# Quaternion Attitude Estimation Kalman Filter
# Assignment 3

Dimitriy Georgiev

April 8, 2021

## 1 Assignment Description

The assignment requires the implementation of a Kalman Filter to combine data from two different types of sensors - Accelerometer and Gyroscope, in order to obtain more accureate estimate of the attitude of a vehicle.

## 2 Setup

The data for this exercise is generated by taking gyroscope and accelerometer measurements during a rolling oscillation, followed by a combined roll and pitch oscillation, and finally by a pitch oscillation, all with frequency of 0.2 Hz, amplitude of 30°, and 60 seconds duration.

## 3 Procedure

1) Transformation matrix $A$ is generated based on the gyro readings.

- Generate transformation matrix $A$:

$$A = \left( I + \frac{\Delta t}{2} \begin{pmatrix} 0 & -roll & -pitch & -yaw \\ roll & 0 & yaw & -pitch \\ pitch & -yaw & 0 & roll \\ yaw & pitch & -roll & 0 \end{pmatrix} \right)$$

2) State estimate and process covariance estimate is calculated based on the $A$ matrix, previous state, previous process covariance, and process noise.

- The state of the system is defined in quaternion format:

$$X = \begin{pmatrix} q1 \\ q2 \\ q3 \\ q4 \end{pmatrix}$$

- The process noise matrix is given to be:

$$Q = \begin{pmatrix} 10^{-4} & 0 & 0 & 0 \\ 0 & 10^{-4} & 0 & 0 \\ 0 & 0 & 10^{-4} & 0 \\ 0 & 0 & 0 & 10^{-4} \end{pmatrix}$$

- Calculate state estimate:

$$\hat{X}_t = AX_{t-1}$$

- Calculate process covariance estimate:

$$\hat{P}_t = AP_{t-1}A^T + Q$$

3) Kalman gain is calculated based on the process covariance estimate and the measurement noise:

- The measurement noise matrix is given to be:

$$R = \begin{pmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 10 \end{pmatrix}$$

- Calculate kalman gain:

$$K = \hat{P}_t(\hat{P}_t + R)^{-1}$$

4) Correction is applied by making a measurement with the accelerometer. The measured acceleration is converted from Euler angles to Quaternions.

- Convert Euler angles to Quaternions:

$$\dot{X} = \begin{pmatrix} q1 \\ q2 \\ q3 \\ q4 \end{pmatrix} = \begin{pmatrix} \cos\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\omega}{2} + \sin\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\omega}{2} \\ \sin\frac{\phi}{2}\cos\frac{\theta}{2}\cos\frac{\omega}{2} - \cos\frac{\phi}{2}\sin\frac{\theta}{2}\sin\frac{\omega}{2} \\ \cos\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\omega}{2} + \sin\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\omega}{2} \\ \cos\frac{\phi}{2}\cos\frac{\theta}{2}\sin\frac{\omega}{2} - \sin\frac{\phi}{2}\sin\frac{\theta}{2}\cos\frac{\omega}{2} \end{pmatrix}$$

*Where $\phi$, $\theta$, and $\omega$ correspond to accelerometer readings in X, Y, and Z axis.*

- Correct state and process covariance estimates:

$$X_t = \hat{X}_t + K(\dot{X}_t - \hat{X}_t)$$
$$P_t = \hat{P}_t - K\hat{P}_t$$

- Update previous state:

$$X_{t-1} = X_t$$
$$P_{t-1} = P_t$$

# 4 Implementation in Python

**0. Include necassary libraries**

[52]:
```python
import numpy as np
import scipy.constants
from math import sqrt
import pandas as pd
from numpy.linalg import inv as inverse
import matplotlib.pyplot as plt
```

