ROBOTICS - MINI PROJECT NUMBER 1

Arvin Mohammadi
student number: 810698303

October 25, 2022

Contents

0 Introduction

In this mini project we learn and use Arduino for extracting and manipulating data from a sensor. The topics covered are:

- Introduction to sensors (MPU-6050 and MPU-9250)
- Extracting data from the two sensors
- Calibration
- Complementary Filters
- Kalman Filters
- Arduino with python (VPython)

In the Introduction chapter, all of the items named above are going to be lightly explained.

0.1 An Introduction to Sensors

These days motion sensors have a lot of use. For instance they are used in smartphones, airplanes, cars, etc. These motion sensors most usually are known as gyroscope, but there is lot more to it than just that.

0.1.1 MPU-6050

The MPU-6050 is one of the most famous motion sensors, and it is used for measurment of **linear acceleration and angular velocity**. Some of the advantages of this sensor are **low energy use, low Price and relatively good results**

0.1.2 MPU-9250

MPU-9250 Module is a 9-axis sensor, which are:

- 3 Axes for gyroscope
- 3 Axes for acceleration
- 3 Axes for magnetic field

0.2 Extracting Data

There are two major ways for extracting data from sensors that are explained below:

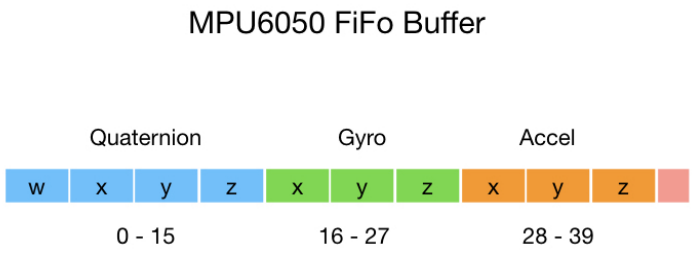
0.2.1 First Method

In the first method, the data is extracted with opperations applied on the raw data. A few of which are: integrating on time intervals, lowering the offsets, level sensitivity.

0.2.2 Second Method

advanced-digital-motion-processing-internal-system is employed for this second method. This system features an internal buffer where is stores data such as gyroscope, acceleration and other calculated data.

Figure 1: FiFo for MPU-6050



With the use of FiFo concept and employing interruption, the quaternion measurements may be derived.

0.3 Calibration

The definition of calibration is: Changing the frame-of-reference from body, that the sensor is located, to the external environment that angles have meaning. in order to nulify the environment effects, the sensor MPU-9250, must be rotated on 8 independant axes. This is very similar to the sensor MPU-6050, with the slight alternation of how magnetometer in the MPU-6050 is calibrated.

0.4 Filtering

In signal-processing, filtering is a process or device which is used for noise cancellation.

0.4.1 Complementary

This filter includes a high-pass-filter on the accelerometer, and a low-pass-filter on gyroscope data. The formula can be described as:

$$Angle = \alpha_1(angle + gyroscopedata.dt) + \alpha_2(accelerometerdata) \quad (1)$$

that $\alpha_1 = 0.98$, $\alpha_2 = 0.02$ of course these values for α_1 and α_2 can be tuned to fit the best result.

0.4.2 Kalman

For recursive estimation of system state, Kalman filter is applied. It predicts the next output with the given input.

0.5 Arduino and Python

The VPython package simplify visualization and animation in 3D space.

1 Question 1

USING a pre-defined Arduino File, qutenrion, roll, pitch and yaw angles should be extracted from MPU-6050 sensor. the method used for this task is the second of data extraction explained in Sec. 0.2.2 The needed libraries for running this code are in **This Link**.

1.1 Desired Outputs

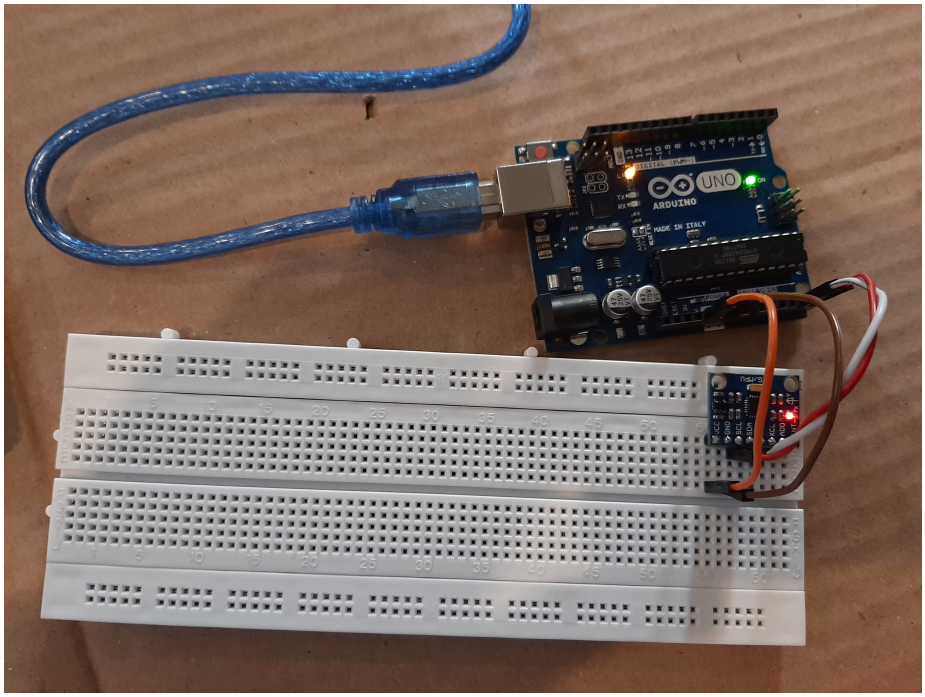
The desired outputs are listed below:

- With help of the mentioned code, output quaternion values and roll, pitch and yaw angles in the serical monitor. Make a video from the setup and result.
- plot roll, pitch and yaw angles in real-time, one time using the plotter serial and another time with the use of Serial-Plot-Software. compare the outcome and make a video from the results.

