time in figure-39. While calculating error, peek values are purged from error data using a built-in MATLAB function called ‘rmoutliers’. Inside

the function, an outlier defined as a value that is more than three scaled median absolute deviations from the median.

It can be observed that error of calculated postions are quite high. Which in the next step, will affect calculation of center of rotation.

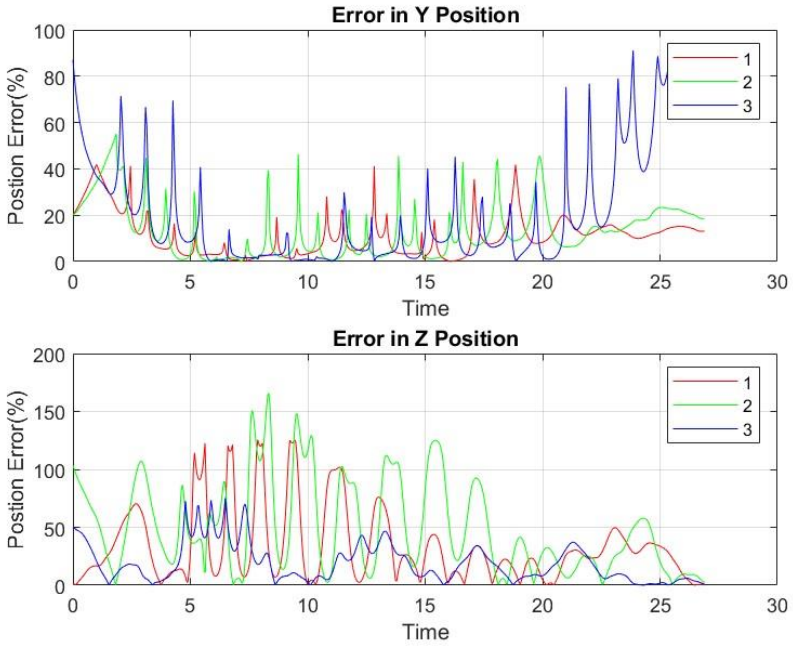

Figure 39:Error of calculated positions

Next step is to calculate center of rotation based on calculated positions. In figure-40 center of rotation is calculated using accelerometer based position for a certain period of time where the rotation movement exists. Center of rotation is calculated for each pair of sensors and mean of pairs are also calculated. Peek values are also purged from the results.

In figure-41 center of rotation is also calculated using gyroscope based position for same period of time. Results this time was very close and also accurate considering real center of rotation and I did not need to purge peek values.

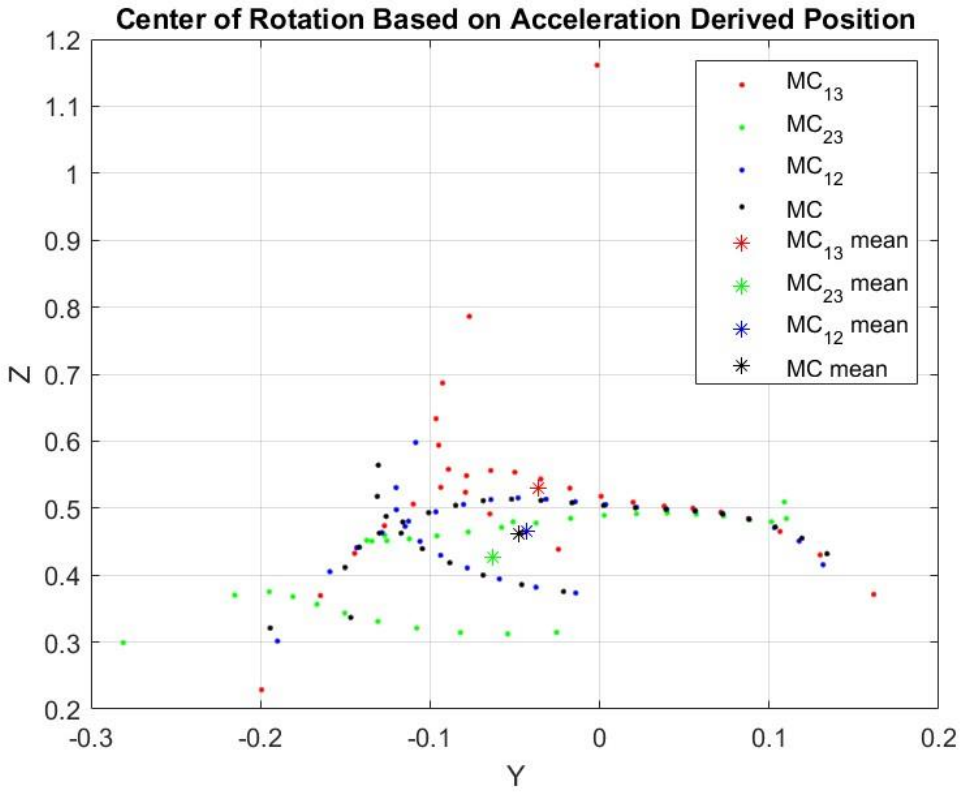

Figure 40:Center of rotation calculated using accelerometer based positions

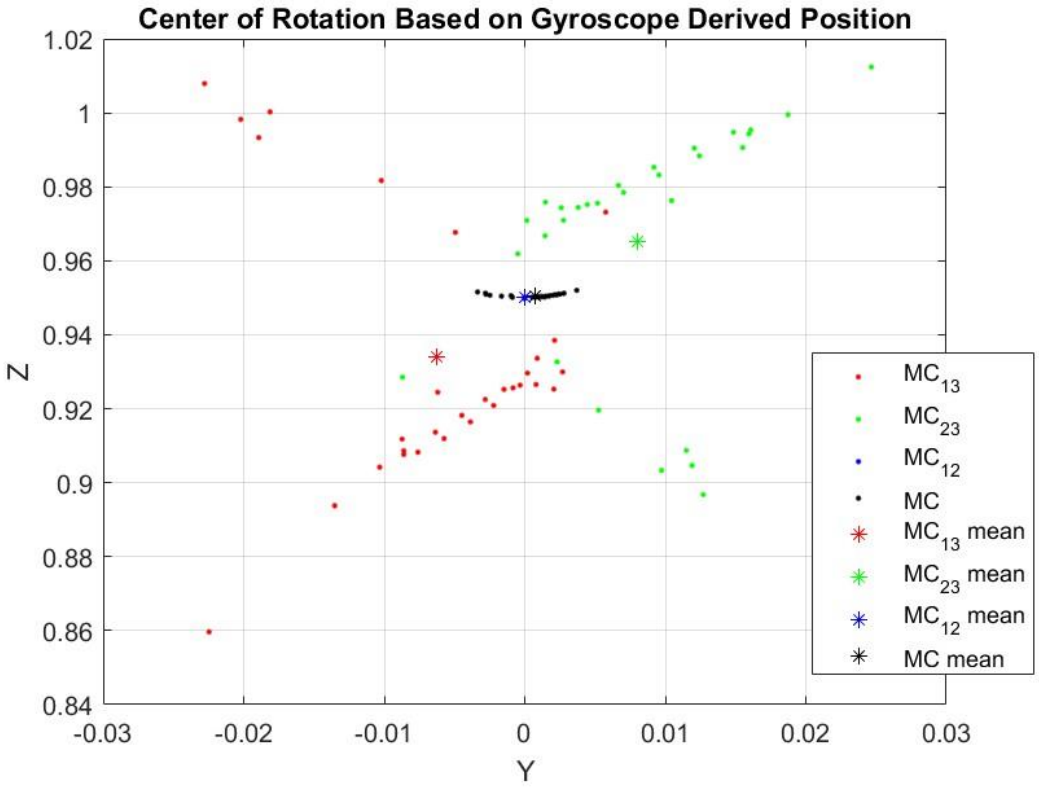

Figure 41:Center of rotation calculated using gyrospcope based positions

Since beginning all the plots was derivated from same measurement of pendulum experiment. Which sensors was mounted on the model ship and model ship was mounted on the tip of pendulum rod. Dimesions of model ship limited me to use distant initial sensor positions and differance in two consecutive sensors was 6.5 cm and can be seen on figure-42 (a) . I re experimented pendulum with larger initial distances which also can be seen on figure- 42 (b).

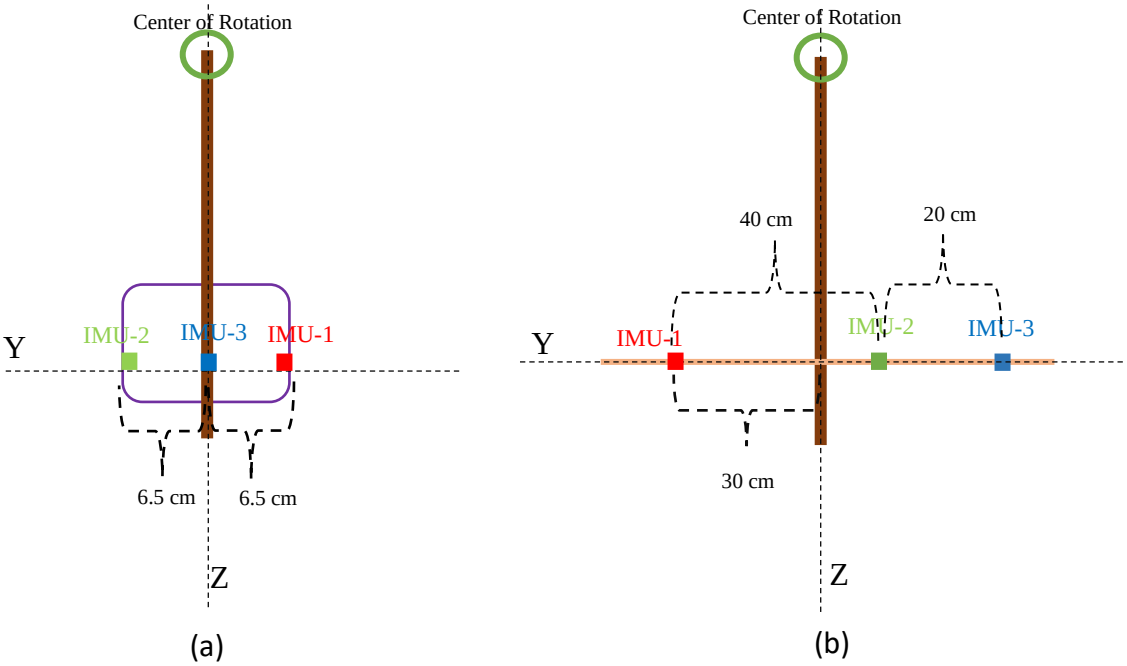

Figure 42:Sensor initial locations used in pendulum experiment

Positions calculated from pendulum experiment with initial sensor locations as figure-42(b) using lowpass filtered acceleration measurements and higpass filtered velocity and position data are plotted on figure-43 and gyroscope based positions are plotted with same initial locations in figure-44. Also in figure-45 and 46 3D postions are shown derived from accelerometer and gyroscope respectively.

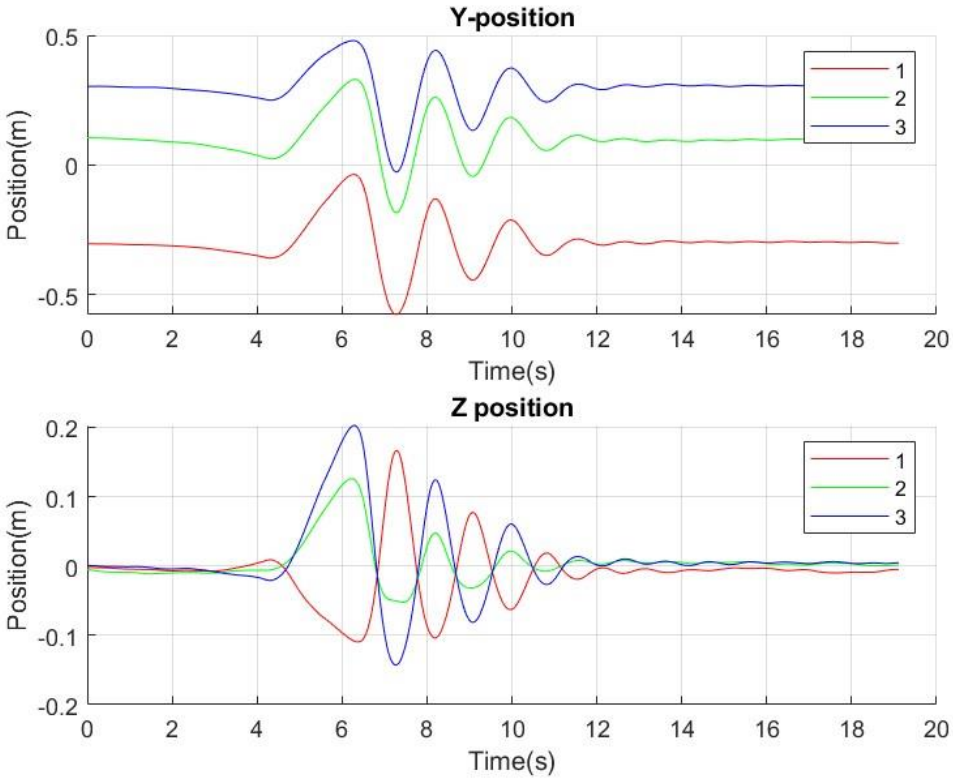

Figure 43:Position results of distant initial location experiment based on accelerometer

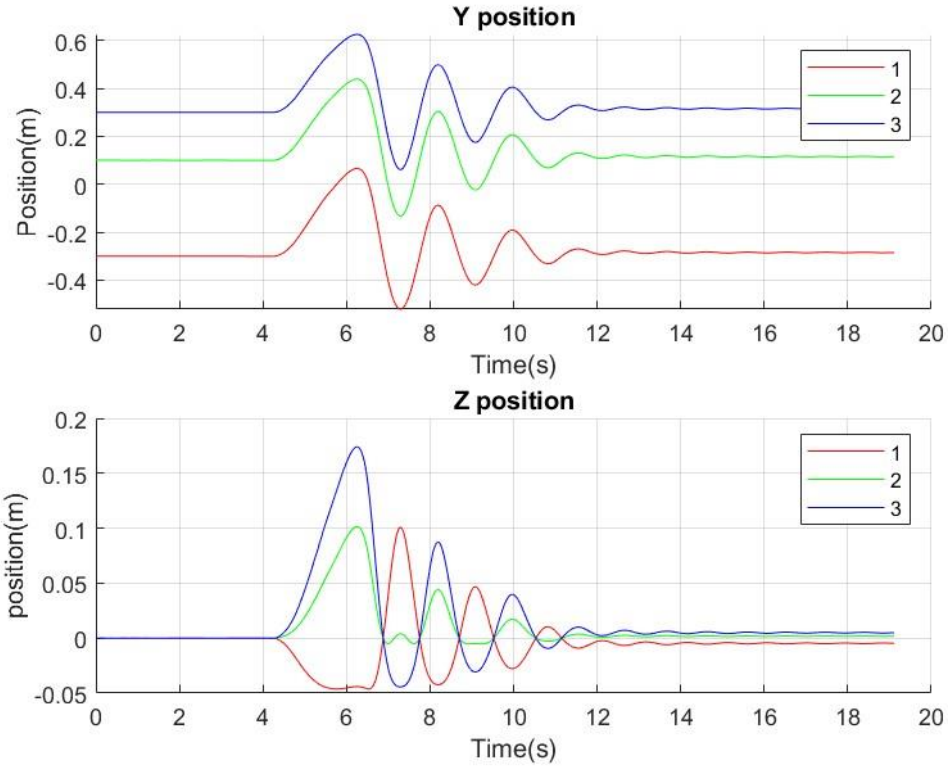

Figure 44:Position results of distant initial location experiment based on gyroscpoe