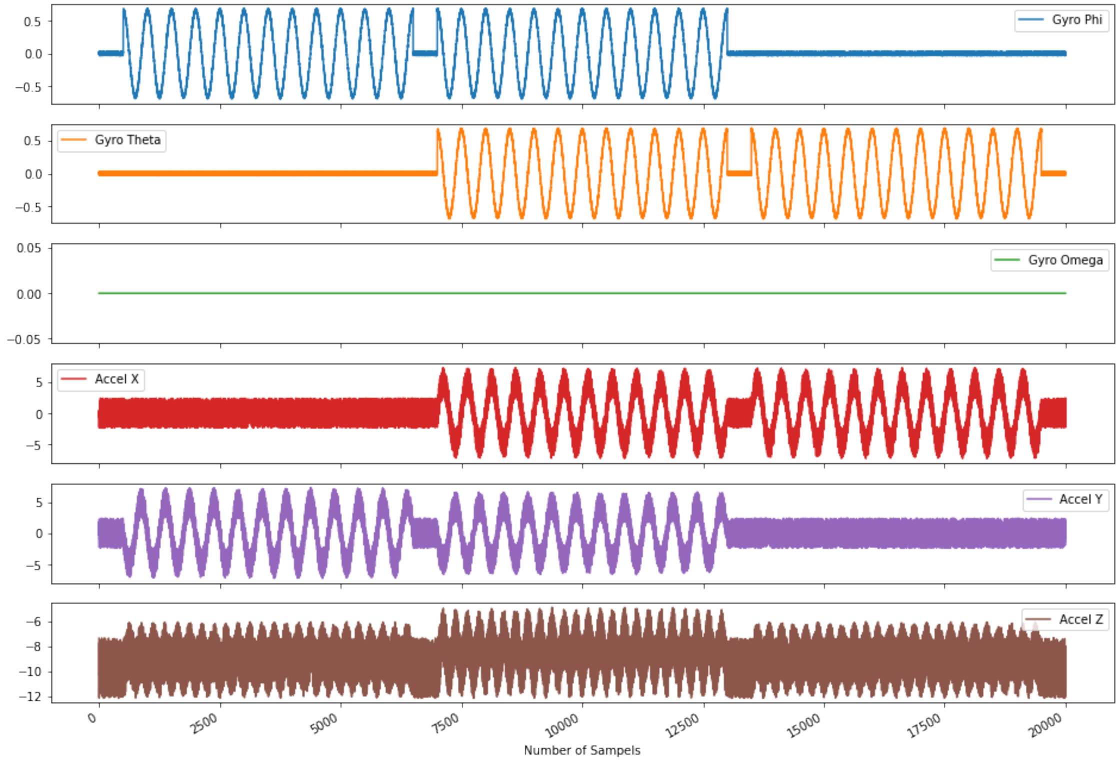
**1. Import measurement data**

[48]:
```python
data_frame = pd.read_excel("../data/KF_data_2.xlsx", engine="openpyxl")

# Collect Data
accelerometer_data = np.array([data_frame["Accel X"], data_frame["Accel Y"],␣
↪data_frame["Accel Z"]], ndmin=2).transpose()
gyro_data = np.array([(data_frame["Gyro Phi"]), (data_frame["Gyro Theta"]),␣
↪(data_frame["Gyro Omega"])], ndmin=2).transpose()

# Display last 10 values in the dataset
data_frame.tail(10)
```

[48]:

| | Gyro Phi | Gyro Theta | Gyro Omega | Accel X | Accel Y | Accel Z |
|---|---|---|---|---|---|---|
| 19991 | 0.016000 | 0.032000 | 0 | -1.324 | 1.957 | -12.100 |
| 19992 | 0.018000 | 0.011000 | 0 | 2.133 | -1.875 | -10.986 |
| 19993 | 0.016000 | -0.017000 | 0 | 0.075 | 2.366 | -11.880 |
| 19994 | 0.007857 | -0.019000 | 0 | -0.213 | -0.403 | -8.510 |
| 19995 | -0.002727 | -0.016000 | 0 | -2.252 | -0.405 | -7.975 |
| 19996 | 0.035000 | -0.014000 | 0 | -2.021 | -0.112 | -8.110 |
| 19997 | 0.032000 | -0.031000 | 0 | 1.042 | -0.610 | -11.263 |
| 19998 | 0.014000 | 0.026000 | 0 | -2.151 | -0.317 | -10.481 |
| 19999 | -0.031000 | -0.012000 | 0 | -0.774 | 0.589 | -8.794 |
| 20000 | -0.009074 | 0.006722 | 0 | 2.424 | -0.677 | -9.847 |

[70]:
```python
axes = data_frame.plot.line(subplots=True, figsize=[16,12])

plt.xlabel("Number of Sampels")
plt.show()
```