1.2 Data Extraction

The first thing required for outputing the desired data, is setting up Arduino Libraries and simulating the sensor. The first portion of the process is easily obtainable using the reference Vid. 1. (Fig. 2)

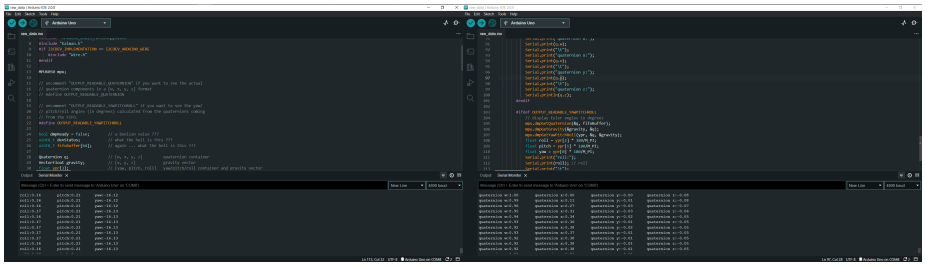
Figure 2: Arduino Setup



Arduino Setup with MPU-6050 connected

Raw data is written in Serial Monitor as shown below:

Figure 3: Arduino Serial Monitor



(a) Serial Monitor - pitch, roll, yaw (b) Serial Monitor - w, x, y, z

both sets of raw data are extracted by the given code

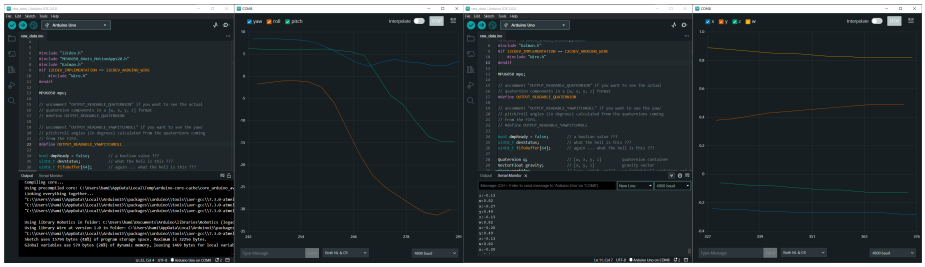
1.3 Data Plot

With the mentioned code, raw data is easily obtainable, hence the plot can be drawn in Ardunio Plotter, SerialPlot software and python.

1.3.1 Arduino plotter

Inside the Arduino plotter, it is needed to follow a certain syntax to get the names and values correctly on the board. (Fig. 4)

Figure 4: Arduino Plotter



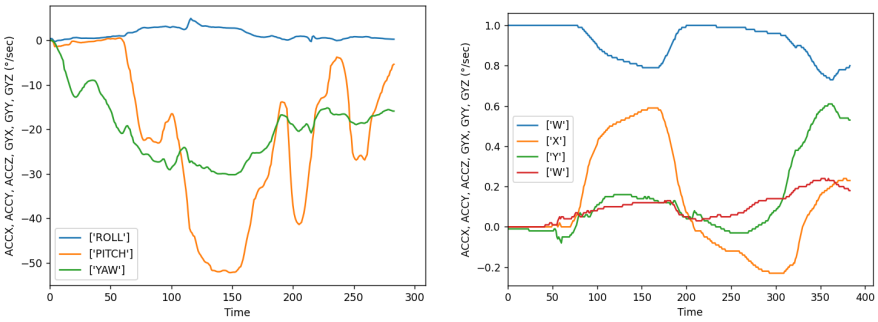
(a) Arduino Plotter - pitch, roll, yaw (b) Arduino Plotter - w, x, y, z

both sets of raw data are plotted by the given code

1.3.2 Python Plotting

For this part, first it was necessary to upload the code from arduino to the arduino UNO kit to generate the data. Then access the generating data from python and plot the data using matplotlib and drawnow packages in real-time

Figure 5: Python Plotter



(a) Python Plotter - pitch, roll, yaw

(b) Python Plotter - w, x, y, z

both sets of raw data are plotted by the given code in Appendix A - Sect. 5.1

2 Question 2

USING the explaination from the Sub-sect. 0.4 it is required that you implement The following filters on the raw data:

2.1 Desired Output

Implementation of

1. Kalman filter
2. Complementary filter (first degree)
3. Complementary filter (second degree)

it is also necessary to compare these filters to find the best performing filter with the following standards:

1. Complexity
2. Accuracy
3. Speed
4. Performance in rapid changes

2.2 Filter Implementation - Kalman

With the median method, Kalman filter takes $2n + 1$ data as input

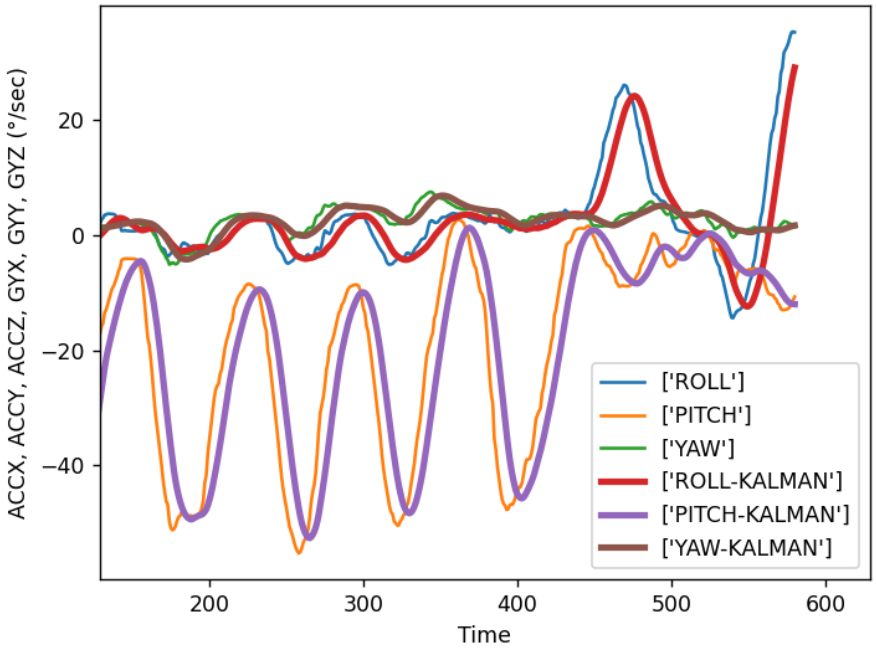
$$\ldots, \quad data_{n-2}, \quad data_{n-1}, \quad data_n, \quad data_{n+1}, \quad data_{n+2}, \quad \ldots$$