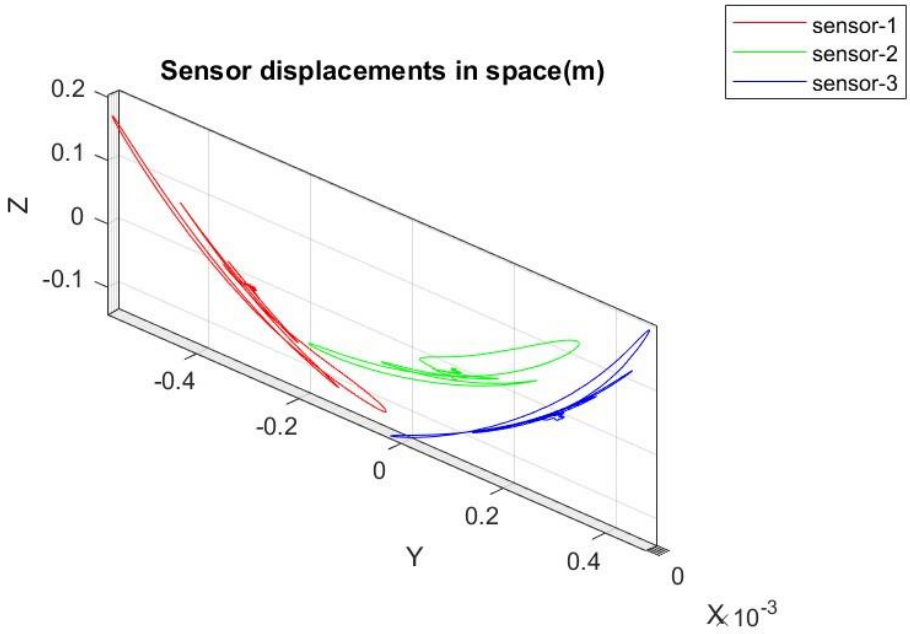

Figure 45:3D sensor movement based on accelerometer

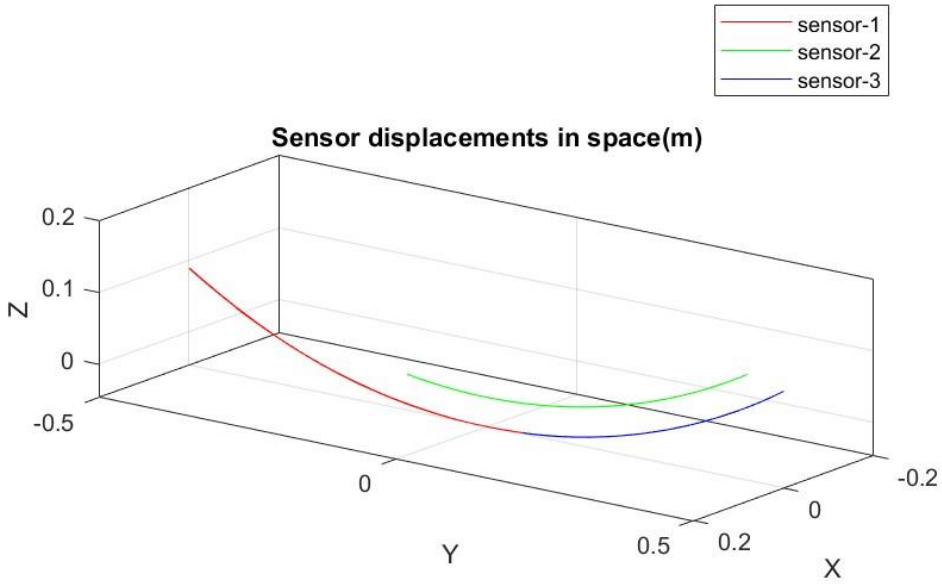

Figure 46:3D sensor movements based on gyroscope

Also differance of two calculated postions and error of calculated positions considering gyroscope based positions are true, also plotted in figure-47 and 48 respectively. Peek values are purged from calculated errors.

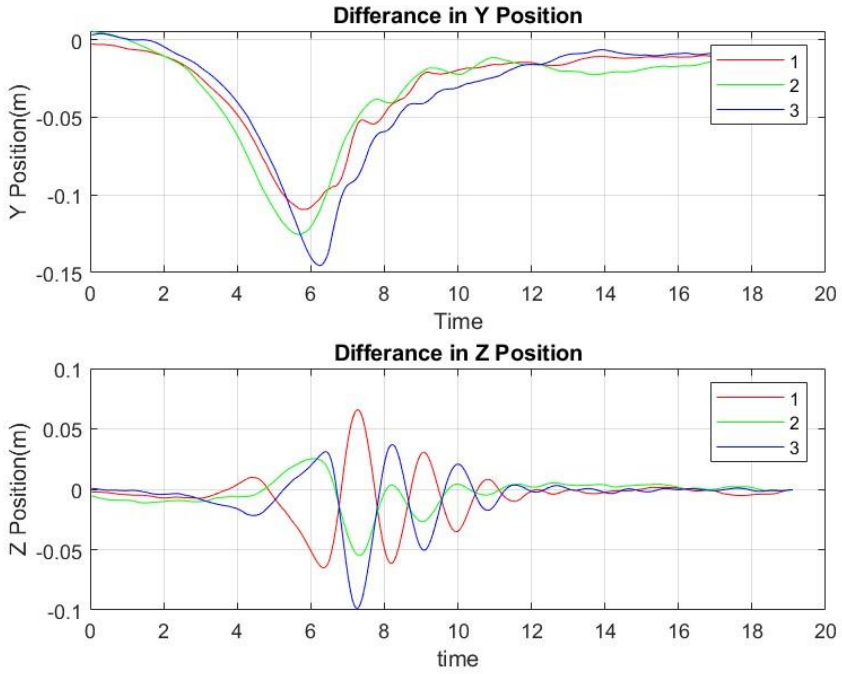

Figure 47:Differance of positions between accelerometer and gyroscope based

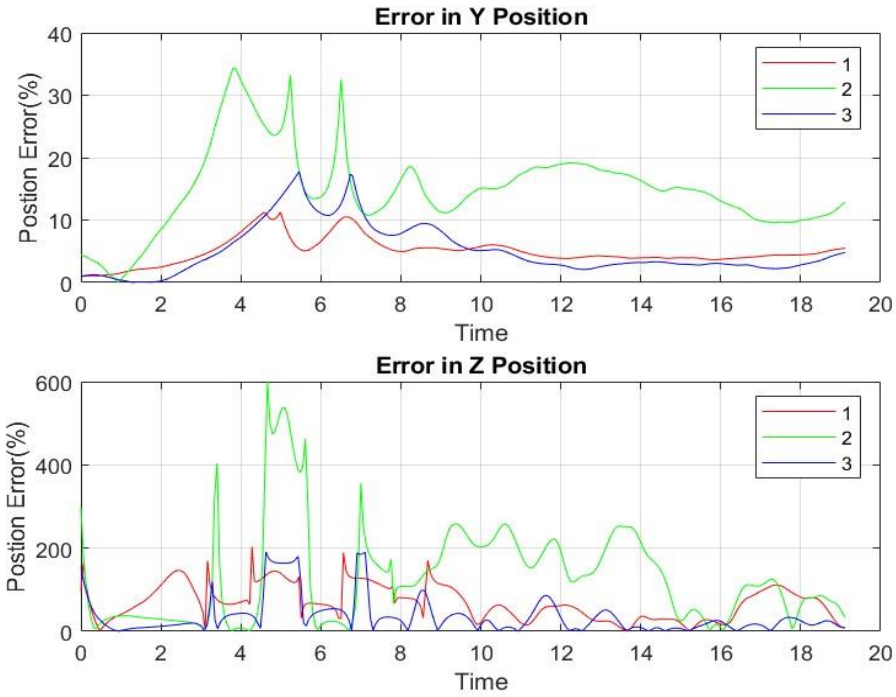

Figure 48:Position errors

Center of rotation also calculated and plotted using accelerometer based postions and gyroscope based positions in figure-49 and 50 respectively.

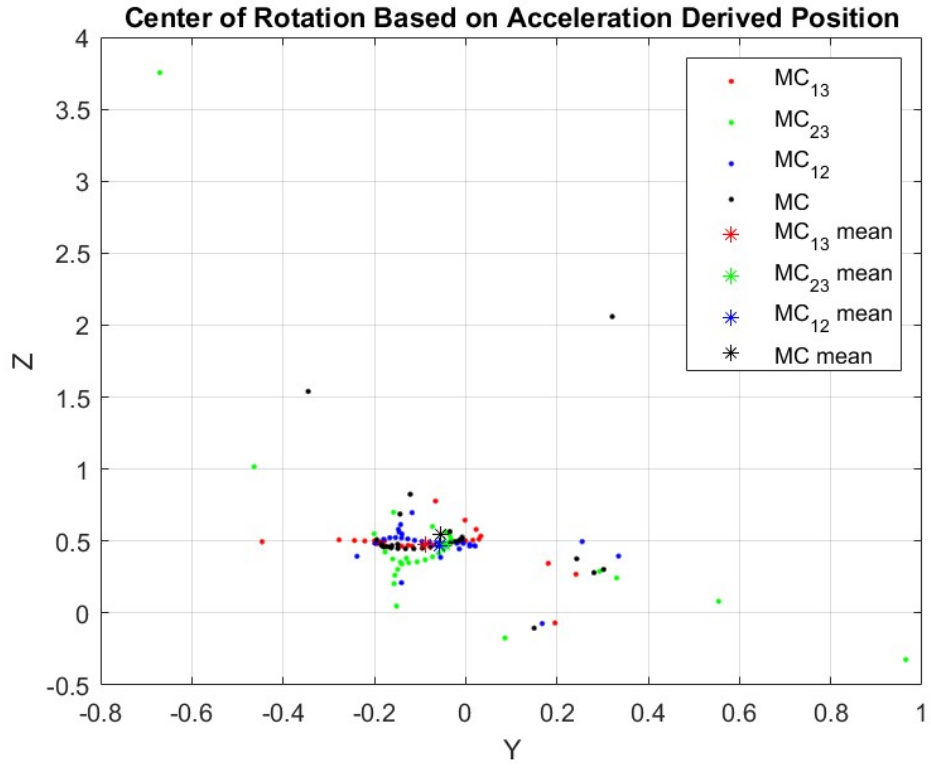

Figure 49:Center of rotation calculated using accelerometer based positions

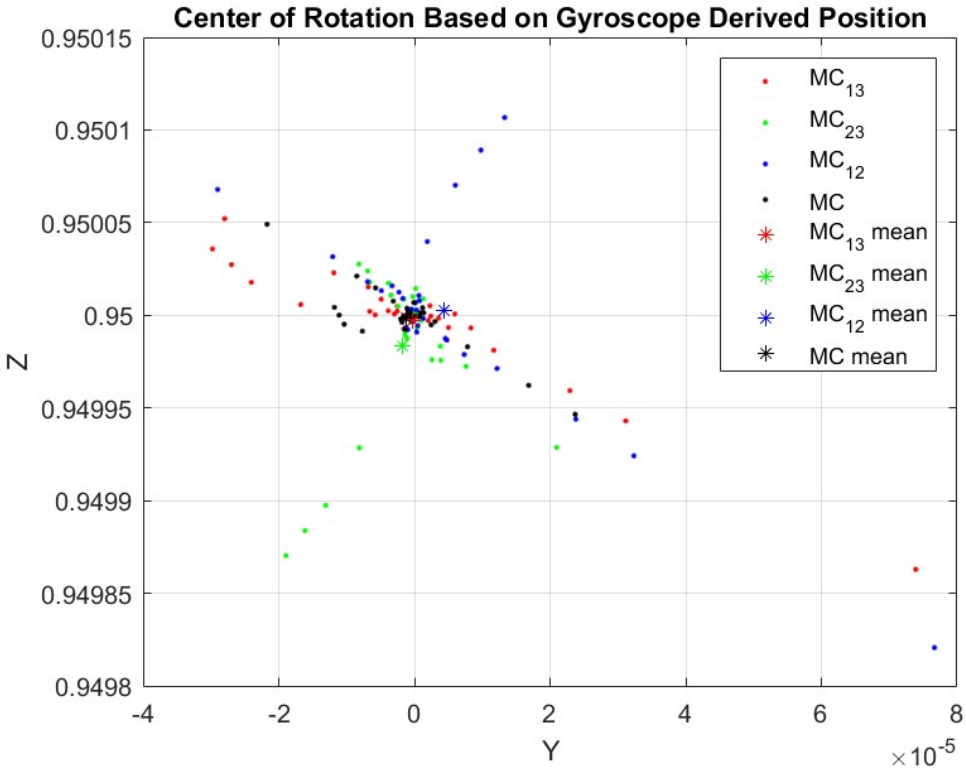

Figure 50:Center of rotation calculated using gyroscope based postions

We can observe that distant initial locations of sensors effected the center of rotation calculation with using gyroscope based positions. Center of rotation results are much closer to the reality and each pair results are also much closer. Because of that, distant intial locations will be used for the rest of the thesis.

3.5 Detrend Function

Detrend is a built-in MATLAB function that removes polynomial trend on a curve.

I tried various order of the polynomials to be removed, best result was 4th order.

For integration, again lowpass filtered acceleration data was used and detrend function only applied to the velocity since this already prevented drift even in the position data without any process after integrating velocity.

Detrended velocity and only integration of detrended velocity and position plots can be seen on figure-51. Accelerometer based positions and gyroscope based positions are in figure-52-53 respectively. Also 3D movement of the sensor's are plotted on figure-54-55 for accelerometer based results and gyroscope based results. I should state detrend function results are shown for distant initial locations of sensors on pendulum and gyroscope based results did not change from the ones calculated when we discuss effects of highpass filtering previous pages.