*Gyroscope data is measured in rad/s, while Accelerometer data is measured in g's*

[38]:
```python
def accelerometer_to_attitude(accelerometer_x, accelerometer_y, accelerometer_z):
    pitch = np.arcsin(accelerometer_x / g)
    roll = np.arcsin(accelerometer_y / (g * np.cos(pitch)))
    yaw = 0

    return roll, pitch, yaw
```

[39]:
```python
def euler_angles_to_quaternions(roll, pitch, yaw):
    q_1 = np.cos(roll / 2) * np.cos(pitch / 2) * np.cos(yaw / 2) + np.sin(roll /␣
↪2) * np.sin(pitch / 2) * np.sin(yaw / 2)
    q_2 = np.sin(roll / 2) * np.cos(pitch / 2) * np.cos(yaw / 2) + np.cos(roll /␣
↪2) * np.sin(pitch / 2) * np.sin(yaw / 2)
    q_3 = np.cos(roll / 2) * np.sin(pitch / 2) * np.cos(yaw / 2) + np.sin(roll /␣
↪2) * np.cos(pitch / 2) * np.sin(yaw / 2)
    q_4 = np.cos(roll / 2) * np.cos(pitch / 2) * np.sin(yaw / 2) + np.sin(roll /␣
↪2) * np.sin(pitch / 2) * np.cos(yaw / 2)

    return q_1, q_2, q_3, q_4
```

[40]:
```
def quoternions_to_euler_angles(q_1, q_2, q_3, q_4):
    phi = np.degrees(np.arctan2(2 * (q_1 * q_2 + q_3 * q_4), 1 - 2 * (q_2 ** 2 +
 ↪q_3 ** 2)))
    theta = np.degrees(np.arcsin(2 * (q_1 * q_3 - q_4 * q_2)))
    omega = np.degrees(np.arctan2(2 * (q_1 * q_4 + q_2 * q_3), 1 - 2 * (q_3 ** 2
 ↪+ q_4 ** 2)))

    return phi, theta, omega
```

[41]:
```
def calculate_transormation_martix_A(gyro_phi, gyro_theta, gyro_omega):
    A = np.array(
        np.identity(4)
        + (delta_t / 2)
        * np.array(
            [
                [0, -gyro_phi, -gyro_theta, -gyro_omega],
                [gyro_phi, 0, gyro_omega, -gyro_theta],
                [gyro_theta, -gyro_omega, 0, gyro_phi],
                [gyro_omega, gyro_theta, -gyro_phi, 0],
            ]
        )
    )
    return A
```

**2. Define initial conditions:**

[53]:
```
# Constants
g = scipy.constants.g  # Standard acceleration of gravity
delta_t = 0.01  # Time step [s], 100Hz

# 4x4 transformation matrices are needed because we are working with quaternions
H = np.identity(4)
C = np.identity(4)

# Initialize covariance matrices
Q_process_noise_matrix = np.array([[10 ** -4, 0, 0, 0], [0, 10 ** -4, 0, 0], [0,
 ↪0, 10 ** -4, 0], [0, 0, 0, 10 ** -4]])
R_measurement_noise_matrix = np.array([[10, 0, 0, 0], [0, 10, 0, 0], [0, 0, 10,
 ↪0], [0, 0, 0, 10]])

q_1, q_2, q_3, q_4 = euler_angles_to_quaternions(0, 0, 0)
prev_state = np.array([q_1, q_2, q_3, q_4], ndmin=2).transpose()
prev_process_covariance = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0],
 ↪[0, 0, 0, 1]])

time = np.linspace(0, 200.1, num=20001)
kalman_corrected_phi = []
```

```
kalman_corrected_theta = []
kalman_corrected_omega = []
```