that $data_n$ is the current data and gives

$$\sum(data_i)/len(data)$$

as an output. This results in noise cancellation if there are no spikes that heavily impact the filter. the implementation for this filter and the rest of the filters can be seen in Appendix A - Sect 5.1 and the resulting plot is shown in Fig. 6

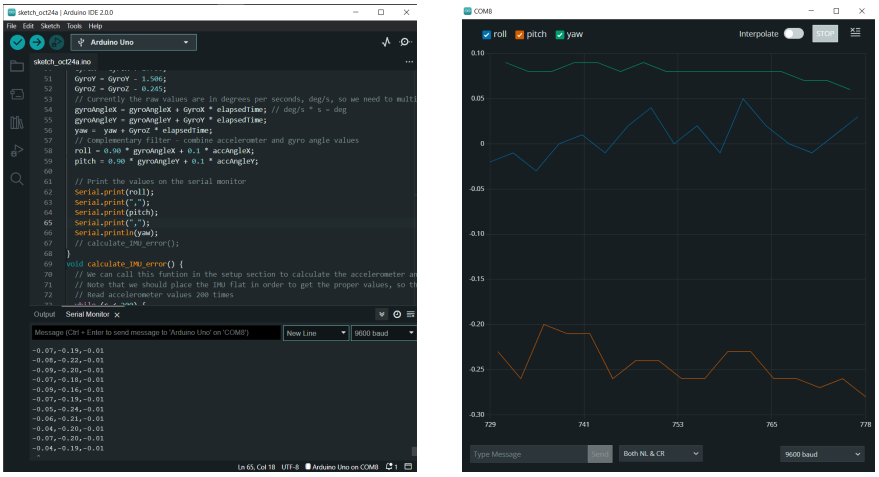
Figure 6: Kalman filter plot



2.3 Filter Implementation - Complementary 1st Degree

This filter is not suited for this experiment. Mainly because it envolves integrating over gyroscope extraced data (which is angular velocity). This data has some offset, and the offset is not something permanent, meaning it changes over time. This results in a noticable error as the time goes by. In conclusion, I don't like this method and I won't be using it from now on because a stacking-up error, is something not at all suited for a video-game. The results can be seen in Fig. 7

Figure 7: Complementary Filter



(a) Serial Monitor (b) Arduino Plotter

offset error is built up over time.

3 Question 3

THE problem at hand is to process the raw data into rotation matrices

3.1 Desired Output

Rotation matrix calculated from:

1. Roll, pitch, yaw values
2. Quaternion

using python.

3.2 Rotation Matrix - from Roll, Pitch, Yaw Values

Since the raw data can already be read from python all that is required to do is to calculate rotation matrix from roll, pitch, yaw values. the following is the formula to do so:

1. Yaw is a counter-clock-wise rotation of α about $z - axis$. The rotation matrix of yaw is given by:

$$R_z(\alpha) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2. Pitch is a counter-clock-wise rotation of β about $y - axis$. The rotation matrix is given by:

$$R_y(\beta) = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{pmatrix}$$

3. Roll is a counter-clock-wise rotation of γ about $x - axis$. The rotation matrix is given by:

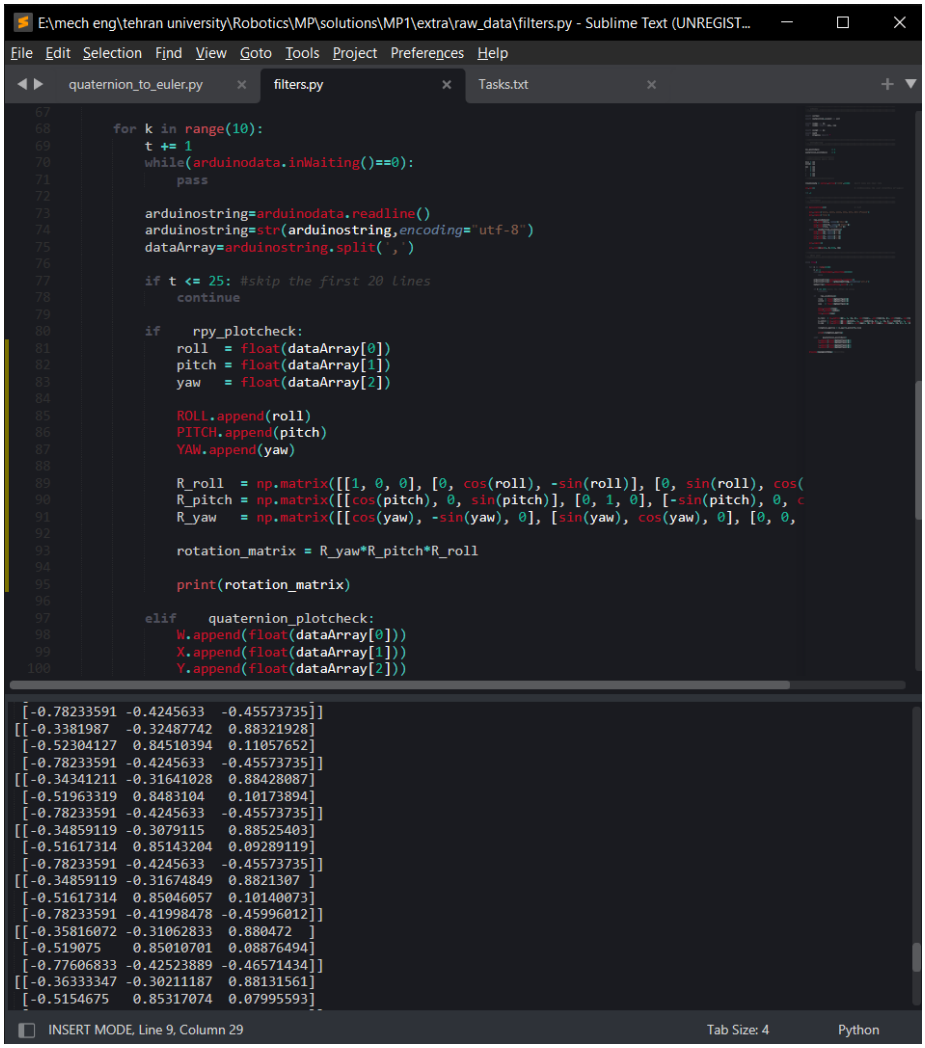
$$R_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{pmatrix}$$