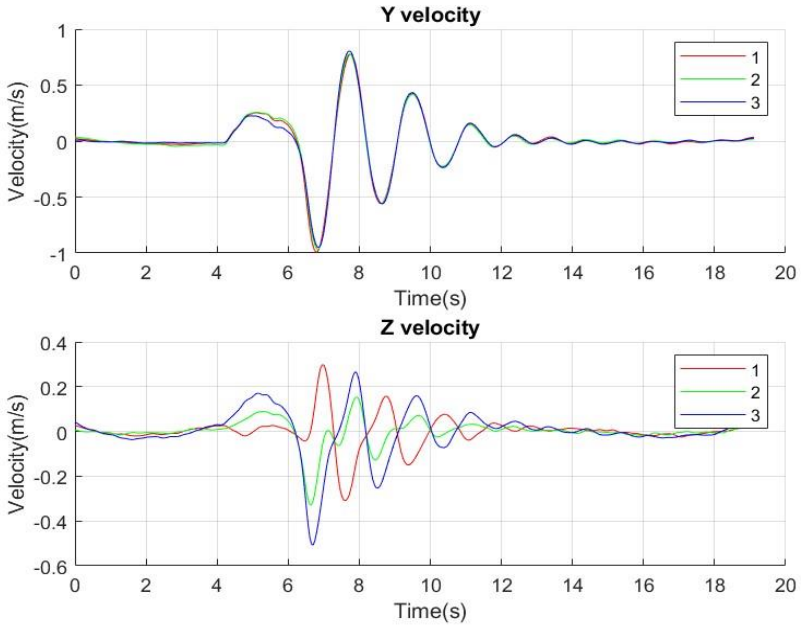

Figure 51:Detrended Velocity data derived from filtered accelerations

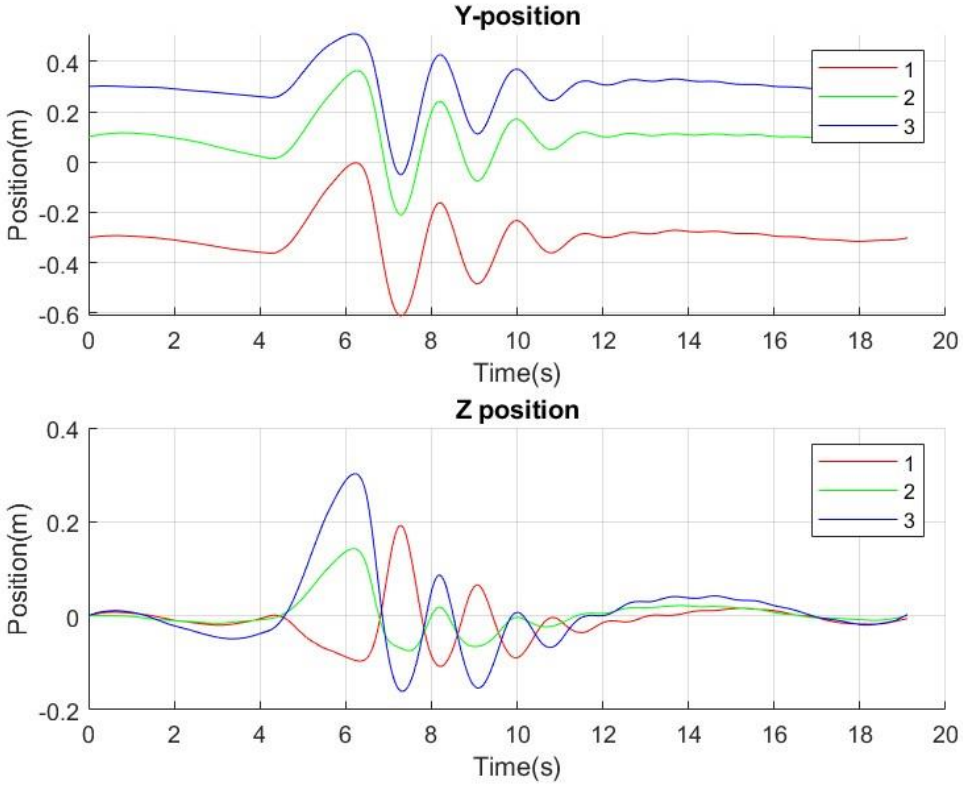

Figure 52:Position data obtained from integration of detrended velocity

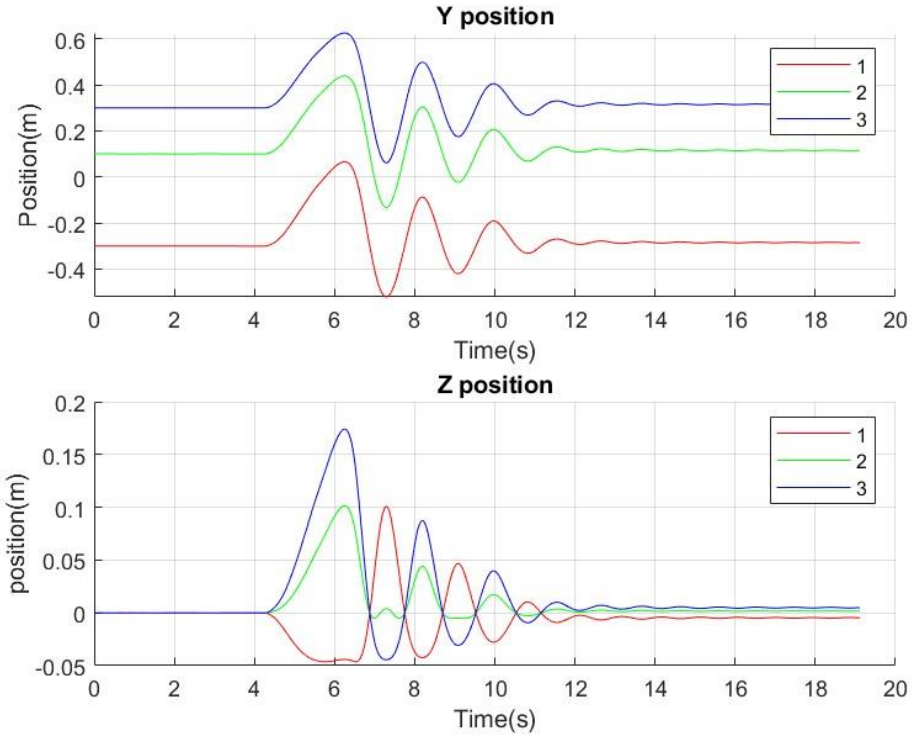

Figure 53:Gyroscope based positions

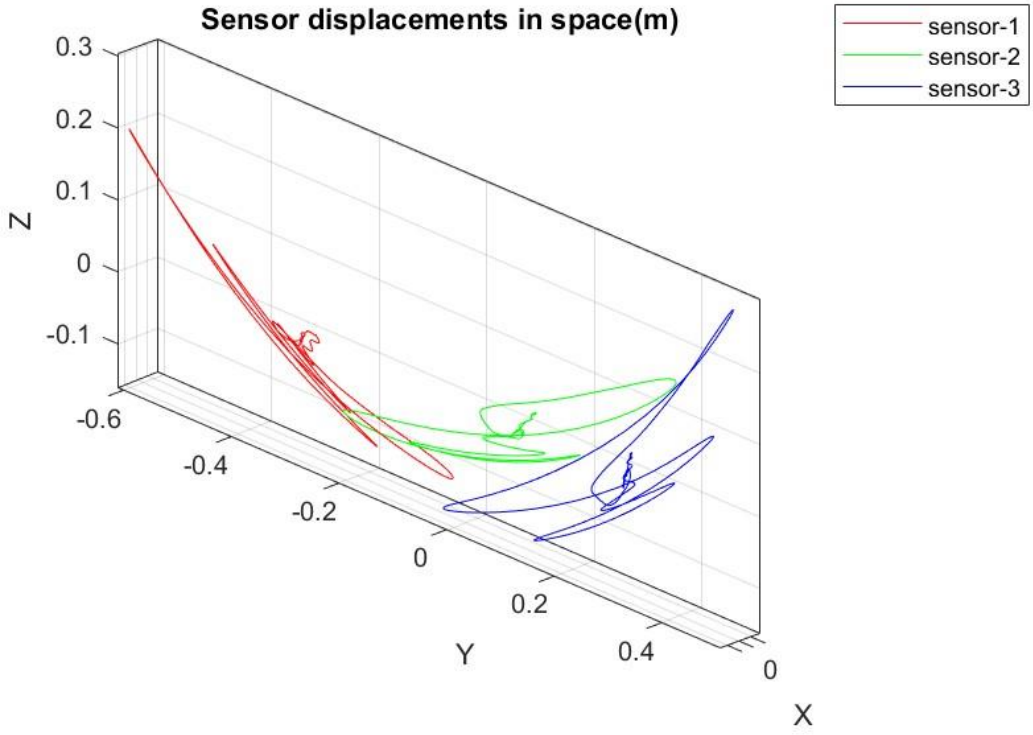

Figure 54:3D positions obtained from integration of detrended velocity

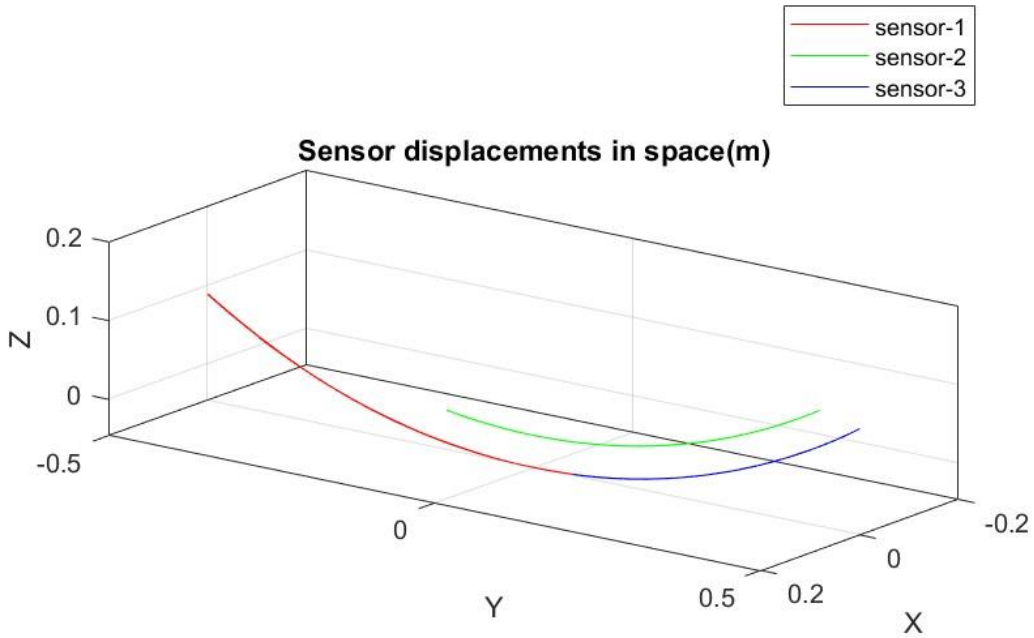

Figure 55:Gyroscope based 3D positions

Differance in positions and considering gyroscope based positions are true, error of position based on accelerometer are also shown in figure-56-57.

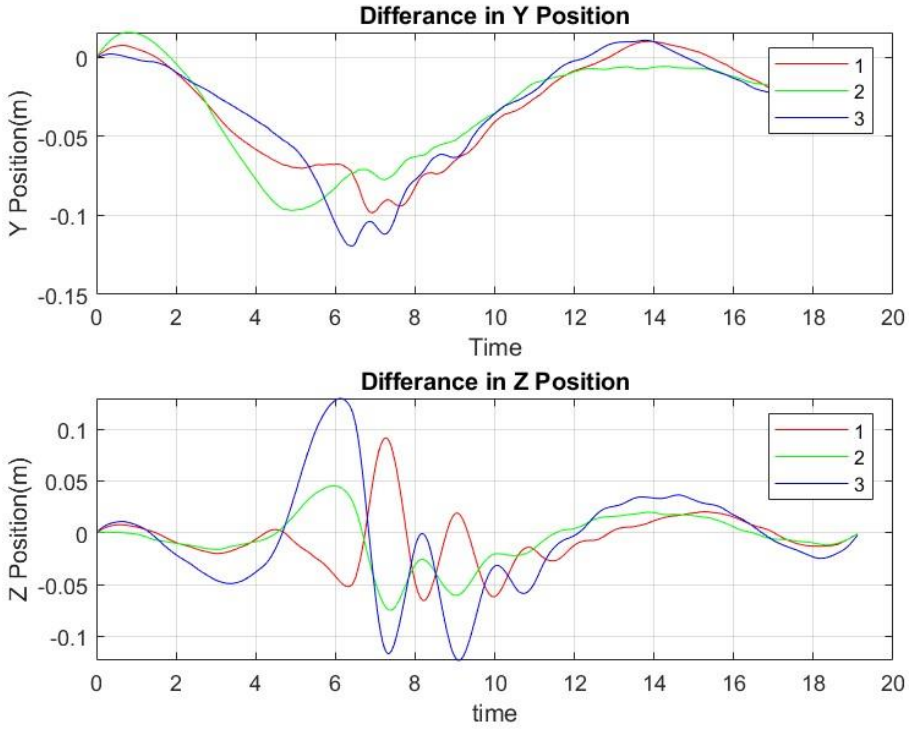

Figure 56:Differance of positions between accelerometer and gyroscope based

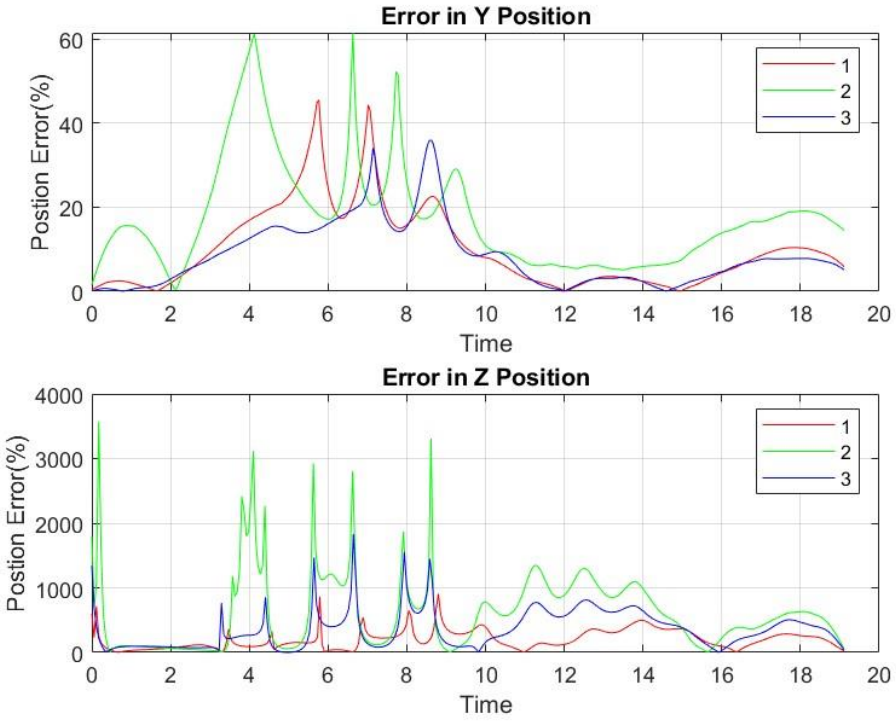

Figure 57:Position errors

Center of rotation is also calculated using accelerometer based positions with the effect of detrending and gyroscope based positions and shown in figure-58-59 respectively.

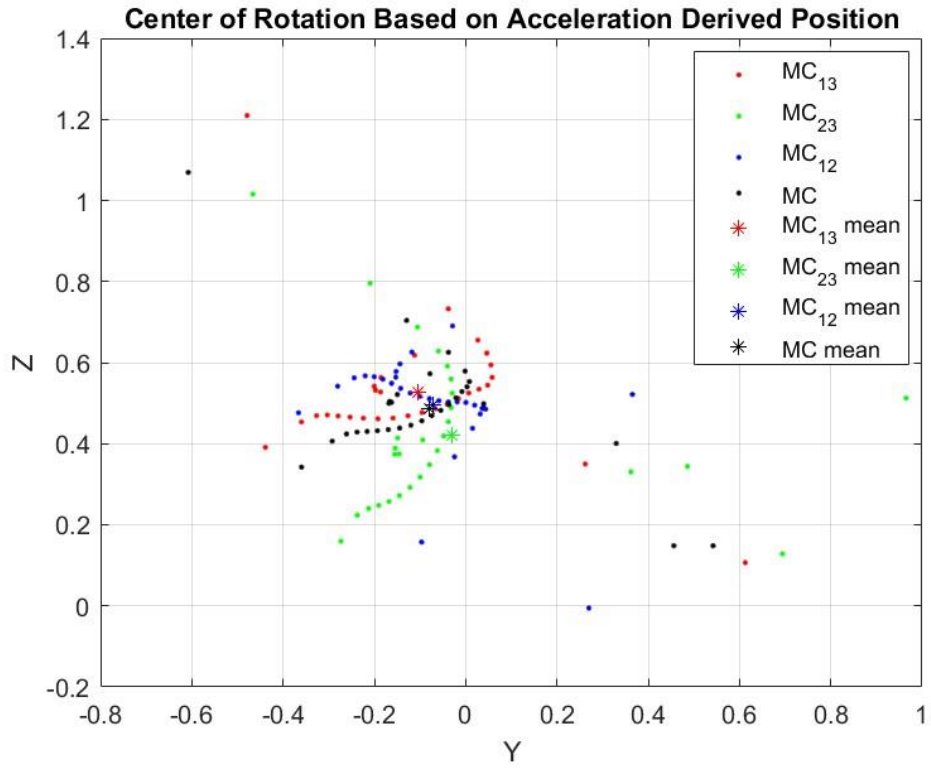

Figure 58:Center of rotation calculated using accelerometer based position

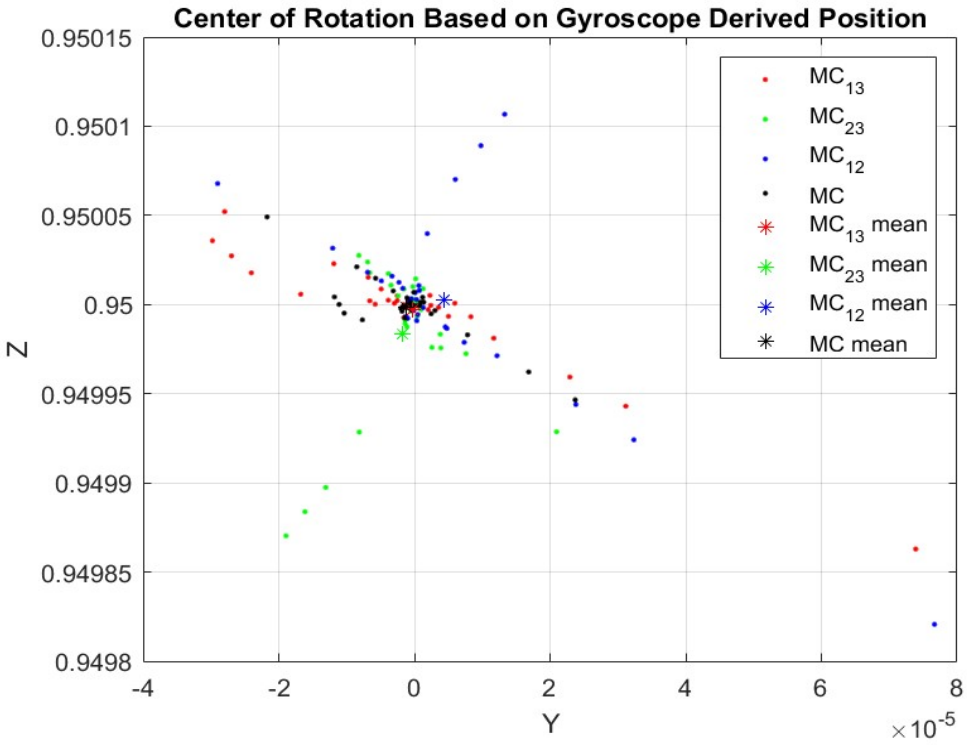

Figure 59:Center of rotation calculated using gyroscope based positions

4. DISCUSSIONS

None of the tried methods was suffiecient enough to track position and use it in order to calculte center of rotation. Methods tried can be classified in 3 groups as: cancelling noise on acceleration, using Highpass filter to integrated data, and removing the deviation using a built-in function called ‘detrend’ on MATLAB.