**4. Begin the recursive algorithm:**

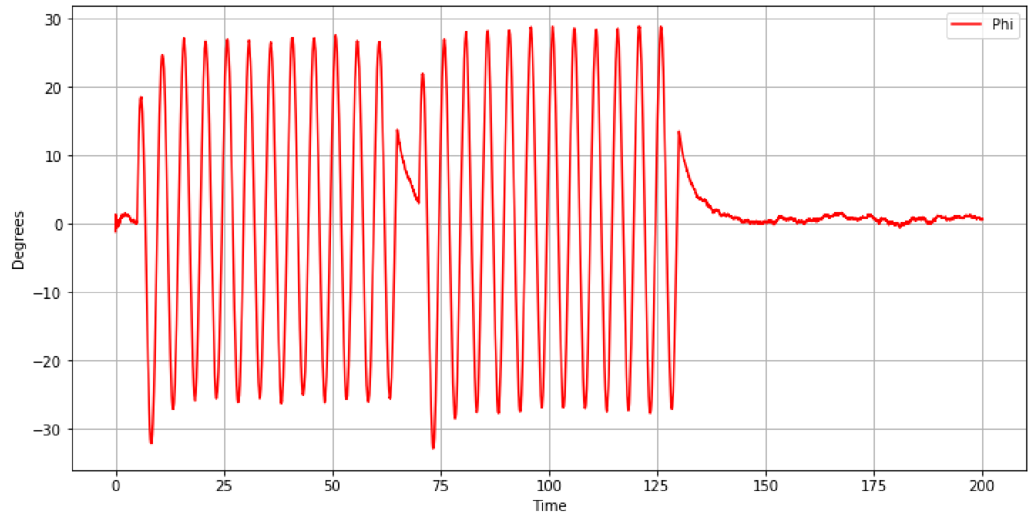
[54]:
```
for accelerometer_measurement, gyro_measurement in zip(accelerometer_data,␣
↪gyro_data):

    # Create a new matrix 'A' with angular velocities from gyro data
    A = calculate_transormation_martix_A(gyro_measurement[0],␣
↪gyro_measurement[1], gyro_measurement[2])

    # Calculate the state estimate and process covariance estimate
    state_estimate = A @ prev_state
    process_covariance_estimate = A @ prev_process_covariance @ A.transpose() +␣
↪Q_process_noise_matrix
    # process_covariance_estimate = np.diag(np.diag(process_covariance_estimate))

    # Compute the Kalman gain
    kalman_gain = (
        process_covariance_estimate
        @ H
        @ inverse(H @ process_covariance_estimate @ H.transpose() +␣
↪R_measurement_noise_matrix)
    )

    # Convert measured acceleration to Euler angles
    accelerometer_roll, accelerometer_pitch, accelerometer_yaw =␣
↪accelerometer_to_attitude(
        accelerometer_measurement[0], accelerometer_measurement[1],␣
↪accelerometer_measurement[2]
    )

    # Convert then to Quaternions
    q_1, q_2, q_3, q_4 = euler_angles_to_quaternions(accelerometer_roll,␣
↪accelerometer_pitch, accelerometer_yaw)
    Y_measurement_matrix = np.array([q_1, q_2, q_3, q_4], ndmin=2).transpose()

    # Compute the measurement
    state_measurement = C @ Y_measurement_matrix + 0  # Assume noise in␣
↪calculating the measurement is 0

    # Correct the state estimate and process covariance estimate
    new_state = state_estimate + kalman_gain @ (state_measurement - H @␣
↪state_estimate)
```