For combining all the three, first the rotation of roll, then pitch and in the end yaw is performed. So the final rotation matrix is going to look like:

$$R = R(\alpha, \beta, \gamma) = R_z(\alpha) R_y(\beta) R_x(\gamma)$$

The python code to this part is shown in Appendix A - Sect. 5.3 and the results are shown at Fig. 8

Figure 8: Rotation Matrix



3.3 Rotation Matrix - from Quaternion Values

If we map the quaternion values as below:

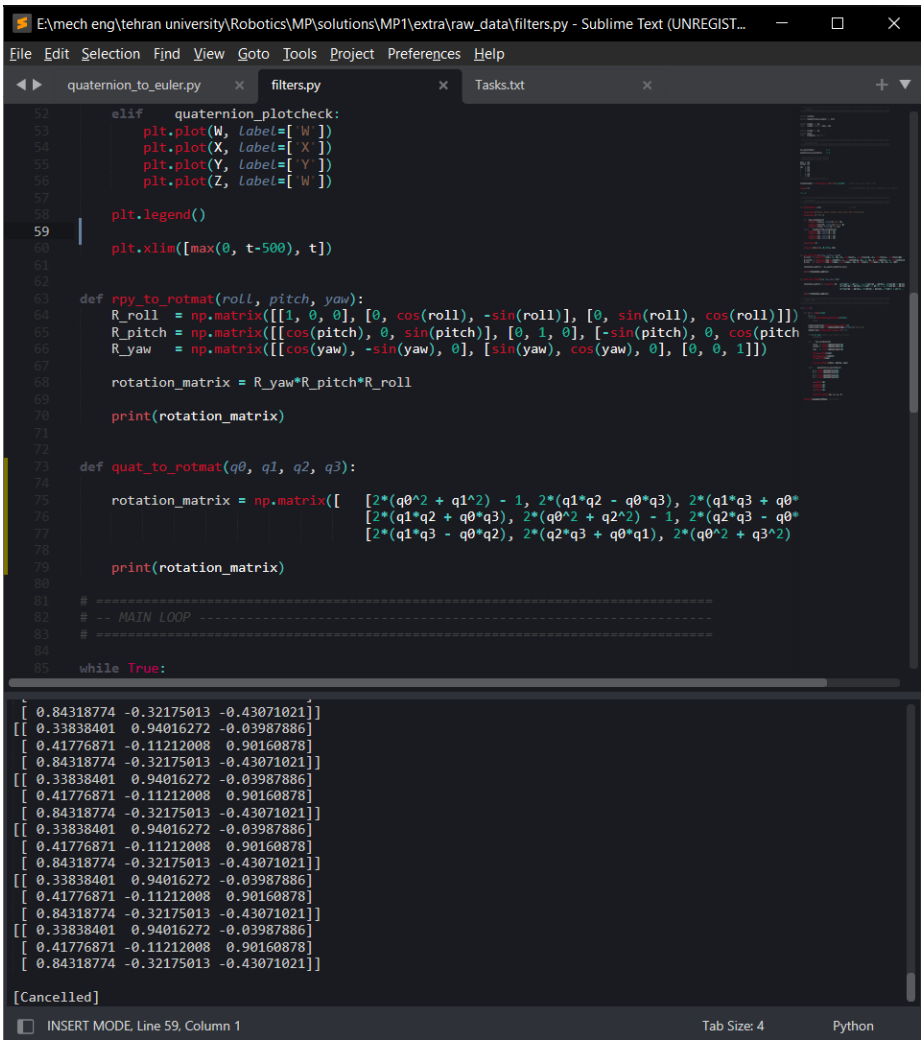
- $w \equiv q_0$
- $x \equiv q_1$
- $y \equiv q_2$
- $z \equiv q_3$

then rotation matrix can be calculated from the following equation:

$$R_{Quaternion} = \begin{pmatrix} 2(q_0^2 + q_1^2) - 1 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & 2(q_0^2 + q_2^2) - 1 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & 2(q_0^2 + q_3^2) - 1 \end{pmatrix}$$

And as the previous part, the code can be founded in Appendix A - Sect. 5.3 and the results are shown in Fig. 9