Cancelling noise has been done using 3 seperated methods, Lowpass filtering, Moving avarage filtering and using Kalman filter as a noice canceller.

All 3 achieved the similar results on cancelling noise on measurements, but none could not achieve the overcome integration drift. Moving avarage filter and Kalman filter can cause more data distortion and phase shift compared to zero phase Butterworth lowpasss filter.

Second method tried was using a Highpass filter to block drifting signals, which accomplished to overcome drift, but still data was distorted and obtained positions differ from reality.

Third method was using ‘detrend’ function to remove polynomial trend on the drifting data. Similar to the Highpass filtering, this method succeded to overcome drift but position data was again distorted.

When I calculte center of rotation based on this distorted data obtained by using Highpass filter or detrending, results was logical but incorrect.

Trying the setup on floating vessel did not needed to happen since we could not get true values of center of rotation.

5. CONCLUSION

In this undergraduate thesis, a method has been experimentally tested in order to determine center fo gravity of a ship. This method uses inertial measurements during the heeling movement of the ship, and tries to determine the angle of heel, metacenter and metacentric height of the ship. Combining these 3 information, center of gravity of the ship can be found.

In order to verify the measurements and calculations, a pre-validation is done by testing the setup on a pendulum. Results show that I could not achieve to track position accurately thus can not calculate center of rotation accurately. Cause of this is a phenomena called integration drift.

Several methods has been tried to overcome the drift, none was sufficent to use and track position accurately. Utilized methods in order to overcome drift could not yield a true positional data, thus center of rotation, even in a considerably smooth movement on a pendulum compared to the heeling of vessel while floating, thus experimenting the setup on floating vessel is not needed.

6. REFERENCES

[1] The works of Archimedes (ed. T.L. Heath, Adamant Media Corporation 2005)

[2] C.B. Barrass, D.R. Derrett, Ship Stability for Masters and Mates (Seventh Edition), 2012, https://doi.org/10.1016/B978-0-08-097093-6.00002-5.

[3] Chhoeung, Sovanna & Hahn, Axel. (2019). Approach to estimate the ship center of gravity based on accelerations and angular velocities without ship parameters. Journal of Physics: Conference Series. 1357. 012028. 10.1088/1742-6596/1357/1/012028.

[4] X. Zhang and S. Jiang, "Application of Fourier Transform and Butterworth Filter in Signal Denoising," 2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP), Xi'an, China, 2021, doi: 10.1109/ICSP51882.2021.9408933.

[5] Ma'arif, Alfian & Iswanto, & Nuryono, Aninditya & Alfian, Rio. (2020). Kalman Filter for Noise Reducer on Sensor Readings. Signal and Image Processing Letters.

[6] R.C. Hibbeler. Fluid Mechanics in SI Units. Pearson Education, ISBN: 9781292247397, 2019