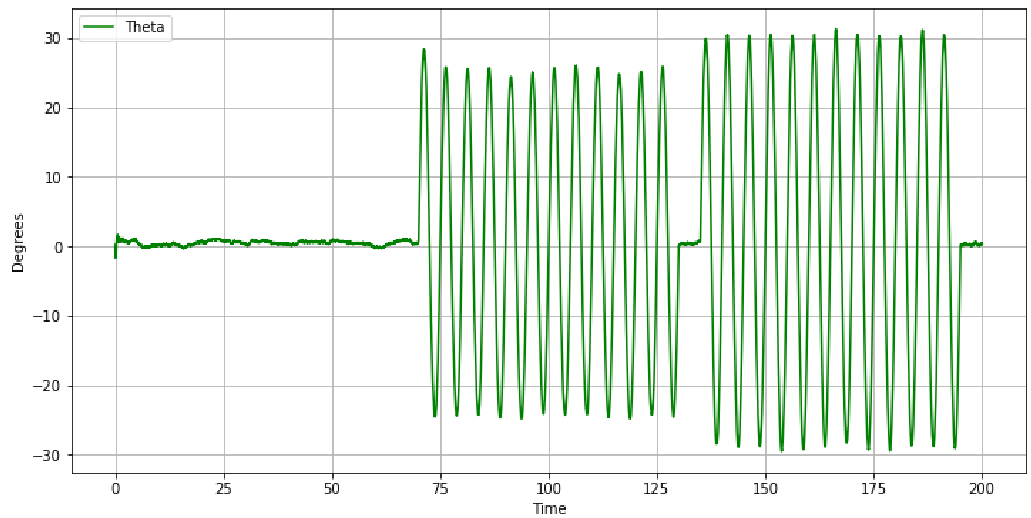
```
    new_process_covariance = process_covariance_estimate - kalman_gain @ H @
↪process_covariance_estimate

    # Update previous state
    prev_state = new_state
    prev_process_covariance = new_process_covariance

    # Append data to list and repeat
    new_phi, new_theta, new_omega = quoternions_to_euler_angles(
        new_state.take(0), new_state.take(1), new_state.take(2), new_state.
↪take(3)
    )

    # Collect data for comparison
    measured_phi, measured_theta, measured_omega =
↪quoternions_to_euler_angles(q_1, q_2, q_3, q_4)

    kalman_corrected_phi.append(new_phi)
    kalman_corrected_theta.append(new_theta)
    kalman_corrected_omega.append(new_omega)
```

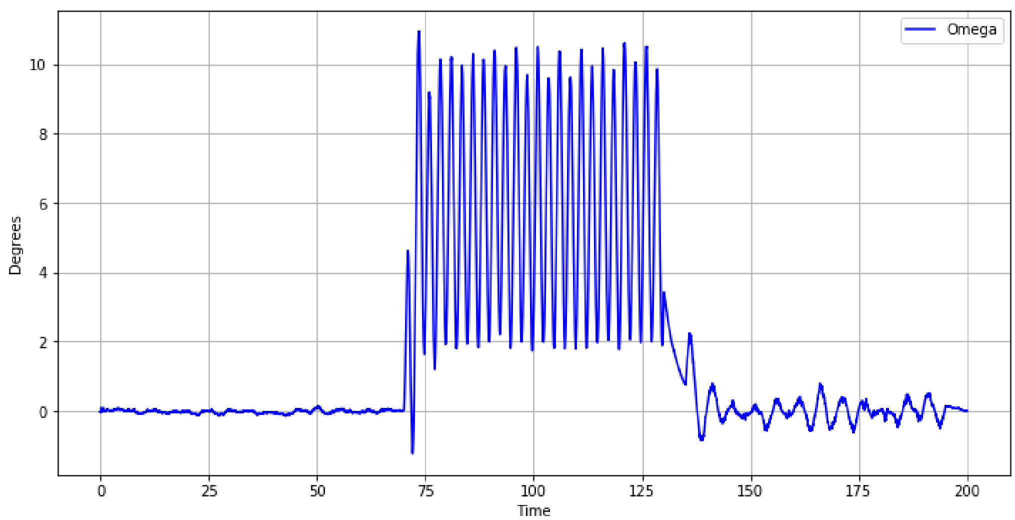
**5. Plot the corrected data via the Kalman Filter:**

[73]:

```
dictionary = {
        "Time": time[:],
        "Phi": kalman_corrected_phi,
        "Theta": kalman_corrected_theta,
        "Omega": kalman_corrected_omega,
    }
kalman_data = pd.DataFrame(data=dictionary)
kalman_data.plot(x="Time", y="Phi", grid=True, color="r", figsize=[12,6])
plt.ylabel("Degrees")
kalman_data.plot(x="Time", y="Theta", grid=True, color="g", figsize=[12,6])
plt.ylabel("Degrees")
kalman_data.plot(x="Time", y="Omega", grid=True, color="b", figsize=[12,6])
plt.ylabel("Degrees")
plt.show()
```


Phi
30
20
10
0
−10
−20
−30
Degrees
0
25
50
75
100
125
150
175
200
Time


Theta
30
20
10
0
−10
−20
−30
Degrees
0
25
50
75
100
125
150
175
200
Time


Omega
10
8
6
4
2
0
Degrees
0
25
50
75
100
125
150
175
200
Time