Figure 9: Rotation Matrix

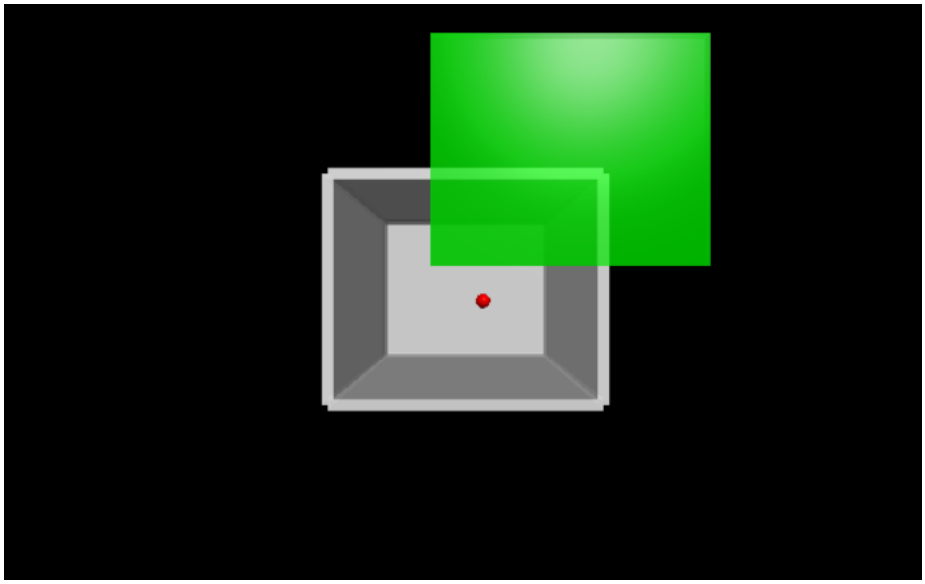


4 Question 4

In this section, the visulization of the game pong should be built in Vpython package: (Fig. 10) The algorithem for making the game is pretty simple.

- the first step is to make the model room and ball and racket
- the next step is to build the collision logic. (meaning based on the ball position, if it hits the walls or the racket it bounces back)
- the final step is to import the arduino kit data from Arduino code (previously explained) and feed it as racket movement controller.
- BOOM you got a game !!!

Figure 10: Game



5 Appendix A - Raw Code

In this Appendix the raw code for Arduino and python is written.

5.1 Raw Data - Python Code

```
# ================================================================================
# -- IMPORTS -------------------------------------------------------------------
# ================================================================================

import serial
import matplotlib.pyplot as plt

import numpy as np
from   numpy import sin, cos

import pylab as py
import math
from   drawnow import *

# ================================================================================
# -- PREPARATION  --------------------------------------------------------------
# ================================================================================

rpy_plotcheck  = 1
quaternion_plotcheck  = 0

# ========================
# initializing empty lists
ROLL = []
PITCH= []
YAW  = []
W    = []
X    = []
Y    = []
Z    = []

ROLL_KALMAN  = []
PITCH_KALMAN  = []
YAW_KALMAN = []
W_KALMAN  = []
X_KALMAN  = []
Y_KALMAN  = []
Z_KALMAN  = []
# ========================

arduinodata = serial.Serial('COM8',9600)  #port name and baud rate

plt.ion()  # initializing the user interface of pyplot
```

```
t = -1

# ============================================================================
# -- FUNCTIONS ---------------------------------------------------------------
# ============================================================================

def makeplotting(): # PLOT

plt.ylabel('ACCX, ACCY, ACCZ, GYX, GYY, GYZ (°/sec)')
plt.xlabel('Time')

if  rpy_plotcheck:
plt.plot(ROLL, label=['ROLL'])
plt.plot(PITCH, label=['PITCH'])
plt.plot(YAW, label=['YAW'])
plt.plot(ROLL_KALMAN, label=['ROLL-KALMAN'], linewidth=3)
plt.plot(PITCH_KALMAN, label=['PITCH-KALMAN'], linewidth=3)
plt.plot(YAW_KALMAN, label=['YAW-KALMAN'], linewidth=3)

elif  quaternion_plotcheck:
plt.plot(W, label=['W'])
plt.plot(X, label=['X'])
plt.plot(Y, label=['Y'])
plt.plot(Z, label=['W'])
plt.plot(W_KALMAN, label=['W_KALMAN'], linewidth=3)
plt.plot(X_KALMAN, label=['X_KALMAN'], linewidth=3)
plt.plot(Y_KALMAN, label=['Y_KALMAN'], linewidth=3)
plt.plot(Z_KALMAN, label=['Z_KALMAN'], linewidth=3)

plt.legend()

plt.xlim([max(0, t-500), t])


def rpy_to_rotmat(roll, pitch, yaw):
R_roll  = np.matrix([[1, 0, 0], [0, cos(roll), -sin(roll)], [0, sin(roll), cos(roll)]])
R_pitch = np.matrix([[cos(pitch), 0, sin(pitch)], [0, 1, 0], [-sin(pitch), 0, cos(pitch)]]
R_yaw = np.matrix([[cos(yaw), -sin(yaw), 0], [sin(yaw), cos(yaw), 0], [0, 0, 1]])

rotation_matrix = R_yaw*R_pitch*R_roll

print(rotation_matrix)


def quat_to_rotmat(q0, q1, q2, q3):

rotation_matrix = np.matrix([ [2*(q0^2 + q1^2) - 1, 2*(q1*q2 - q0*q3), 2*(q1*q3 + q0*q2)],
[2*(q1*q2 + q0*q3), 2*(q0^2 + q2^2) - 1, 2*(q2*q3 - q0*q1)],
[2*(q1*q3 - q0*q2), 2*(q2*q3 + q0*q1), 2*(q0^2 + q3^2) - 1]])
```

```
print(rotation_matrix)


# ============================================================================
# -- FILTERS -----------------------------------------------------------------
# ============================================================================

def kalman(data):
"""
the data must be 1D and the dimension shoulh be equal to (2n+1)
"""
data = np.array(data)
return float(np.sum(data)/np.size(data))

# ============================================================================
# -- MAIN LOOP ---------------------------------------------------------------
# ============================================================================


# ==========================================
# temp values initilization
temp_value  = 15
roll_kalman_temp  = np.zeros(temp_value)
pitch_kalman_temp  = np.zeros(temp_value)
yaw_kalman_temp  = np.zeros(temp_value)
w_kalman_temp  = np.zeros(temp_value)
x_kalman_temp  = np.zeros(temp_value)
y_kalman_temp  = np.zeros(temp_value)
z_kalman_temp  = np.zeros(temp_value)
# ==========================================

while True:

for k in range(10):
t += 1
while(arduinodata.inWaiting()==0):
pass

arduinostring=arduinodata.readline()
arduinostring=str(arduinostring,encoding="utf-8")
dataArray=arduinostring.split(',')

if t <= 48: #skip the first 20 lines
continue



if    rpy_plotcheck:
```

```
roll  = float(dataArray[0])
pitch = float(dataArray[1])
yaw   = float(dataArray[2])

roll_kalman_temp[-t%temp_value]  = roll
pitch_kalman_temp[-t%temp_value]  = pitch
yaw_kalman_temp[-t%temp_value]  = yaw

ROLL.append(roll)
PITCH.append(pitch)
YAW.append(yaw)

ROLL_KALMAN.append(kalman(roll_kalman_temp))
PITCH_KALMAN.append(kalman(pitch_kalman_temp))
YAW_KALMAN.append(kalman(yaw_kalman_temp))


rpy_to_rotmat(roll, pitch, yaw)

elif    quaternion_plotcheck:
w = float(dataArray[0])
x = float(dataArray[1])
y = float(dataArray[2])
z = float(dataArray[3])

W.append(w)
X.append(x)
Y.append(y)
Z.append(z)

quat_to_rotmat(w, x, y, z)

drawnow(makeplotting) #plotting
```