7. APPENDIXES

7.1 MATLAB SCRIPT

```
%%
clc;
clear;
close all;


%% LOADING DIFFERENT MEASUREMENTS

switch 1
    case 1
            load('matlab1.mat')
            t = linspace(0,45.601-26.492,(numel(sens1(:,1))));

            s1x = (0/100).*ones(1,numel(t)); %m
            s1y = (-30/100).*ones(1,numel(t)); %m
            s1z = (0/100).*ones(1,numel(t)); %m

            s2x = (0/100).*ones(1,numel(t)); %m
            s2y = (10/100).*ones(1,numel(t)); %m
            s2z = (0/100).*ones(1,numel(t)); %m

            s3x = (0/100).*ones(1,numel(t)); %m
            s3y = (30/100).*ones(1,numel(t)); %m
            s3z = (0/100).*ones(1,numel(t)); %m
    case 2
            load('matlab2.mat')
            t = linspace(0,37.264-20.864,(numel(sens1(:,1))));

            s1x = (0/100).*ones(1,numel(t)); %m
            s1y = (-30/100).*ones(1,numel(t)); %m
            s1z = (0/100).*ones(1,numel(t)); %m

            s2x = (0/100).*ones(1,numel(t)); %m
            s2y = (10/100).*ones(1,numel(t)); %m
            s2z = (0/100).*ones(1,numel(t)); %m

            s3x = (0/100).*ones(1,numel(t)); %m
            s3y = (30/100).*ones(1,numel(t)); %m
            s3z = (0/100).*ones(1,numel(t)); %m
    case 3
            load('matlab3.mat')
            t = linspace(0,33.450-14.669,(numel(sens1(:,1))));

            s1x = (0/100).*ones(1,numel(t)); %m
            s1y = (-30/100).*ones(1,numel(t)); %m
            s1z = (0/100).*ones(1,numel(t)); %m

            s2x = (0/100).*ones(1,numel(t)); %m
            s2y = (10/100).*ones(1,numel(t)); %m
            s2z = (0/100).*ones(1,numel(t)); %m

            s3x = (0/100).*ones(1,numel(t)); %m
            s3y = (30/100).*ones(1,numel(t)); %m
            s3z = (0/100).*ones(1,numel(t)); %m
     case 4
            load('matlab11.mat')
            t = linspace(0,75.222-48.334,(numel(sens1(:,1))));

            s1x = (0/100).*ones(1,numel(t)); %m
            s1y = (6.5/100).*ones(1,numel(t)); %m
            s1z = (2.5/100).*ones(1,numel(t)); %m

            s2x = (0/100).*ones(1,numel(t)); %m
            s2y = (-6.5/100).*ones(1,numel(t)); %m
            s2z = (2.5/100).*ones(1,numel(t)); %m

            s3x = (8/100).*ones(1,numel(t)); %m
            s3y = (0/100).*ones(1,numel(t)); %m
            s3z = (2.5/100).*ones(1,numel(t)); %m

    case 5
            load('matlab22.mat')
            t = linspace(0,74.627-48.236,(numel(sens1(:,1))));

            s1x = (8/100).*ones(1,numel(t)); %m
            s1y = (0/100).*ones(1,numel(t)); %m
            s1z = (2.5/100).*ones(1,numel(t)); %m

            s2x = (0/100).*ones(1,numel(t)); %m
            s2y = (-6.5/100).*ones(1,numel(t)); %m
            s2z = (2.5/100).*ones(1,numel(t)); %m

            s3x = (0/100).*ones(1,numel(t)); %m
            s3y = (6.5/100).*ones(1,numel(t)); %m
```

```
            s3z = (2.5/100).*ones(1,numel(t)); %m
    case 6
            load('matlab33.mat')
            t = linspace(0,33.358-4.996,(numel(sens1(:,1))));

            s1x = (0/100).*ones(1,numel(t)); %m
            s1y = (-6.5/100).*ones(1,numel(t)); %m
            s1z = (2.5/100).*ones(1,numel(t)); %m

            s2x = (8/100).*ones(1,numel(t)); %m
            s2y = (0/100).*ones(1,numel(t)); %m
            s2z = (2.5/100).*ones(1,numel(t)); %m

            s3x = (0/100).*ones(1,numel(t)); %m
            s3y = (6.5/100).*ones(1,numel(t)); %m
            s3z = (2.5/100).*ones(1,numel(t)); %m
    case 7
        load('matlab44.mat')
            t = linspace(0,35.656-3.251,(numel(sens1(:,1))));


            s1x = (0/100).*ones(1,numel(t)); %m
            s1y = (-6.5/100).*ones(1,numel(t)); %m
            s1z = (2.5/100).*ones(1,numel(t)); %m

            s2x = (0/100).*ones(1,numel(t)); %m
            s2y = (0/100).*ones(1,numel(t)); %m
            s2z = (6/100).*ones(1,numel(t)); %m

            s3x = (0/100).*ones(1,numel(t)); %m
            s3y = (6.5/100).*ones(1,numel(t)); %m
            s3z = (2.5/100).*ones(1,numel(t)); %m
    case 8
        load('matlab55.mat')
            t = linspace(0,72.541-44.66,(numel(sens1(:,1))));

            s1x = (0/100).*ones(1,numel(t)); %m
            s1y = (-6.5/100).*ones(1,numel(t)); %m
            s1z = (2.5/100).*ones(1,numel(t)); %m

            s2x = (8/100).*ones(1,numel(t)); %m
            s2y = (0/100).*ones(1,numel(t)); %m
            s2z = (2.5/100).*ones(1,numel(t)); %m

            s3x = (0/100).*ones(1,numel(t)); %m
            s3y = (0/100).*ones(1,numel(t)); %m
            s3z = (6/100).*ones(1,numel(t)); %m

end



b=18/100; %m
g=9.81;
Fs = numel(t)/t(end);
L = numel(t);
f = Fs*(0:(L/2))/L;
samplePeriod = 1/Fs;

%% Substracting offsets
wx1 = sens1(:,1);
wx1= wx1 - mean(wx1(1:30));
wx1 = wx1.';
wy1 = sens1(:,2);
wy1= wy1 - mean(wy1(1:30));
wy1 = wy1.';
wz1 = sens1(:,3);
wz1= wz1 - mean(wz1(1:30));
wz1 = wz1.';
x1 = sens1(:,4);
x1= x1 - mean(x1(1:30));
x1 = x1.';
y1 = sens1(:,5);
y1= y1 - mean(y1(1:30));
y1 = y1.';
z1 = sens1(:,6);
z1= z1 - (mean(z1(1:30)-g));
z1 = z1.';

wx2 = sens2(:,1);
wx2= wx2 - mean(wx2(1:30));
wx2 = wx2.';
wy2 = sens2(:,2);
wy2= wy2 - mean(wy2(1:30));
wy2 = wy2.';
wz2 = sens2(:,3);
wz2= wz2 - mean(wz2(1:30));
wz2 = wz2.';
x2 = sens2(:,4);
x2= x2 - mean(x2(1:30));
x2 = x2.';
y2 = sens2(:,5);
y2= y2 - mean(y2(1:30));
y2 = y2.';
z2 = sens2(:,6);
z2= z2 - (mean(z2(1:30)-g));
z2 = z2.';

wx3 = sens3(:,1);
```

```
wx3= wx3 - mean(wx3(1:30));
wx3 = wx3.';
wy3 = sens3(:,2);
wy3= wy3 - mean(wy3(1:30));
wy3 = wy3.';
wz3 = sens3(:,3);
wz3= wz3 - mean(wz3(1:30));
wz3 = wz3.';
x3 = sens3(:,4);
x3= x3 - mean(x3(1:30));
x3 = x3.';
y3 = sens3(:,5);
y3= y3 - mean(y3(1:30));
y3 = y3.';
z3 = sens3(:,6);
z3= z3 - (mean(z3(1:30)-g));
z3 = z3.';

%%  angular velocity-position
w_x_1 = wx1;
w_y_1 = wy1;
w_z_1 = wz1;

w_x_2 = wx2;
w_y_2 = wy2;
w_z_2 = wz2;

w_x_3 = wx3;
w_y_3 = wy3;
w_z_3 = wz3;



figure(1)
hold on
plot(t,w_x_1,'r')
plot(t,w_x_2,'g')
plot(t,w_x_3,'b')
xlabel('Time(s)')
ylabel('Angular Velocity (deg/s)')
title('Angular Velocity X Axis')
legend('1','2','3')
grid on
hold off



figure(222)
hold on
plot(t,w_y_1,'r')
plot(t,w_y_2,'g')
plot(t,w_y_3,'b')
xlabel('time(s)')
ylabel('Angular Velocity Y Axis(deg/s)')
legend('1','2','3')
grid on
hold off

figure(333)
hold on
plot(t,w_z_1,'r')
plot(t,w_z_2,'g')
plot(t,w_z_3,'b')
xlabel('time(s)')
ylabel('Angular Velocity Z Axis(deg/s)')
legend('1','2','3')
grid on
hold off


sdeg_sensor_1_x = 0+(cumtrapz(t,w_x_1));
sdeg_sensor_2_x = 0+(cumtrapz(t,w_x_2));
sdeg_sensor_3_x = 0+(cumtrapz(t,w_x_3));

sdeg_sensor_1_y = detrend(0+(cumtrapz(t,w_y_1)),4);
sdeg_sensor_2_y = detrend(0+(cumtrapz(t,w_y_2)),4);
sdeg_sensor_3_y = detrend(0+(cumtrapz(t,w_y_3)),4);

sdeg_sensor_1_z = detrend(0+(cumtrapz(t,w_z_1)),4);
sdeg_sensor_2_z = detrend(0+(cumtrapz(t,w_z_2)),4);
sdeg_sensor_3_z = detrend(0+(cumtrapz(t,w_z_3)),4);


sdeg_x = (sdeg_sensor_1_x + sdeg_sensor_2_x +sdeg_sensor_3_x)./3;
sdeg_y = (sdeg_sensor_1_y + sdeg_sensor_2_y +sdeg_sensor_3_y)./3;
sdeg_z = (sdeg_sensor_1_z + sdeg_sensor_2_z +sdeg_sensor_3_z)./3;

figure(2)
hold on
plot(t,sdeg_sensor_1_x,'r')
plot(t,sdeg_sensor_2_x,'g')
plot(t,sdeg_sensor_3_x,'b')
plot(t,sdeg_x,'k')
legend('1','2','3','mean')
xlabel("Time(s)")
ylabel("Angle of Rotation(Deg)")
title("Angle of Rotation Around X Axis")
grid on
```

```
hold off

figure(2222)
hold on
plot(t,sdeg_sensor_1_y,'r')
plot(t,sdeg_sensor_2_y,'g')
plot(t,sdeg_sensor_3_y,'b')
plot(t,sdeg_y,'k')
legend('1','2','3','mean')
xlabel("Time(s)")
ylabel("Angle of Rotation(Deg)")
title("Angle of Rotation Around Y Axis")
grid on
hold off

figure(3333)
hold on
plot(t,sdeg_sensor_1_z,'r')
plot(t,sdeg_sensor_2_z,'g')
plot(t,sdeg_sensor_3_z,'b')
plot(t,sdeg_z,'k')
legend('1','2','3','mean')
xlabel("Time(s)")
ylabel("Angle of Rotation(Deg)")
title("Angle of Rotation Around Z Axis")
grid on
hold off
%% ACCELERATION DATA PROCESSING

%apply rotation matrix around x axis
a = numel(t);
acc_sensor_1=zeros(a,3);
acc_sensor_2=zeros(a,3);
acc_sensor_3=zeros(a,3);
s1_in=zeros(a,3);
s2_in=zeros(a,3);
s3_in=zeros(a,3);
for i=1:a
  rot_mat_x = [1 0 0;0 cosd(-sdeg_x(i)) -sind(-sdeg_x(i));0 sind(-sdeg_x(i)) cosd(-sdeg_x(i))];
  acc_sensor_1(i,:) = [x1(i) y1(i) z1(i)]*rot_mat_x;
  acc_sensor_2(i,:) = [x2(i) y2(i) z2(i)]*rot_mat_x;
  acc_sensor_3(i,:) = [x3(i) y3(i) z3(i)]*rot_mat_x;
  s1_in(i,:) = [s1x(i),s1y(i),s1z(i)]*rot_mat_x;
  s2_in(i,:) = [s2x(i),s2y(i),s2z(i)]*rot_mat_x;
  s3_in(i,:) = [s3x(i),s3y(i),s3z(i)]*rot_mat_x;

end

%apply rotation matrix around y axis
x1 = acc_sensor_1(:,1);
y1 = acc_sensor_1(:,2);
z1 = acc_sensor_1(:,3);
x2 = acc_sensor_2(:,1);
y2 = acc_sensor_2(:,2);
z2 = acc_sensor_2(:,3);
x3 = acc_sensor_3(:,1);
y3 = acc_sensor_3(:,2);
z3 = acc_sensor_3(:,3);
s1x = s1_in(:,1);
s1y = s1_in(:,2);
s1z = s1_in(:,3);
s2x = s2_in(:,1);
s2y = s2_in(:,2);
s2z = s2_in(:,3);
s3x = s3_in(:,1);
s3y = s3_in(:,2);
s3z = s3_in(:,3);


a = numel(t);
acc_sensor_1=zeros(a,3);
acc_sensor_2=zeros(a,3);
acc_sensor_3=zeros(a,3);
s1_in=zeros(a,3);
s2_in=zeros(a,3);
s3_in=zeros(a,3);
for i=1:a
  rot_mat_y = [cosd(-sdeg_y(i)) 0 sind(-sdeg_y(i));0 1 0;-sind(-sdeg_y(i)) 0 cosd(-sdeg_y(i))];

  acc_sensor_1(i,:) = [x1(i) y1(i) z1(i)]*rot_mat_y;
  acc_sensor_2(i,:) = [x2(i) y2(i) z2(i)]*rot_mat_y;
  acc_sensor_3(i,:) = [x3(i) y3(i) z3(i)]*rot_mat_y;
  s1_in(i,:) = [s1x(i),s1y(i),s1z(i)]*rot_mat_y;
  s2_in(i,:) = [s2x(i),s2y(i),s2z(i)]*rot_mat_y;
  s3_in(i,:) = [s3x(i),s3y(i),s3z(i)]*rot_mat_y;

end

%apply rotation matrix around z axis
x1 = acc_sensor_1(:,1);
y1 = acc_sensor_1(:,2);
z1 = acc_sensor_1(:,3);
x2 = acc_sensor_2(:,1);
y2 = acc_sensor_2(:,2);
z2 = acc_sensor_2(:,3);
x3 = acc_sensor_3(:,1);
y3 = acc_sensor_3(:,2);
z3 = acc_sensor_3(:,3);
s1x = s1_in(:,1);
```

```
s1y = s1_in(:,2);
s1z = s1_in(:,3);
s2x = s2_in(:,1);
s2y = s2_in(:,2);
s2z = s2_in(:,3);
s3x = s3_in(:,1);
s3y = s3_in(:,2);
s3z = s3_in(:,3);

a = numel(t);
acc_sensor_1=zeros(a,3);
acc_sensor_2=zeros(a,3);
acc_sensor_3=zeros(a,3);
s1_in=zeros(a,3);
s2_in=zeros(a,3);
s3_in=zeros(a,3);
for i=1:a
  rot_mat_z = [cosd(-sdeg_z(i)) -sind(-sdeg_y(i)) 0;sind(-sdeg_y(i)) cosd(-sdeg_y(i)) 0;0 0 1];

  acc_sensor_1(i,:) = [x1(i) y1(i) z1(i)]*rot_mat_z;
  acc_sensor_2(i,:) = [x2(i) y2(i) z2(i)]*rot_mat_z;
  acc_sensor_3(i,:) = [x3(i) y3(i) z3(i)]*rot_mat_z;
  s1_in(i,:) = [s1x(i),s1y(i),s1z(i)]*rot_mat_z;
  s2_in(i,:) = [s2x(i),s2y(i),s2z(i)]*rot_mat_z;
  s3_in(i,:) = [s3x(i),s3y(i),s3z(i)]*rot_mat_z;

end


acc_sensor_1 = acc_sensor_1.';
acc_sensor_2 = acc_sensor_2.';
acc_sensor_3 = acc_sensor_3.';
s1_in=s1_in.';
s2_in=s2_in.';
s3_in=s3_in.';


acc_sensor_1(3,:) = acc_sensor_1(3,:) - g;
acc_sensor_2(3,:) = acc_sensor_2(3,:) - g;
acc_sensor_3(3,:) = acc_sensor_3(3,:) - g;

%% SMOOTHDATA

a_sensor_1(1,:) = smoothdata(acc_sensor_1(1,:));
a_sensor_1(2,:) = smoothdata(acc_sensor_1(2,:));
a_sensor_1(3,:) = smoothdata(acc_sensor_1(3,:));

a_sensor_2(1,:) = smoothdata(acc_sensor_2(1,:));
a_sensor_2(2,:) = smoothdata(acc_sensor_2(2,:));
a_sensor_2(3,:) = smoothdata(acc_sensor_2(3,:));

a_sensor_3(1,:) = smoothdata(acc_sensor_3(1,:));
a_sensor_3(2,:) = smoothdata(acc_sensor_3(2,:));
a_sensor_3(3,:) = smoothdata(acc_sensor_3(3,:));

%% fft to acc data
%SENSOR-1
ax1 = fft(acc_sensor_1(1,:));
P2 = abs(ax1/L);
P1x = P2(1:L/2+1);
P1x(2:end-1) = 2*P1x(2:end-1);

ay1 = fft(acc_sensor_1(2,:));
P2 = abs(ay1/L);
P1y = P2(1:L/2+1);
P1y(2:end-1) = 2*P1y(2:end-1);

az1 = fft(acc_sensor_1(3,:));
P2 = abs(az1/L);
P1z = P2(1:L/2+1);
P1z(2:end-1) = 2*P1z(2:end-1);

figure(91)
% subplot(3,1,1)
% plot(f,P1x)
% title('Power Spectrum of acc sensor 1 x ')
% xlabel('f (Hz)')
% ylabel('|P1(f)|')

subplot(2,1,1)
% figure(92)
plot(f,P1y)
title('Power Spectrum of sensor 1 y acc ')
xlabel('f (Hz)')
ylabel('|P1(f)|')

subplot(2,1,2)
% figure(93)
plot(f,P1z)
title('Power Spectrum of sensor 1 z acc ')
xlabel('f (Hz)')
ylabel('|P1(f)|')


%SENSOR-2
ax2 = fft(acc_sensor_2(1,:));
P2 = abs(ax2/L);
P1x = P2(1:L/2+1);
P1x(2:end-1) = 2*P1x(2:end-1);
```

```
ay2 = fft(acc_sensor_2(2,:));
P2 = abs(ay2/L);
P1y = P2(1:L/2+1);
P1y(2:end-1) = 2*P1y(2:end-1);

az2 = fft(acc_sensor_2(3,:));
P2 = abs(az2/L);
P1z = P2(1:L/2+1);
P1z(2:end-1) = 2*P1z(2:end-1);

figure(94)
% subplot(3,1,1)
% plot(f,P1x)
% title('Power Spectrum of acc sensor 2 x ')
% xlabel('f (Hz)')
% ylabel('|P1(f)|')

subplot(2,1,1)
% figure(95)
plot(f,P1y)
title('Power Spectrum of sensor 2 y acc ')
xlabel('f (Hz)')
ylabel('|P1(f)|')

subplot(2,1,2)
% figure(96)
plot(f,P1z)
title('Power Spectrum of sensor 2 z acc ')
xlabel('f (Hz)')
ylabel('|P1(f)|')

%SENSOR-3
ax3 = fft(acc_sensor_3(1,:));
P2 = abs(ax2/L);
P1x = P2(1:L/2+1);
P1x(2:end-1) = 2*P1x(2:end-1);

ay3 = fft(acc_sensor_3(2,:));
P2 = abs(ay3/L);
P1y = P2(1:L/2+1);
P1y(2:end-1) = 2*P1y(2:end-1);

az3 = fft(acc_sensor_3(3,:));
P2 = abs(az3/L);
P1z = P2(1:L/2+1);
P1z(2:end-1) = 2*P1z(2:end-1);

figure(97)
% subplot(3,1,1)
% plot(f,P1x)
% title('Power Spectrum of acc sensor 3 x ')
% xlabel('f (Hz)')
% ylabel('|P1(f)|')

subplot(2,1,1)
% figure(98)
plot(f,P1y)
title('Power Spectrum of sensor 3 y acc ')
xlabel('f (Hz)')
ylabel('|P1(f)|')

subplot(2,1,2)
% figure(99)
plot(f,P1z)
title('Power Spectrum of sensor 3 z acc ')
xlabel('f (Hz)')
ylabel('|P1(f)|')


%% LOWPASS filtering the acceleration data

switch 1
    case 1
        %SENSOR-1
order = 20;
filtCutOff = 1.11;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
a_sensor_1(1,:) = filtfilt(b, p, acc_sensor_1(1,:));

filtCutOff = 3.35;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
a_sensor_1(2,:) = filtfilt(b, p, acc_sensor_1(2,:));

filtCutOff = 3.3;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
a_sensor_1(3,:) = filtfilt(b, p, acc_sensor_1(3,:));

%SENSOR-2
filtCutOff = 1.41;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
a_sensor_2(1,:) = filtfilt(b, p, acc_sensor_2(1,:));

filtCutOff = 3.35;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
a_sensor_2(2,:) = filtfilt(b, p, acc_sensor_2(2,:));

filtCutOff = 3.3;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
```

```
a_sensor_2(3,:) = filtfilt(b, p, acc_sensor_2(3,:));

%SENSOR-3
filtCutOff = 1.41;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
a_sensor_3(1,:) = filtfilt(b, p, acc_sensor_3(1,:));

filtCutOff = 3.3;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
a_sensor_3(2,:) = filtfilt(b, p, acc_sensor_3(2,:));

filtCutOff = 3.7;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
a_sensor_3(3,:) = filtfilt(b, p, acc_sensor_3(3,:));
    case 4
%SENSOR-1
order = 20;
filtCutOff = 1.11;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
a_sensor_1(1,:) = filtfilt(b, p, acc_sensor_1(1,:));

filtCutOff = 2.6;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
a_sensor_1(2,:) = filtfilt(b, p, acc_sensor_1(2,:));

filtCutOff = 2.97;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
a_sensor_1(3,:) = filtfilt(b, p, acc_sensor_1(3,:));

%SENSOR-2
filtCutOff = 1.41;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
a_sensor_2(1,:) = filtfilt(b, p, acc_sensor_2(1,:));

filtCutOff = 2.6;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
a_sensor_2(2,:) = filtfilt(b, p, acc_sensor_2(2,:));

filtCutOff = 2.9;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
a_sensor_2(3,:) = filtfilt(b, p, acc_sensor_2(3,:));

%SENSOR-3
filtCutOff = 1.41;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
a_sensor_3(1,:) = filtfilt(b, p, acc_sensor_3(1,:));

filtCutOff = 2.6;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
a_sensor_3(2,:) = filtfilt(b, p, acc_sensor_3(2,:));

filtCutOff = 2.9;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'low');
a_sensor_3(3,:) = filtfilt(b, p, acc_sensor_3(3,:));
end

%% Kalman Filter acc data to reduce noise

%SENSOR-1
xk = 0; Pk = 0; R = 60; Q = 10;
for kk=1:numel(t)
 zk = acc_sensor_1(1,kk);
 Pk = Pk + Q;
 Kk = Pk / ( Pk + R );
 xk = xk + (Kk * (zk - xk) );
 Pk = (1-Kk)*Pk;
 a_sensor_1(1,kk) = xk;
end

% xk = 0; Pk = 0; R = 30; Q = 1;
for kk=1:numel(t)
 zk = acc_sensor_1(2,kk);
 Pk = Pk + Q;
 Kk = Pk / ( Pk + R );
 xk = xk + (Kk * (zk - xk) );
 Pk = (1-Kk)*Pk;
 a_sensor_1(2,kk) = xk;
end

% xk = 0; Pk = 0; R = 30; Q = 1;
for kk=1:numel(t)
 zk = acc_sensor_1(3,kk);
 Pk = Pk + Q;
 Kk = Pk / ( Pk + R );
 xk = xk + (Kk * (zk - xk) );
 Pk = (1-Kk)*Pk;
 a_sensor_1(3,kk) = xk;
end

%SENSOR-2
% xk = 0; Pk = 0; R = 30; Q = 1;
for kk=1:numel(t)
 zk = acc_sensor_2(1,kk);
 Pk = Pk + Q;
 Kk = Pk / ( Pk + R );
 xk = xk + (Kk * (zk - xk) );
 Pk = (1-Kk)*Pk;
 a_sensor_2(1,kk) = xk;
end
```

```
% xk = 0; Pk = 0; R = 30; Q = 1;
for kk=1:numel(t)
 zk = acc_sensor_2(2,kk);
 Pk = Pk + Q;
 Kk = Pk / ( Pk + R );
 xk = xk + (Kk * (zk - xk) );
 Pk = (1-Kk)*Pk;
 a_sensor_2(2,kk) = xk;
end

% xk = 0; Pk = 0; R = 30; Q = 1;
for kk=1:numel(t)
 zk = acc_sensor_2(3,kk);
 Pk = Pk + Q;
 Kk = Pk / ( Pk + R );
 xk = xk + (Kk * (zk - xk) );
 Pk = (1-Kk)*Pk;
 a_sensor_2(3,kk) = xk;
end

%SENSOR-3
% xk = 0; Pk = 0; R = 30; Q = 1;
for kk=1:numel(t)
 zk = acc_sensor_3(1,kk);
 Pk = Pk + Q;
 Kk = Pk / ( Pk + R );
 xk = xk + (Kk * (zk - xk) );
 Pk = (1-Kk)*Pk;
 a_sensor_3(1,kk) = xk;
end

% xk = 0; Pk = 0; R = 30; Q = 1;
for kk=1:numel(t)
 zk = acc_sensor_3(2,kk);
 Pk = Pk + Q;
 Kk = Pk / ( Pk + R );
 xk = xk + (Kk * (zk - xk) );
 Pk = (1-Kk)*Pk;
 a_sensor_3(2,kk) = xk;
end

% xk = 0; Pk = 0; R = 30; Q = 1;
for kk=1:numel(t)
 zk = acc_sensor_3(3,kk);
 Pk = Pk + Q;
 Kk = Pk / ( Pk + R );
 xk = xk + (Kk * (zk - xk) );
 Pk = (1-Kk)*Pk;
 a_sensor_3(3,kk) = xk;
end


%% acceleration final plots
switch 2
    case 1 % RAW
figure(3)
subplot(2,1,1)
hold on
plot(t,acc_sensor_1(2,:),'r')
plot(t,acc_sensor_2(2,:),'g')
plot(t,acc_sensor_3(2,:),'b')
legend('1','2','3')
xlabel('Time(s)')
ylabel('Y acceleration(m/s^2)')
title('Y acceleration')
grid on
hold off


subplot(2,1,2)
hold on
plot(t,acc_sensor_1(3,:),'r')
plot(t,acc_sensor_2(3,:),'g')
plot(t,acc_sensor_3(3,:),'b')
legend('1','2','3')
xlabel('Time(s)')
ylabel('Z acceleration(m/s^2)')
title('Z acceleration')
grid on
hold off
    case 2 %FILTERED
        figure(3)
subplot(2,1,1)
hold on
plot(t,a_sensor_1(2,:),'r')
plot(t,a_sensor_2(2,:),'g')
plot(t,a_sensor_3(2,:),'b')
legend('1','2','3')
xlabel('Time(s)')
ylabel('Y acceleration(m/s^2)')
title('Y acceleration')
grid on
hold off


subplot(2,1,2)
hold on
plot(t,a_sensor_1(3,:),'r')
```

```
plot(t,a_sensor_2(3,:),'g')
plot(t,a_sensor_3(3,:),'b')
legend('1','2','3')
xlabel('Time(s)')
ylabel('Z acceleration(m/s^2)')
title('Z acceleration')
grid on
hold off
end



%% ANGLE OF HEEL CALC
switch 2
    case 1 %raw
%FOR IMU-1
theta_1 = atand(acc_sensor_1(3,:)./acc_sensor_1(2,:));
% phi_1 = atan(acc_sensor_1(3,:)./acc_sensor_1(1,:));

%FOR IMU-2
theta_2 = atand(acc_sensor_2(3,:)./acc_sensor_2(2,:));
% phi_2 = atan(acc_sensor_2(3,:)./acc_sensor_2(1,:));

%FOR IMU-4
theta_3 = atand(acc_sensor_3(3,:)./acc_sensor_3(2,:));
% phi_3= atan(acc_sensor_3(3,:)./acc_sensor_3(1,:)); %all dependent on time
    case 2 %processed
        %FOR IMU-1
theta_1 = atand(a_sensor_1(3,:)./a_sensor_1(2,:));
% phi_1 = atan(a_sensor_1(3,:)./a_sensor_1(1,:));

%FOR IMU-2
theta_2 = atand(a_sensor_2(3,:)./a_sensor_2(2,:));
% phi_2 = atan(a_sensor_2(3,:)./a_sensor_2(1,:));

%FOR IMU-4
theta_3 = atand(a_sensor_3(3,:)./a_sensor_3(2,:));
% phi_3= atan(a_sensor_3(3,:)./a_sensor_3(1,:)); %all dependent on time
end

figure(6)
hold on
plot(t,theta_1,'r')
plot(t,theta_2,'g')
plot(t,theta_3,'b')
xlabel('Time(s)')
ylabel('Heeling angle(deg)')
legend('1','2','3')
title('Angle of Heel')
grid on
hold off



%% Integration
switch 2
    case 1 %raw
%SENSOR-1
acc_sensor_1_x = (acc_sensor_1(1,:));
acc_sensor_1_y = (acc_sensor_1(2,:));
acc_sensor_1_z = (acc_sensor_1(3,:));

v_sensor_1_x = cumtrapz(t,acc_sensor_1_x);%INTEGRATION
v_sensor_1_y = cumtrapz(t,acc_sensor_1_y);
v_sensor_1_z = cumtrapz(t,acc_sensor_1_z);

%SENSOR-2
acc_sensor_2_x = (acc_sensor_2(1,:));
acc_sensor_2_y = (acc_sensor_2(2,:));
acc_sensor_2_z = (acc_sensor_2(3,:));

v_sensor_2_x = cumtrapz(t,acc_sensor_2_x);
v_sensor_2_y = cumtrapz(t,acc_sensor_2_y);
v_sensor_2_z = cumtrapz(t,acc_sensor_2_z);%INTEGRATION

%SENSOR-3
acc_sensor_3_x = (acc_sensor_3(1,:));
acc_sensor_3_y = (acc_sensor_3(2,:));
acc_sensor_3_z = (acc_sensor_3(3,:));

v_sensor_3_x = cumtrapz(t,acc_sensor_3_x);
v_sensor_3_y = cumtrapz(t,acc_sensor_3_y);
v_sensor_3_z = cumtrapz(t,acc_sensor_3_z);%INTEGRATION

    case 2 %processed
%SENSOR-1
a_sensor_1_x = (a_sensor_1(1,:));
a_sensor_1_y = (a_sensor_1(2,:));
a_sensor_1_z = (a_sensor_1(3,:));

v_sensor_1_x = cumtrapz(t,a_sensor_1_x);%INTEGRATION
v_sensor_1_y = cumtrapz(t,a_sensor_1_y);
v_sensor_1_z = cumtrapz(t,a_sensor_1_z);

% SENSOR-2
a_sensor_2_x = (a_sensor_2(1,:));
a_sensor_2_y = (a_sensor_2(2,:));
a_sensor_2_z = (a_sensor_2(3,:));
```

```
v_sensor_2_x = cumtrapz(t,a_sensor_2_x);
v_sensor_2_y = cumtrapz(t,a_sensor_2_y);
v_sensor_2_z = cumtrapz(t,a_sensor_2_z);%INTEGRATION

% SENSOR-3
a_sensor_3_x = (a_sensor_3(1,:));
a_sensor_3_y = (a_sensor_3(2,:));
a_sensor_3_z = (a_sensor_3(3,:));

v_sensor_3_x = cumtrapz(t,a_sensor_3_x);
v_sensor_3_y = cumtrapz(t,a_sensor_3_y);
v_sensor_3_z = cumtrapz(t,a_sensor_3_z);%INTEGRATION
end
%% DETRENDING VELOCITY

dt_order = 4;
v_sensor_1_x = detrend(v_sensor_1_x,dt_order);
v_sensor_1_y = detrend(v_sensor_1_y,dt_order);
v_sensor_1_z = detrend(v_sensor_1_z,dt_order); %DETREND

v_sensor_2_x = detrend(v_sensor_2_x,dt_order);
v_sensor_2_y = detrend(v_sensor_2_y,dt_order);
v_sensor_2_z = detrend(v_sensor_2_z,dt_order);%DETREND

v_sensor_3_x = detrend(v_sensor_3_x,dt_order);
v_sensor_3_y = detrend(v_sensor_3_y,dt_order);
v_sensor_3_z = detrend(v_sensor_3_z,dt_order); %DETREND

%% FFT TO VELOCITY
%SENSOR-1
vx1 = fft(v_sensor_1_x);
P2 = abs(vx1/L);
P1x = P2(1:L/2+1);
P1x(2:end-1) = 2*P1x(2:end-1);

vy1 = fft(v_sensor_1_y);
P2 = abs(vy1/L);
P1y = P2(1:L/2+1);
P1y(2:end-1) = 2*P1y(2:end-1);

vz1 = fft(v_sensor_1_z);
P2 = abs(vz1/L);
P1z = P2(1:L/2+1);
P1z(2:end-1) = 2*P1z(2:end-1);

figure(991)
% subplot(3,1,1)
% plot(f,P1x)
% title('Power Spectrum of velocity 1 x ')
% xlabel('f (Hz)')
% ylabel('|P1(f)|')

subplot(2,1,1)
% figure(992)
plot(f,P1y)
title('Power Spectrum of sensor 1 y velocity')
xlabel('f (Hz)')
ylabel('|P1(f)|')

subplot(2,1,2)
% figure(993)
plot(f,P1z)
title('Power Spectrum of sensor 1 z velocity')
xlabel('f (Hz)')
ylabel('|P1(f)|')


%SENSOR-2
vx1 = fft(v_sensor_2_x);
P2 = abs(vx1/L);
P1x = P2(1:L/2+1);
P1x(2:end-1) = 2*P1x(2:end-1);

vy1 = fft(v_sensor_2_y);
P2 = abs(vy1/L);
P1y = P2(1:L/2+1);
P1y(2:end-1) = 2*P1y(2:end-1);

vz1 = fft(v_sensor_2_z);
P2 = abs(vz1/L);
P1z = P2(1:L/2+1);
P1z(2:end-1) = 2*P1z(2:end-1);

figure(994)
% subplot(3,1,1)
% plot(f,P1x)
% title('Power Spectrum of velocity 2 x ')
% xlabel('f (Hz)')
% ylabel('|P1(f)|')

% figure(995)
subplot(2,1,1)
plot(f,P1y)
title('Power Spectrum of sensor 2 y velocity')
xlabel('f (Hz)')
ylabel('|P1(f)|')

% figure(996)
```

```
subplot(2,1,2)
plot(f,P1z)
title('Power Spectrum of sensor 2 z velocity ')
xlabel('f (Hz)')
ylabel('|P1(f)|')


%SENSOR-3
vx1 = fft(v_sensor_3_x);
P2 = abs(vx1/L);
P1x = P2(1:L/2+1);
P1x(2:end-1) = 2*P1x(2:end-1);

vy1 = fft(v_sensor_3_y);
P2 = abs(vy1/L);
P1y = P2(1:L/2+1);
P1y(2:end-1) = 2*P1y(2:end-1);

vz1 = fft(v_sensor_3_z);
P2 = abs(vz1/L);
P1z = P2(1:L/2+1);
P1z(2:end-1) = 2*P1z(2:end-1);

figure(997)
% subplot(3,1,1)
% plot(f,P1x)
% title('Power Spectrum of velocity 3 x ')
% xlabel('f (Hz)')
% ylabel('|P1(f)|')

% figure(998)
subplot(2,1,1)
plot(f,P1y)
title('Power Spectrum of sensor 3 y velocity')
xlabel('f (Hz)')
ylabel('|P1(f)|')

% figure(999)
subplot(2,1,2)
plot(f,P1z)
title('Power Spectrum of sensor 3 z velocity')
xlabel('f (Hz)')
ylabel('|P1(f)|')

%% HIGH pass filtering velocity

switch 1
    case 1
%SENSOR-1
order = 1;
filtCutOff = 0.5;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_x = filtfilt(b, p, v_sensor_1_x);

filtCutOff = 0.1;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_y = filtfilt(b, p, v_sensor_1_y);

filtCutOff = 0.125;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_z = filtfilt(b, p, v_sensor_1_z);


%SENSOR-2
filtCutOff = 0.104;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_2_x = filtfilt(b, p, v_sensor_2_x);


filtCutOff = 0.05;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_2_y = filtfilt(b, p, v_sensor_2_y);


filtCutOff = 0.125;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_2_z = filtfilt(b, p, v_sensor_2_z);

%SENSOR-3
filtCutOff = 0.680;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_x = filtfilt(b, p, v_sensor_3_x);


filtCutOff = 0.000001;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_y = filtfilt(b, p, v_sensor_3_y);

filtCutOff = 0.13;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_z = filtfilt(b, p, v_sensor_3_z);
    case 2
        %SENSOR-1
order = 1;
filtCutOff = 0.2;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_x = filtfilt(b, p, v_sensor_1_x);
```

```
filtCutOff = 0.06;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_y = filtfilt(b, p, v_sensor_1_y);

filtCutOff = 0.18;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_z = filtfilt(b, p, v_sensor_1_z);


%SENSOR-2
filtCutOff = 0.2;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_2_x = filtfilt(b, p, v_sensor_2_x);


filtCutOff = 0.06;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_2_y = filtfilt(b, p, v_sensor_2_y);


filtCutOff = 0.24;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_2_z = filtfilt(b, p, v_sensor_2_z);

%SENSOR-3
filtCutOff = 0.2;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_x = filtfilt(b, p, v_sensor_3_x);


filtCutOff = 0.06;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_y = filtfilt(b, p, v_sensor_3_y);

filtCutOff = 0.18;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_z = filtfilt(b, p, v_sensor_3_z);

    case 3
        %SENSOR-1
order = 1;
filtCutOff = 0.5;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_x = filtfilt(b, p, v_sensor_1_x);


filtCutOff = 0.15;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_y = filtfilt(b, p, v_sensor_1_y);

filtCutOff = 0.15;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_z = filtfilt(b, p, v_sensor_1_z);


%SENSOR-2
filtCutOff = 0.104;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_2_x = filtfilt(b, p, v_sensor_2_x);


filtCutOff = 0.104;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_2_y = filtfilt(b, p, v_sensor_2_y);


filtCutOff = 0.15;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_2_z = filtfilt(b, p, v_sensor_2_z);

%SENSOR-3
filtCutOff = 0.680;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_x = filtfilt(b, p, v_sensor_3_x);


filtCutOff = 0.001;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_y = filtfilt(b, p, v_sensor_3_y);

filtCutOff = 0.15;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_z = filtfilt(b, p, v_sensor_3_z);
    case 4
        %SENSOR-1
order = 1;
filtCutOff = 0.52;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_x = filtfilt(b, p, v_sensor_1_x);


filtCutOff = 0.11;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_y = filtfilt(b, p, v_sensor_1_y);

filtCutOff = 0.11;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_z = filtfilt(b, p, v_sensor_1_z);
```

```
%SENSOR-2
filtCutOff = 0.52;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_2_x = filtfilt(b, p, v_sensor_2_x);


filtCutOff = 0.11;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_2_y = filtfilt(b, p, v_sensor_2_y);


filtCutOff = 0.037;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_2_z = filtfilt(b, p, v_sensor_2_z);

%SENSOR-3
filtCutOff = 0.52;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_x = filtfilt(b, p, v_sensor_3_x);


filtCutOff = 0.07;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_y = filtfilt(b, p, v_sensor_3_y);

filtCutOff = 0.11;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_z = filtfilt(b, p, v_sensor_3_z);
    case 5
        %SENSOR-1
order = 1;
filtCutOff = 0.18;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_x = filtfilt(b, p, v_sensor_1_x);


filtCutOff = 0.07;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_y = filtfilt(b, p, v_sensor_1_y);

filtCutOff = 0.037;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_z = filtfilt(b, p, v_sensor_1_z);


%SENSOR-2
filtCutOff = 0.18;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_2_x = filtfilt(b, p, v_sensor_2_x);


filtCutOff = 0.07;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_2_y = filtfilt(b, p, v_sensor_2_y);


filtCutOff = 0.037;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_2_z = filtfilt(b, p, v_sensor_2_z);

%SENSOR-3
filtCutOff = 0.18;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_x = filtfilt(b, p, v_sensor_3_x);


filtCutOff = 0.07;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_y = filtfilt(b, p, v_sensor_3_y);

filtCutOff = 0.037;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_z = filtfilt(b, p, v_sensor_3_z);
    case 6
        %SENSOR-1
order = 1;
filtCutOff = 0.21;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_x = filtfilt(b, p, v_sensor_1_x);


filtCutOff = 0.1;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_y = filtfilt(b, p, v_sensor_1_y);

filtCutOff = 0.03;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_1_z = filtfilt(b, p, v_sensor_1_z);


%SENSOR-2
filtCutOff = 0.14;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_2_x = filtfilt(b, p, v_sensor_2_x);


filtCutOff = 0.07;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
```

```
v_sensor_2_y = filtfilt(b, p, v_sensor_2_y);


filtCutOff = 0.03;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_2_z = filtfilt(b, p, v_sensor_2_z);

%SENSOR-3
filtCutOff = 0.14;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_x = filtfilt(b, p, v_sensor_3_x);


filtCutOff = 0.03;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_y = filtfilt(b, p, v_sensor_3_y);

filtCutOff = 0.07;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
v_sensor_3_z = filtfilt(b, p, v_sensor_3_z);
end

%% INTEGRATING VELOCITY TO FIND POSITION

s_sensor_1_x = cumtrapz(t,v_sensor_1_x);
s_sensor_1_y = cumtrapz(t,v_sensor_1_y);
s_sensor_1_z = cumtrapz(t,v_sensor_1_z);%INTEGRATION

s_sensor_2_x = cumtrapz(t,v_sensor_2_x);
s_sensor_2_y = cumtrapz(t,v_sensor_2_y);
s_sensor_2_z = cumtrapz(t,v_sensor_2_z);%INTEGRATION

s_sensor_3_x = cumtrapz(t,v_sensor_3_x);
s_sensor_3_y = cumtrapz(t,v_sensor_3_y);
s_sensor_3_z = cumtrapz(t,v_sensor_3_z);%INTEGRATION

%% FFT to position data

%SENSOR-1
sx1 = fft(s_sensor_1_x);
P2 = abs(sx1/L);
P1x = P2(1:L/2+1);
P1x(2:end-1) = 2*P1x(2:end-1);

sy1 = fft(s_sensor_1_y);
P2 = abs(sy1/L);
P1y = P2(1:L/2+1);
P1y(2:end-1) = 2*P1y(2:end-1);

sz1 = fft(s_sensor_1_z);
P2 = abs(sz1/L);
P1z = P2(1:L/2+1);
P1z(2:end-1) = 2*P1z(2:end-1);

figure(9991)
% subplot(3,1,1)
% plot(f,P1x)
% title('Power Spectrum of position 1 x ')
% xlabel('f (Hz)')
% ylabel('|P1(f)|')

% figure(9992)
subplot(2,1,1)
plot(f,P1y)
title('Power Spectrum of sensor 1 y position')
xlabel('f (Hz)')
ylabel('|P1(f)|')

% figure(9993)
subplot(2,1,2)
plot(f,P1z)
title('Power Spectrum of sensor 1 z position')
xlabel('f (Hz)')
ylabel('|P1(f)|')


%SENSOR-2
sx2 = fft(s_sensor_2_x);
P2 = abs(sx2/L);
P1x = P2(1:L/2+1);
P1x(2:end-1) = 2*P1x(2:end-1);

sy2 = fft(s_sensor_2_y);
P2 = abs(sy2/L);
P1y = P2(1:L/2+1);
P1y(2:end-1) = 2*P1y(2:end-1);

sz2 = fft(s_sensor_2_z);
P2 = abs(sz2/L);
P1z = P2(1:L/2+1);
P1z(2:end-1) = 2*P1z(2:end-1);

figure(9994)
% subplot(3,1,1)
% plot(f,P1x)
% title('Power Spectrum of position 2 x ')
% xlabel('f (Hz)')
% ylabel('|P1(f)|')
```

```
% figure(9995)
subplot(2,1,1)
plot(f,P1y)
title('Power Spectrum of sensor 2 y position')
xlabel('f (Hz)')
ylabel('|P1(f)|')

% figure(9996)
subplot(2,1,2)
plot(f,P1z)
title('Power Spectrum of sensor 2 z position')
xlabel('f (Hz)')
ylabel('|P1(f)|')


%SENSOR-3
sx3 = fft(s_sensor_3_x);
P2 = abs(sx3/L);
P1x = P2(1:L/2+1);
P1x(2:end-1) = 2*P1x(2:end-1);

sy3 = fft(s_sensor_3_y);
P2 = abs(sy3/L);