5.2 Raw Data - Arduino Code

```
/* Read quaternion and roll, pitch, yaw from MPU6050
* Robotics Course semester fall 2022 _ MiniProject #1
*/


#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
#include "Wire.h"
#endif

MPU6050 mpu;


// uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
// quaternion components in a [w, x, y, z] format
//#define OUTPUT_READABLE_QUATERNION

// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO.
#define OUTPUT_READABLE_YAWPITCHROLL

bool dmpReady = false;
uint8_t devStatus;
uint8_t fifoBuffer[64];

Quaternion q;                 // [w, x, y, z]         quaternion container
VectorFloat gravity;          // [x, y, z]            gravity vector
float ypr[3];                 // [yaw, pitch, roll]   yaw/pitch/roll container and gravity



// ================================================================
//                          INITIAL SETUP
// ================================================================

void setup() {
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
Wire.begin();
Wire.setClock(400000);
#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
Fastwire::setup(400, true);
#endif

Serial.begin(9600);
```

```
while (!Serial);

Serial.println(F("Initializing I2C devices..."));
mpu.initialize();

Serial.println(F("Testing device connections..."));
Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 conn

Serial.println(F("Initializing DMP..."));
devStatus = mpu.dmpInitialize();

mpu.setXGyroOffset(220);
mpu.setYGyroOffset(76);
mpu.setZGyroOffset(-85);
mpu.setZAccelOffset(1788);

if (devStatus == 0) {
mpu.CalibrateAccel(6);
mpu.CalibrateGyro(6);
mpu.PrintActiveOffsets();
Serial.println(F("Enabling DMP..."));
mpu.setDMPEnabled(true);

dmpReady = true;
}
else {
Serial.print(F("DMP Initialization failed (code "));
Serial.print(devStatus);
Serial.println(F(")"));
}
}


// ================================================================
//                          MAIN PROGRAM LOOP
// ================================================================

void loop() {
if (!dmpReady) return;
// read a packet from FIFO
if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) {
#ifdef OUTPUT_READABLE_QUATERNION
// display quaternion values in easy matrix form: w x y z
mpu.dmpGetQuaternion(&q, fifoBuffer);
Serial.print(q.w);
Serial.print(",");
Serial.print(q.x);
Serial.print(",");
Serial.print(q.y);
Serial.print(",");
```

```
Serial.print(q.z);
Serial.print("\r\n");
#endif

#ifdef OUTPUT_READABLE_YAWPITCHROLL
// display Euler angles in degrees
mpu.dmpGetQuaternion(&q, fifoBuffer);
mpu.dmpGetGravity(&gravity, &q);
mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
float roll = ypr[2] * 180/M_PI;
float pitch = ypr[1] * 180/M_PI;
float yaw = ypr[0] * 180/M_PI;
Serial.print(roll); // roll
Serial.print(",");
Serial.print(pitch); // pitch
Serial.print(",");
Serial.print(yaw); // yaw
Serial.print("\r\n");
#endif
}
}
```

5.3 Rotation Matrix Calculation - Python Code

In this part Rotation matrix is calculated from

- Quaternion values
- roll, pitch, yaw values

using python:

```
def rpy_to_rotmat(roll, pitch, yaw):
R_roll  = np.matrix([[1, 0, 0], [0, cos(roll), -sin(roll)], [0, sin(roll), cos(roll)]])
R_pitch = np.matrix([[cos(pitch), 0, sin(pitch)], [0, 1, 0], [-sin(pitch), 0, cos(pitch)]]
R_yaw = np.matrix([[cos(yaw), -sin(yaw), 0], [sin(yaw), cos(yaw), 0], [0, 0, 1]])

rotation_matrix = R_yaw*R_pitch*R_roll

print(rotation_matrix)


def quat_to_rotmat(q0, q1, q2, q3):

rotation_matrix = np.matrix([ [2*(q0^2 + q1^2) - 1, 2*(q1*q2 - q0*q3), 2*(q1*q3 + q0*q2)],
[2*(q1*q2 + q0*q3), 2*(q0^2 + q2^2) - 1, 2*(q2*q3 - q0*q1)],
[2*(q1*q3 - q0*q2), 2*(q2*q3 + q0*q1), 2*(q0^2 + q3^2) - 1]])

print(rotation_matrix)
```