P1y = P2(1:L/2+1);
P1y(2:end-1) = 2*P1y(2:end-1);

sz3 = fft(s_sensor_3_z);
P2 = abs(sz3/L);
P1z = P2(1:L/2+1);
P1z(2:end-1) = 2*P1z(2:end-1);

figure(9997)
% subplot(3,1,1)
% plot(f,P1x)
% title('Power Spectrum of position 3 x ')
% xlabel('f (Hz)')
% ylabel('|P1(f)|')

% figure(9998)
subplot(2,1,1)
plot(f,P1y)
title('Power Spectrum of sensor 3 y position')
xlabel('f (Hz)')
ylabel('|P1(f)|')

% figure(9999)
subplot(2,1,2)
plot(f,P1z)
title('Power Spectrum of sensor 3 z position')
xlabel('f (Hz)')
ylabel('|P1(f)|')

%% HIGH pass filtering position
switch 1
    case 1
        %SENSOR-1
order = 1;
filtCutOff = 0.04;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_1_x = filtfilt(b, p, s_sensor_1_x);

filtCutOff = 0.05;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_1_y = filtfilt(b, p, s_sensor_1_y);

filtCutOff = 0.125;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_1_z = filtfilt(b, p, s_sensor_1_z);


%SENSOR-2
filtCutOff = 0.18;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_2_x = filtfilt(b, p, s_sensor_2_x);

filtCutOff = 0.1;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_2_y = filtfilt(b, p, s_sensor_2_y);

filtCutOff = 0.05;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_2_z = filtfilt(b, p, s_sensor_2_z);


%SENSOR-3
filtCutOff = 0.11;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_3_x = filtfilt(b, p, s_sensor_3_x);

filtCutOff = 0.1;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_3_y = filtfilt(b, p, s_sensor_3_y);

filtCutOff = 0.2;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_3_z = filtfilt(b, p, s_sensor_3_z);
    case 4
%SENSOR-1
order = 1;
```

```
filtCutOff = 0.04;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_1_x = filtfilt(b, p, s_sensor_1_x);

filtCutOff = 0.07;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_1_y = filtfilt(b, p, s_sensor_1_y);

filtCutOff = 0.037;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_1_z = filtfilt(b, p, s_sensor_1_z);


%SENSOR-2
filtCutOff = 0.18;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_2_x = filtfilt(b, p, s_sensor_2_x);

filtCutOff = 0.07;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_2_y = filtfilt(b, p, s_sensor_2_y);

filtCutOff = 0.07;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_2_z = filtfilt(b, p, s_sensor_2_z);


%SENSOR-3
filtCutOff = 0.11;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_3_x = filtfilt(b, p, s_sensor_3_x);

filtCutOff = 0.07;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_3_y = filtfilt(b, p, s_sensor_3_y);

filtCutOff = 0.037;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_3_z = filtfilt(b, p, s_sensor_3_z);
    case 5
        %SENSOR-1
order = 1;
filtCutOff = 0.07;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_1_x = filtfilt(b, p, s_sensor_1_x);

filtCutOff = 0.07;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_1_y = filtfilt(b, p, s_sensor_1_y);

filtCutOff = 0.18;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_1_z = filtfilt(b, p, s_sensor_1_z);


%SENSOR-2
filtCutOff = 0.18;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_2_x = filtfilt(b, p, s_sensor_2_x);

filtCutOff = 0.037;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_2_y = filtfilt(b, p, s_sensor_2_y);

filtCutOff = 0.15;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_2_z = filtfilt(b, p, s_sensor_2_z);


%SENSOR-3
filtCutOff = 0.11;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_3_x = filtfilt(b, p, s_sensor_3_x);

filtCutOff = 0.037;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_3_y = filtfilt(b, p, s_sensor_3_y);

filtCutOff = 0.07;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_3_z = filtfilt(b, p, s_sensor_3_z);
    case 6
        %SENSOR-1
order = 1;
filtCutOff = 0.04;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_1_x = filtfilt(b, p, s_sensor_1_x);

filtCutOff = 0.07;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_1_y = filtfilt(b, p, s_sensor_1_y);

filtCutOff = 0.1;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_1_z = filtfilt(b, p, s_sensor_1_z);


%SENSOR-2
filtCutOff = 0.18;
```

```
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_2_x = filtfilt(b, p, s_sensor_2_x);

filtCutOff = 0.07;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_2_y = filtfilt(b, p, s_sensor_2_y);

filtCutOff = 0.1;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_2_z = filtfilt(b, p, s_sensor_2_z);


%SENSOR-3
filtCutOff = 0.11;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_3_x = filtfilt(b, p, s_sensor_3_x);

filtCutOff = 0.07;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_3_y = filtfilt(b, p, s_sensor_3_y);

filtCutOff = 0.07;
[b, p] = butter(order, (2*filtCutOff)/(1/samplePeriod), 'high');
s_sensor_3_z = filtfilt(b, p, s_sensor_3_z);

end
%% DETRENDING POSITION

s_sensor_1_x = detrend(s_sensor_1_x,dt_order);
s_sensor_1_y = detrend(s_sensor_1_y,dt_order);
s_sensor_1_z = detrend(s_sensor_1_z,dt_order); %DETREND

s_sensor_2_x = detrend(s_sensor_2_x,dt_order);
s_sensor_2_y = detrend(s_sensor_2_y,dt_order);
s_sensor_2_z = detrend(s_sensor_2_z,dt_order); %DETREND

s_sensor_3_x = detrend(s_sensor_3_x,dt_order);
s_sensor_3_y = detrend(s_sensor_3_y,dt_order);
s_sensor_3_z = detrend(s_sensor_3_z,dt_order); %DETREND


%% adding initial positions
s_sensor_1_x = s_sensor_1_x +s1_in(1,:);
s_sensor_1_y = s_sensor_1_y +s1_in(2,:);
s_sensor_1_z = s_sensor_1_z +s1_in(3,:);

s_sensor_2_x = s_sensor_2_x +s2_in(1,:);
s_sensor_2_y = s_sensor_2_y +s2_in(2,:);
s_sensor_2_z = s_sensor_2_z +s2_in(3,:);

s_sensor_3_x = s_sensor_3_x +s3_in(1,:);
s_sensor_3_y = s_sensor_3_y +s3_in(2,:);
s_sensor_3_z = s_sensor_3_z +s3_in(3,:);


%% INTEGRATION PLOTS

figure(7)
plot3(s_sensor_1_x,s_sensor_1_y,s_sensor_1_z,'r')
hold on
plot3(s_sensor_2_x,s_sensor_2_y,s_sensor_2_z,'g')
hold on
plot3(s_sensor_3_x,s_sensor_3_y,s_sensor_3_z,'b')
xlabel("X")
ylabel("Y")
zlabel("Z")
title("Sensor displacements in space(m)")
legend('sensor-1','sensor-2','sensor-3')
axis equal
grid on
box on
hold off



%X-Y-Z PLOTS OF VELOCITY AND POSITON
figure(8)
subplot(2,1,1)
hold on
plot(t,v_sensor_1_y,'r'),
plot(t,v_sensor_2_y,'g')
plot(t,v_sensor_3_y,'b'),
legend('1','2','3')
title('Y velocity')
xlabel('Time(s)')
ylabel('Velocity(m/s)')
grid on
hold off

subplot(2,1,2)
hold on
plot(t,v_sensor_1_z,'r'),
plot(t,v_sensor_2_z,'g')
plot(t,v_sensor_3_z,'b'),
legend('1','2','3')
title('Z velocity')
xlabel('Time(s)')
ylabel('Velocity(m/s)')
grid on
```

```
hold off


figure(11)
subplot(2,1,1)
hold on,
plot(t,s_sensor_1_y,'r'),
plot(t,s_sensor_2_y,'g')
plot(t,s_sensor_3_y,'b'),
legend('1','2','3')
title('Y-position')
xlabel('Time(s)')
ylabel('Position(m)')
grid on
hold off

subplot(2,1,2)
hold on,
plot(t,s_sensor_1_z,'r'),
plot(t,s_sensor_2_z,'g')
plot(t,s_sensor_3_z,'b'),
legend('1','2','3')
title('Z position')
xlabel('Time(s)')
ylabel('Position(m)')
grid on
hold off



%% Position KIYASLAMA

r = .95; %m
y1distance =r.*sind(sdeg_x)+s1_in(2,:);
z1distance =(r-r.*cosd(sdeg_x))+s1_in(3,:);

y2distance =r.*sind(sdeg_x)+s2_in(2,:);
z2distance =(r-r.*cosd(sdeg_x))+s2_in(3,:);

y3distance =r.*sind(sdeg_x)+s3_in(2,:);
z3distance =(r-r.*cosd(sdeg_x))+s3_in(3,:);


figure(30)
subplot(2,1,1)
hold on
plot(t,y1distance,'r')
plot(t,y2distance,'g')
plot(t,y3distance,'b')
legend('1','2','3')
title('Y position')
xlabel('Time(s)')
ylabel('Position(m)')
grid on
hold off

subplot(2,1,2)
hold on
plot(t,z1distance,'r')
plot(t,z2distance,'g')
plot(t,z3distance,'b')
legend('1','2','3')
title('Z position')
xlabel('Time(s)')
ylabel('position(m)')
grid on
hold off

figure(31)
hold on
plot3(s1_in(1,:),y1distance,z1distance,'r')
plot3(s2_in(1,:),y2distance,z2distance,'g')
plot3(s3_in(1,:),y3distance,z3distance,'b')
title("Sensor displacements in space(m)")
legend('sensor-1','sensor-2','sensor-3')
axis equal
xlabel("X")
ylabel("Y")
zlabel("Z")
xlim([-0.2 0.2])
ylim([-0.5 0.5])
grid on
box on
hold off

%% METACENTER ACC BASED

A=1;
in_P  = 140;
fin_P = 170;
for i=in_P:fin_P
    LP =i;

a1 = [s_sensor_1_y(:,LP) s_sensor_1_z(:,LP)]; %sensor1 initial coordinate(yz plane)
a2 = [s_sensor_2_y(:,LP) s_sensor_2_z(:,LP)]; %sensor2 initial coordinate(yz plane)
a3 = [s_sensor_3_y(:,LP) s_sensor_3_z(:,LP)]; %sensor3 initial coordinate (yz plane)

b1 = [s_sensor_1_y(:,LP+A) s_sensor_1_z(:,LP+A)]; %sensor1 final coordinate (yz plane)
```

```
b2 = [s_sensor_2_y(:,LP+A) s_sensor_2_z(:,LP+A)]; %sensor2 final coordinate (yz plane)
b3 = [s_sensor_3_y(:,LP+A) s_sensor_3_z(:,LP+A)]; %sensor3 final coordinate (yz plane)


m1 = (b2(2)-a2(2))/(b2(1)-a2(1));
m2 = -1/m1;
m3 = (b3(2)-a3(2))/(b3(1)-a3(1));
m4 = -1/m3;
m5 = (b1(2)-a1(2))/(b1(1)-a1(1));
m6 = -1/m5;
%1,3,5 doğru eğimi(L)
%2,4,6 sonsuz dikliğin eğimi(V)


mp1 = (a2+b2)./2;
mp2 = (a3+b3)./2;
mp3 = (a1+b1)./2;%başlangıç ve bitiş arasındaki orta nokta

c1 = mp1(2)-m1*mp1(1);
c2 = mp2(2)-m3*mp2(1);
c3 = mp3(2)-m5*mp3(1);%L doğruları sabitleri

c4 = mp1(2)-m2*mp1(1);
c5 = mp2(2)-m4*mp2(1);
c6 = mp3(2)-m6*mp3(1);%V doğruları sabitleri

syms y

co_2_3_y = double(solve(m2*y + c4 == m4*y + c5,y));
co_1_3_y = double(solve(m4*y + c5 == m6*y + c6,y));
co_1_2_y = double(solve(m2*y + c4 == m6*y + c6,y));

co_2_3_z = m2*co_2_3_y + c4;
co_1_3_z = m4*co_1_3_y +c5;
co_1_2_z = m6*co_1_2_y + c6;

mc_2_3 = [co_2_3_y,co_2_3_z];
mc_1_3 = [co_1_3_y,co_1_3_z];
mc_1_2 = [co_1_2_y,co_1_2_z];
MC = (mc_2_3 + mc_1_3 + mc_1_2)./3;

MCP(i,1:2) = [MC(1), MC(2)];
mcP_2_3(i,1:2) = [mc_2_3(1), mc_2_3(2)];
mcP_1_3(i,1:2) = [mc_1_3(1), mc_1_3(2)];
mcP_1_2(i,1:2) = [mc_1_2(1), mc_1_2(2)];
end


MCP = [MCP(in_P:fin_P,1), MCP(in_P:fin_P,2)];
mcP_1_2 = [mcP_1_2(in_P:fin_P,1), mcP_1_2(in_P:fin_P,2)];
mcP_1_3 = [mcP_1_3(in_P:fin_P,1), mcP_1_3(in_P:fin_P,2)];
mcP_2_3 = [mcP_2_3(in_P:fin_P,1), mcP_2_3(in_P:fin_P,2)];

MCP     = rmoutliers(MCP,"mean");
mcP_1_2 = rmoutliers(mcP_1_2,"mean");
mcP_1_3 = rmoutliers(mcP_1_3,"mean");
mcP_2_3 = rmoutliers(mcP_2_3,"mean");

MC_mean = [mean(MCP(:,1)),mean(MCP(:,2))]
mcP_1_2_mean = [mean(mcP_1_2(:,1)),mean(mcP_1_2(:,2))]
mcP_1_3_mean = [mean(mcP_1_3(:,1)),mean(mcP_1_3(:,2))]
mcP_2_3_mean = [mean(mcP_2_3(:,1)),mean(mcP_2_3(:,2))]

MC_std = [std(MCP(:,1)),std(MCP(:,2))]
mcP_1_2_std = [std(mcP_1_2(:,1)),std(mcP_1_2(:,2))]
mcP_1_3_std = [std(mcP_1_3(:,1)),std(mcP_1_3(:,2))]
mcP_2_3_std = [std(mcP_2_3(:,1)),std(mcP_2_3(:,2))]


figure(14)
lw_1 = 2;
lw_2 = 4;
plot(mcP_1_3(:,1),mcP_1_3(:,2),'r.')
hold on
plot(mcP_2_3(:,1),mcP_2_3(:,2),'g.')
hold on
plot(mcP_1_2(:,1),mcP_1_2(:,2),'b.')
hold on
plot(MCP(:,1),MCP(:,2),'k.')
plot(mcP_1_3_mean(1),mcP_1_3_mean(2),'r*')
plot(mcP_2_3_mean(1),mcP_2_3_mean(2),'g*')
plot(mcP_1_2_mean(1),mcP_1_2_mean(2),'b*')
plot(MC_mean(1),MC_mean(2),'k*')
legend('MC_1_3','MC_2_3','MC_1_2','MC','MC_1_3 mean','MC_2_3 mean','MC_1_2 mean','MC mean')
title('Center of Rotation Based on Acceleration Derived Position')
grid on
xlabel('Y')
ylabel('Z')
hold off




%% METACENTER - KIYASLAMA

A=1;
in_P  = 140;
fin_P = 170;
```

```
for i=in_P:fin_P
    LP =i;

a1 = [y1distance(:,LP) z1distance(:,LP)]; %sensor1 initial coordinate(yz plane)
a2 = [y2distance(:,LP) z2distance(:,LP)]; %sensor2 initial coordinate(yz plane)
a3 = [y3distance(:,LP) z3distance(:,LP)]; %sensor3 initial coordinate (yz plane)

b1 = [y1distance(:,LP+A) z1distance(:,LP+A)]; %sensor1 final coordinate (yz plane)
b2 = [y2distance(:,LP+A) z2distance(:,LP+A)]; %sensor2 final coordinate (yz plane)
b3 = [y3distance(:,LP+A) z3distance(:,LP+A)]; %sensor3 final coordinate (yz plane)


m1 = (b2(2)-a2(2))/(b2(1)-a2(1));
m2 = -1/m1;
m3 = (b3(2)-a3(2))/(b3(1)-a3(1));
m4 = -1/m3;
m5 = (b1(2)-a1(2))/(b1(1)-a1(1));
m6 = -1/m5;
%1,3,5 doğru eğimi(L)
%2,4,6 sonsuz dikliğin eğimi(V)


mp1 = (a2+b2)./2;
mp2 = (a3+b3)./2;
mp3 = (a1+b1)./2;%başlangıç ve bitiş arasındaki orta nokta

c1 = mp1(2)-m1*mp1(1);
c2 = mp2(2)-m3*mp2(1);
c3 = mp3(2)-m5*mp3(1);%L doğruları sabitleri

c4 = mp1(2)-m2*mp1(1);
c5 = mp2(2)-m4*mp2(1);
c6 = mp3(2)-m6*mp3(1);%V doğruları sabitleri

syms y

co_2_3_y = double(solve(m2*y + c4 == m4*y + c5,y));
co_1_3_y = double(solve(m4*y + c5 == m6*y + c6,y));
co_1_2_y = double(solve(m2*y + c4 == m6*y + c6,y));

co_2_3_z = m2*co_2_3_y + c4;
co_1_3_z = m4*co_1_3_y +c5;
co_1_2_z = m6*co_1_2_y + c6;

mc_2_3 = [co_2_3_y,co_2_3_z];
mc_1_3 = [co_1_3_y,co_1_3_z];
mc_1_2 = [co_1_2_y,co_1_2_z];
MC = (mc_2_3 + mc_1_3 + mc_1_2)./3;

MCPk(i,1:2) = [MC(1), MC(2)];
mcPk_2_3(i,1:2) = [mc_2_3(1), mc_2_3(2)];
mcPk_1_3(i,1:2) = [mc_1_3(1), mc_1_3(2)];
mcPk_1_2(i,1:2) = [mc_1_2(1), mc_1_2(2)];
end

MCPk = [MCPk(in_P:fin_P,1), MCPk(in_P:fin_P,2)];
mcPk_1_2 = [mcPk_1_2(in_P:fin_P,1), mcPk_1_2(in_P:fin_P,2)];
mcPk_1_3 = [mcPk_1_3(in_P:fin_P,1), mcPk_1_3(in_P:fin_P,2)];
mcPk_2_3 = [mcPk_2_3(in_P:fin_P,1), mcPk_2_3(in_P:fin_P,2)];

MCPk     = rmoutliers(MCPk,"mean");
mcPk_1_2 = rmoutliers(mcPk_1_2,"mean");
mcPk_1_3 = rmoutliers(mcPk_1_3,"mean");
mcPk_2_3 = rmoutliers(mcPk_2_3,"mean");

MCk_mean = [mean(MCPk(:,1)),mean(MCPk(:,2))]
mcPk_1_2_mean = [mean(mcPk_1_2(:,1)),mean(mcPk_1_2(:,2))]
mcPk_1_3_mean = [mean(mcPk_1_3(:,1)),mean(mcPk_1_3(:,2))]
mcPk_2_3_mean = [mean(mcPk_2_3(:,1)),mean(mcPk_2_3(:,2))]

MCk_std = [std(MCPk(:,1)),std(MCPk(:,2))]
mcPk_1_2_std = [std(mcPk_1_2(:,1)),std(mcPk_1_2(:,2))]
mcPk_1_3_std = [std(mcPk_1_3(:,1)),std(mcPk_1_3(:,2))]
mcPk_2_3_std = [std(mcPk_2_3(:,1)),std(mcPk_2_3(:,2))]



figure(32)
plot(mcPk_1_3(:,1),mcPk_1_3(:,2),'r.')
hold on
plot(mcPk_2_3(:,1),mcPk_2_3(:,2),'g.')
hold on
plot(mcPk_1_2(:,1),mcPk_1_2(:,2),'b.')
hold on
plot(MCPk(:,1),MCPk(:,2),'k.')
hold on
plot(mcPk_1_3_mean(1),mcPk_1_3_mean(2),'r*')
plot(mcPk_2_3_mean(1),mcPk_2_3_mean(2),'g*')
plot(mcPk_1_2_mean(1),mcPk_1_2_mean(2),'b*')
plot(MCk_mean(1),MCk_mean(2),'k*')
legend('MC_1_3','MC_2_3','MC_1_2','MC','MC_1_3 mean','MC_2_3 mean','MC_1_2 mean','MC mean')
title('Center of Rotation Based on Gyroscope Derived Position')
grid on
xlabel('Y')
ylabel('Z')
hold off
```

```
%% ERROR CALCULATION

difference_y1 = s_sensor_1_y(2:end) - y1distance(2:end);
difference_z1 = s_sensor_1_z(2:end) - z1distance(2:end);

difference_y2 = s_sensor_2_y(2:end) - y2distance(2:end);
difference_z2 = s_sensor_2_z(2:end) - z2distance(2:end);

difference_y3 = s_sensor_3_y(2:end) - y3distance(2:end);
difference_z3 = s_sensor_3_z(2:end) - z3distance(2:end);

er_y1 = (difference_y1./y1distance(2:end)).*100;
er_z1 = (difference_z1./z1distance(2:end)).*100;

er_y2 = (difference_y2./y2distance(2:end)).*100;
er_z2 = (difference_z2./z2distance(2:end)).*100;

er_y3 = (difference_y3./y3distance(2:end)).*100;
er_z3 = (difference_z3./z3distance(2:end)).*100;

er_y1 = abs(rmoutliers(er_y1));
er_z1 = abs(rmoutliers(er_z1));

er_y2 = abs(rmoutliers(er_y2));
er_z2 = abs(rmoutliers(er_z2));

er_y3 = abs(rmoutliers(er_y3));
er_z3 = abs(rmoutliers(er_z3));


figure(40)
subplot(2,1,1)
plot(t(2:end),difference_y1,'r')
hold on
plot(t(2:end),difference_y2,'g')
hold on
plot(t(2:end),difference_y3,'b')
legend('1','2','3')
title('Difference in Y Position')
xlabel('Time')
ylabel('Y Position(m)')
grid on
hold off

subplot(2,1,2)
plot(t(2:end),difference_z1,'r')
hold on
plot(t(2:end),difference_z2,'g')
hold on
plot(t(2:end),difference_z3,'b')
legend('1','2','3')
title('Difference in Z Position')
xlabel('time')
ylabel('Z Position(m)')
grid on
hold off

figure(41)
subplot(2,1,1)
plot(linspace(0,t(end),numel(er_y1)),er_y1,'r')
hold on
plot(linspace(0,t(end),numel(er_y2)),er_y2,'g')
hold on
plot(linspace(0,t(end),numel(er_y3)),er_y3,'b')
legend('1','2','3')
title('Error in Y Position')
xlabel('Time')
ylabel('Postion Error(%)')
grid on
hold off

subplot(2,1,2)
plot(linspace(0,t(end),numel(er_z1)),er_z1,'r')
hold on
plot(linspace(0,t(end),numel(er_z2)),er_z2,'g')
hold on
plot(linspace(0,t(end),numel(er_z3)),er_z3,'b')
legend('1','2','3')
title('Error in Z Position')
xlabel('Time')
ylabel('Postion Error(%)')
grid on
hold off

%mean errors
error_1 = [mean(er_y1),mean(er_z1)]
error_2 = [mean(er_y2),mean(er_z2)]
error_3 = [mean(er_y3),mean(er_z3)]
```