5.4 Complementary Filter - Arduion Code

This is the Arduino code for implementing complementary filter

```
/*
Arduino and MPU6050 Accelerometer and Gyroscope Sensor Tutorial
by Dejan, https://howtomechatronics.com
*/
#include <Wire.h>
const int MPU = 0x68; // MPU6050 I2C address
float AccX, AccY, AccZ;
float GyroX, GyroY, GyroZ;
float accAngleX, accAngleY, gyroAngleX, gyroAngleY, gyroAngleZ;
float roll, pitch, yaw;
float AccErrorX, AccErrorY, GyroErrorX, GyroErrorY, GyroErrorZ;
float elapsedTime, currentTime, previousTime;
int c = 0;
void setup() {
Serial.begin(9600);
Wire.begin();                      // Initialize comunication
Wire.beginTransmission(MPU);       // Start communication with MPU6050 // MPU=0x68
Wire.write(0x6B);                  // Talk to the register 6B
Wire.write(0x00);                  // Make reset - place a 0 into the 6B register
Wire.endTransmission(true);        //end the transmission
// Call this function if you need to get the IMU error values for your module
calculate_IMU_error();
delay(20);
}
void loop() {
// === Read acceleromter data === //
Wire.beginTransmission(MPU);
Wire.write(0x3B); // Start with register 0x3B (ACCEL_XOUT_H)
Wire.endTransmission(false);
Wire.requestFrom(MPU, 6, true); // Read 6 registers total, each axis value is stored in 2
//For a range of +-2g, we need to divide the raw values by 16384, according to the datashe
AccX = (Wire.read() << 8 | Wire.read()) / 16384.0; // X-axis value
AccY = (Wire.read() << 8 | Wire.read()) / 16384.0; // Y-axis value
AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0; // Z-axis value
// Calculating Roll and Pitch from the accelerometer data
accAngleX = (atan(AccY / sqrt(pow(AccX, 2) + pow(AccZ, 2))) * 180 / PI) - 0.58; // AccErro
accAngleY = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow(AccZ, 2))) * 180 / PI) + 1.58; // Ac
// === Read gyroscope data === //
previousTime = currentTime;        // Previous time is stored before the actual time read
currentTime = millis();            // Current time actual time read
elapsedTime = (currentTime - previousTime) / 1000; // Divide by 1000 to get seconds
Wire.beginTransmission(MPU);
Wire.write(0x43); // Gyro data first register address 0x43
Wire.endTransmission(false);
Wire.requestFrom(MPU, 6, true); // Read 4 registers total, each axis value is stored in 2
GyroX = (Wire.read() << 8 | Wire.read()) / 131.0; // For a 250deg/s range we have to divid
GyroY = (Wire.read() << 8 | Wire.read()) / 131.0;
```

```
GyroZ = (Wire.read() << 8 | Wire.read()) / 131.0;
// Correct the outputs with the calculated error values offset
GyroX = GyroX + 1.766;
GyroY = GyroY - 1.506;
GyroZ = GyroZ - 0.245;
// Currently the raw values are in degrees per seconds, deg/s, so we need to multiply by s
gyroAngleX = gyroAngleX + GyroX * elapsedTime; // deg/s * s = deg
gyroAngleY = gyroAngleY + GyroY * elapsedTime;
yaw =  yaw + GyroZ * elapsedTime;
// Complementary filter - combine acceleromter and gyro angle values
roll = 0.96 * gyroAngleX + 0.04 * accAngleX;
pitch = 0.96 * gyroAngleY + 0.04 * accAngleY;

// Print the values on the serial monitor
Serial.print(roll);
Serial.print("/");
Serial.print(pitch);
Serial.print("/");
Serial.println(yaw);
// calculate_IMU_error();
}
void calculate_IMU_error() {
// We can call this funtion in the setup section to calculate the accelerometer and gyro d
// Note that we should place the IMU flat in order to get the proper values, so that we th
// Read accelerometer values 200 times
while (c < 200) {
Wire.beginTransmission(MPU);
Wire.write(0x3B);
Wire.endTransmission(false);
Wire.requestFrom(MPU, 6, true);
AccX = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
AccY = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
AccZ = (Wire.read() << 8 | Wire.read()) / 16384.0 ;
// Sum all readings
AccErrorX = AccErrorX + ((atan((AccY) / sqrt(pow((AccX), 2) + pow((AccZ), 2))) * 180 / PI)
AccErrorY = AccErrorY + ((atan(-1 * (AccX) / sqrt(pow((AccY), 2) + pow((AccZ), 2))) * 180
c++;
}
//Divide the sum by 200 to get the error value
AccErrorX = AccErrorX / 200;
AccErrorY = AccErrorY / 200;
c = 0;
// Read gyro values 200 times
while (c < 200) {
Wire.beginTransmission(MPU);
Wire.write(0x43);
Wire.endTransmission(false);
Wire.requestFrom(MPU, 6, true);
GyroX = Wire.read() << 8 | Wire.read();
GyroY = Wire.read() << 8 | Wire.read();
```

```
GyroZ = Wire.read() << 8 | Wire.read();
// Sum all readings
GyroErrorX = GyroErrorX + (GyroX / 131.0);
GyroErrorY = GyroErrorY + (GyroY / 131.0);
GyroErrorZ = GyroErrorZ + (GyroZ / 131.0);
c++;
}
//Divide the sum by 200 to get the error value
GyroErrorX = GyroErrorX / 200;
GyroErrorY = GyroErrorY / 200;
GyroErrorZ = GyroErrorZ / 200;
// Print the error values on the Serial Monitor
// Serial.print("AccErrorX: ");
// Serial.println(AccErrorX);
// Serial.print("AccErrorY: ");
// Serial.println(AccErrorY);
Serial.print("GyroErrorX: ");
Serial.print(GyroErrorX);
Serial.print("GyroErrorY: ");
Serial.print(GyroErrorY);
Serial.print("GyroErrorZ: ");
Serial.println(GyroErrorZ);
}
```