7.2 APPENDIX-2 ARDUINO IDE SCRIPT

```
// Include Wire Library for I2C
#include <Wire.h>



long accelX, accelY, accelZ;
float gForceX, gForceY, gForceZ;

long gyroX, gyroY, gyroZ;
float rotX, rotY, rotZ;

void TCA9548A(uint8_t bus)
{
  Wire.beginTransmission(0x70);  // TCA9548A address is 0x70
  Wire.write(1 << bus);          // send byte to select bus
  Wire.endTransmission();
}

void setup() {
  // Start Wire library for I2C
  Serial.begin(2000000);
  Wire.begin();

   // Set multiplexer to channel 1
  TCA9548A(1);
   setupMPU();

  // Set multiplexer to channel 2
  TCA9548A(2);
   setupMPU();

  // Set multiplexer to channel 7
  TCA9548A(7);
   setupMPU();
}
```

```
void loop() {


 // Set multiplexer to channel 1
 TCA9548A(1);
 recordAccelRegistersOne();
 recordGyroRegistersOne();
 printDataImuOne();
 delay(10);


 // Set multiplexer to channel 2
 TCA9548A(2);
 recordAccelRegistersTwo();
 recordGyroRegistersTwo();
 printDataImuTwo();
 delay(10);
 // Set multiplexer to channel 7
 TCA9548A(7);
 recordAccelRegistersSeven();
 recordGyroRegistersSeven();
 printDataImuSeven();
 delay(10);


}
void setupMPU(){
 Wire.beginTransmission(0b1101000); //This is the I2C address of the MPU (b1101000/b1101001 for AC0 low/high datasheet sec. 9.2)
 Wire.write(0x6B); //Accessing the register 6B - Power Management (Sec. 4.28)
 Wire.write(0b00000000); //Setting SLEEP register to 0. (Required; see Note on p. 9)
 Wire.endTransmission();
 Wire.beginTransmission(0b1101000); //I2C address of the MPU
 Wire.write(0x1B); //Accessing the register 1B - Gyroscope ConFigureuration (Sec. 4.4)
 Wire.write(0x00000000); //Setting the gyro to full scale +/- 250deg./s
 Wire.endTransmission();
 Wire.beginTransmission(0b1101000); //I2C address of the MPU
 Wire.write(0x1C); //Accessing the register 1C - Acccelerometer ConFigureuration (Sec. 4.5)
 Wire.write(0b00000000); //Setting the accel to +/- 2g
 Wire.endTransmission();
}
```

```
void recordAccelRegistersOne() {
  Wire.beginTransmission(0b1101000); //I2C address of the MPU
  Wire.write(0x3B); //Starting register for Accel Readings
  Wire.endTransmission();
  Wire.requestFrom(0b1101000,6); //Request Accel Registers (3B - 40)
  while(Wire.available() < 6);
  accelX = Wire.read()<<8|Wire.read(); //Store first two bytes into accelX
  accelY = Wire.read()<<8|Wire.read(); //Store middle two bytes into accelY
  accelZ = Wire.read()<<8|Wire.read(); //Store last two bytes into accelZ
  processAccelDataOne();
}

void processAccelDataOne(){
  gForceX = ((accelX-2706) / 16384.0)*9.81; //unit conversion acc(m/s^2)
  gForceY = ((accelY-71) / 16384.0)*9.81;
  gForceZ = ((accelZ+1402) / 16384.0)*9.81;
}

void recordGyroRegistersOne() {
  Wire.beginTransmission(0b1101000); //I2C address of the MPU
  Wire.write(0x43); //Starting register for Gyro Readings
  Wire.endTransmission();
  Wire.requestFrom(0b1101000,6); //Request Gyro Registers (43 - 48)
  while(Wire.available() < 6);
  gyroX = Wire.read()<<8|Wire.read(); //Store first two bytes into accelX
  gyroY = Wire.read()<<8|Wire.read(); //Store middle two bytes into accelY
  gyroZ = Wire.read()<<8|Wire.read(); //Store last two bytes into accelZ
  processGyroDataOne();
}

void processGyroDataOne() {
  rotX = ((gyroX+37) / 131.0);//*(2*3.14/360); //unit conversion ang vel(rad/s)
  rotY = ((gyroY-35) / 131.0);//*(2*3.14/360);
  rotZ = ((gyroZ+40) / 131.0);//*(2*3.14/360);
}
```

```
void recordAccelRegistersTwo() {
  Wire.beginTransmission(0b1101000); //I2C address of the MPU
  Wire.write(0x3B); //Starting register for Accel Readings
  Wire.endTransmission();
  Wire.requestFrom(0b1101000,6); //Request Accel Registers (3B - 40)
  while(Wire.available() < 6);
  accelX = Wire.read()<<8|Wire.read(); //Store first two bytes into accelX
  accelY = Wire.read()<<8|Wire.read(); //Store middle two bytes into accelY
  accelZ = Wire.read()<<8|Wire.read(); //Store last two bytes into accelZ
  processAccelDataTwo();
}

void processAccelDataTwo(){
  gForceX = ((accelX+1742) / 16384.0)*9.81;
  gForceY = ((accelY-3332) / 16384.0)*9.81;
  gForceZ = ((accelZ+897) / 16384.0)*9.81;
}

void recordGyroRegistersTwo() {
  Wire.beginTransmission(0b1101000); //I2C address of the MPU
  Wire.write(0x43); //Starting register for Gyro Readings
  Wire.endTransmission();
  Wire.requestFrom(0b1101000,6); //Request Gyro Registers (43 - 48)
  while(Wire.available() < 6);
  gyroX = Wire.read()<<8|Wire.read(); //Store first two bytes into accelX
  gyroY = Wire.read()<<8|Wire.read(); //Store middle two bytes into accelY
  gyroZ = Wire.read()<<8|Wire.read(); //Store last two bytes into accelZ
  processGyroDataTwo();
}

void processGyroDataTwo() {
  rotX = ((gyroX+123) / 131.0);//*(2*3.14/360);
  rotY = ((gyroY-4) / 131.0);//(2*3.14/360);
  rotZ = ((gyroZ-17) / 131.0);//(2*3.14/360);
}
```

```
void recordAccelRegistersSeven() {
  Wire.beginTransmission(0b1101000); //I2C address of the MPU
  Wire.write(0x3B); //Starting register for Accel Readings
  Wire.endTransmission();
  Wire.requestFrom(0b1101000,6); //Request Accel Registers (3B - 40)
  while(Wire.available() < 6);
  accelX = Wire.read()<<8|Wire.read(); //Store first two bytes into accelX
  accelY = Wire.read()<<8|Wire.read(); //Store middle two bytes into accelY
  accelZ = Wire.read()<<8|Wire.read(); //Store last two bytes into accelZ
  processAccelDataSeven();
}

void processAccelDataSeven(){
  gForceX = ((accelX+37) / 16384.0)*9.81;
  gForceY = ((accelY+1821) / 16384.0)*9.81;
  gForceZ = ((accelZ+887) / 16384.0)*9.81;
}

void recordGyroRegistersSeven() {
  Wire.beginTransmission(0b1101000); //I2C address of the MPU
  Wire.write(0x43); //Starting register for Gyro Readings
  Wire.endTransmission();
  Wire.requestFrom(0b1101000,6); //Request Gyro Registers (43 - 48)
  while(Wire.available() < 6);
  gyroX = Wire.read()<<8|Wire.read(); //Store first two bytes into accelX
  gyroY = Wire.read()<<8|Wire.read(); //Store middle two bytes into accelY
  gyroZ = Wire.read()<<8|Wire.read(); //Store last two bytes into accelZ
  processGyroDataSeven();
}

void processGyroDataSeven() {
  rotX = ((gyroX+45) / 131.0);//*(2*3.14/360);
  rotY = ((gyroY+83) / 131.0);//*(2*3.14/360);
   rotZ = ((gyroZ-113) / 131.0);//*(2*3.14/360);
}
```

```
void printDataImuOne() {
  Serial.print("Gyro-1 (rad/s)");
   Serial.print('\t');
  Serial.print(" X=");
   Serial.print('\t');
  Serial.print(rotX);
   Serial.print('\t');
  Serial.print(" Y=");
   Serial.print('\t');
  Serial.print(rotY);
   Serial.print('\t');
  Serial.print(" Z=");
   Serial.print('\t');
  Serial.print(rotZ);
   Serial.print('\t');
  Serial.print(" Accel-1 (m/s^2)");
   Serial.print('\t');
  Serial.print(" X=");
   Serial.print('\t');
  Serial.print(gForceX);
   Serial.print('\t');
  Serial.print(" Y=");
   Serial.print('\t');
  Serial.print(gForceY);
   Serial.print('\t');
  Serial.print(" Z=");
   Serial.print('\t');
  Serial.println(gForceZ);
    //Serial.print('\t');
 // Serial.println();
}



void printDataImuTwo() {
  Serial.print("Gyro-2 (rad/s)");
   Serial.print('\t');
  Serial.print(" X=");
   Serial.print('\t');
```

```
  Serial.print(rotX);
   Serial.print('\t');
  Serial.print(" Y=");
   Serial.print('\t');
  Serial.print(rotY);
   Serial.print('\t');
  Serial.print(" Z=");
   Serial.print('\t');
  Serial.print(rotZ);
   Serial.print('\t');
  Serial.print(" Accel-2 (m/s^2)");
   Serial.print('\t');
  Serial.print(" X=");
   Serial.print('\t');
  Serial.print(gForceX);
   Serial.print('\t');
  Serial.print(" Y=");
   Serial.print('\t');
  Serial.print(gForceY);
   Serial.print('\t');
  Serial.print(" Z=");
   Serial.print('\t');
  Serial.println(gForceZ);
   //Serial.print('\t');
  //Serial.println();
}


void printDataImuSeven() {
  Serial.print("Gyro-7 (rad/s)");
   Serial.print('\t');
  Serial.print(" X=");
   Serial.print('\t');
  Serial.print(rotX);
   Serial.print('\t');
  Serial.print(" Y=");
   Serial.print('\t');
  Serial.print(rotY);
```

```
 Serial.print('\t');
Serial.print(" Z=");
 Serial.print('\t');
Serial.print(rotZ);
 Serial.print('\t');
Serial.print(" Accel-7 (m/s^2)");
 Serial.print('\t');
Serial.print(" X=");
 Serial.print('\t');
Serial.print(gForceX);
 Serial.print('\t');
Serial.print(" Y=");
 Serial.print('\t');
Serial.print(gForceY);
 Serial.print('\t');
Serial.print(" Z=");
 Serial.print('\t');
Serial.println(gForceZ);
 //Serial.print('\t');
//Serial.println();
}
```

7.3 TECHNICAL DRAWING OF MODEL VESSEL