5.5 Game - Python Code

```
# ================================================================================================
# -- IMPORTS -----------------------------------------------------------------------------------
# ================================================================================================



from vpython import *

import serial
import matplotlib.pyplot as plt

import numpy as np
from   numpy import sin, cos

import pylab as py
import math
from   drawnow import *



# ================================================================================================
# -- ARDUINO PREP ------------------------------------------------------------------------------
# ================================================================================================



arduinodata = serial.Serial('COM8',9600)  #port name and baud rate

t = -1



# ================================================================================================
# -- FILTERS -----------------------------------------------------------------------------------
# ================================================================================================



def kalman(data):
"""
the data must be 1D and the dimension shoulh be equal to (2n+1)
"""
data = np.array(data)
return float(np.sum(data)/np.size(data))
```

```
# ================================================================================================
# -- CONSTANTS -----------------------------------------------------------------------------------
# ================================================================================================



# ==========================================
# define simulation constants
room_x = 12
room_y = 10
room_z  = 16
wall_thickness = 0.5
wall_color  = vector(1, 1, 1)
wall_opacity = 0.8
front_opacity  = 0.7
marble_radius  = 0.5
ball_color  = vector(0, 0, 1)
ball_color_attack = vector(1, 0, 0)
racket_color  = vector(0, 1, 0)
racket_speed  = 0.3

racket_size  = vector(room_x, room_y, wall_thickness)
# ==========================================

# ==========================================
# temp values initilization
temp_value  = 5
roll_kalman_temp  = np.zeros(temp_value)
pitch_kalman_temp  = np.zeros(temp_value)
yaw_kalman_temp  = np.zeros(temp_value)
# ==========================================



# ================================================================================================
# -- ROOM DEFINITION -----------------------------------------------------------------------------
# ================================================================================================



# making the box
my_floor  = box(size=vector(room_x, wall_thickness, room_z), pos=vector(0, -room_y/2, 0),
my_ceiling  = box(size=vector(room_x, wall_thickness, room_z), pos=vector(0, room_y/2, 0),
left_wall  = box(size=vector(wall_thickness, room_y, room_z), pos=vector(-room_x/2, 0, 0),
right_wall  = box(size=vector(wall_thickness, room_y, room_z), pos=vector(+room_x/2, 0, 0)
back_wall  = box(size=vector(room_x, room_y, wall_thickness), pos=vector(0, 0, -room_z/2),
racket   = box(size=racket_size, pos=vector(0, 0, +room_z/2), color=racket_color, opacity=

# making the ball
marble = sphere(color=ball_color, radius=marble_radius)
```

```
# assigning marble speed
marble_x  = 0
marble_y  = 0
marble_z  = 0

delta_x = 0.1
delta_y  = 0.1
delta_z  = -0.1


racket_speed_x = 0
racket_speed_y = 0



# ============================================================================================
# -- MAIN LOOP ---------------------------------------------------------------------------
# ============================================================================================



# start the loop
while True:

#reset the game
if marble_z > room_z/2:
break

t += 1

# reading the data from arduino
while(arduinodata.inWaiting()==0):
pass

arduinostring=arduinodata.readline()
arduinostring=str(arduinostring,encoding="utf-8")
dataArray=arduinostring.split(',')

# skip the first few lines
if t <= 20:
continue

# current roll, pitch and yaw
roll  = float(dataArray[0])
pitch = float(dataArray[1])
yaw   = float(dataArray[2])

# kalman implemented
roll_kalman_temp[-t%temp_value]  = roll
```

```
pitch_kalman_temp[-t%temp_value]  = pitch
yaw_kalman_temp[-t%temp_value]  = yaw

# movement from arduino kit to the racket
if roll_kalman_temp[int((temp_value-1)/2)] > 5:
racket_speed_x = -racket_speed
if roll_kalman_temp[int((temp_value-1)/2)] < -5:
racket_speed_x = +racket_speed
if  (roll_kalman_temp[int((temp_value-1)/2)] < 5) and (roll_kalman_temp[int((temp_value-1)
racket_speed_x = 0

if pitch_kalman_temp[int((temp_value-1)/2)] > 5:
racket_speed_y = +racket_speed
if pitch_kalman_temp[int((temp_value-1)/2)] < -5:
racket_speed_y = -racket_speed
if  (pitch_kalman_temp[int((temp_value-1)/2)] < 5) and (pitch_kalman_temp[int((temp_value-
racket_speed_y = 0

marble_x = marble_x + delta_x
marble_y = marble_y + delta_y
marble_z = marble_z + delta_z

racket.pos.x = racket.pos.x + racket_speed_x
racket.pos.y = racket.pos.y + racket_speed_y

# collision detection
if ((marble_x + marble_radius)>(room_x/2 - wall_thickness/2)) or ((marble_x - marble_radiu
delta_x = -delta_x

if ((marble_y + marble_radius)>(room_y/2 - wall_thickness/2)) or ((marble_y - marble_radiu
delta_y = -delta_y

if ((marble_z - marble_radius)<(-room_z/2 + wall_thickness/2)):
delta_z = -delta_z

# collision with racket
is_marbel_in_front_of_racket = (marble_x > racket.pos.x - racket_size.x/2) and (marble_x <

if  ((marble_z + marble_radius)>(room_z/2 - wall_thickness/2)) and is_marbel_in_front_of_r
delta_z = -delta_z

# change color when near on attack mode
if delta_z > 0:
marble.color = ball_color_attack
elif delta_z <= 0:
marble.color = ball_color

# update the marble position
marble.pos = vector(marble_x, marble_y, marble_z)
```

6 References

1. Youtube video: How to install Arduino Library from github