Politecnico di Torino

Video Game Control using an STM32 Microcontroller linked to a Python Code Commands Generator

Yithzak Alarcón

Sara Contreras

M.Sc Students

Prof. Marco Vacca

Electronic Systems for Sensor Acquisition

POLITECNICO DI TORINO

Department of Mechatronic Engineering

January 31, 2022

Contents

1 Introduction

A microcontroller is an integrated circuit device composed of a microprocessor along with memory and input/output peripherals on a single chip [9]. It can be used to control part or all the functions of an embedded system. For this project, we used an STM32F401RE microcontroller that is part of the 32-bit flash microcontrollers family based on ARM®Cortex®M processors. It has 84 MHz CPU/105 DMIPS and it is characterized to be the smallest, cost-effective solution with notable power efficiency. Another important feature of the STM32F4 series is that consists of 7 lines of digital signal controllers which allows to have better results with the real-time control capabilities of the microcontroller and the digital signal processor [6,7].

The goal of this project is to control a video game with the STM32. The video game selected for this case was Mario-Kart and the main task decided to achieved with the microcontroller is to press the keys of the keyboard to control the game in general, from the selection of characters to control the direction of the car. For this purpose was used the 3-axis accelerometer LIS2DW12 and the 3-axis gyroscope LSM6DSO integrated in the X-NUCLEO-IKS01A3 sensor expansion board. The filters selected to process the signals were the moving average and the complementary filter which will be explain in the Design Methodology section.

This report present the development of this project which is composed by four main stages: The first one is the theoretical approach, the second is Design Methodology, the third is the development of the Keyboard interface and finally the code implementation.

2 Theoretical approach

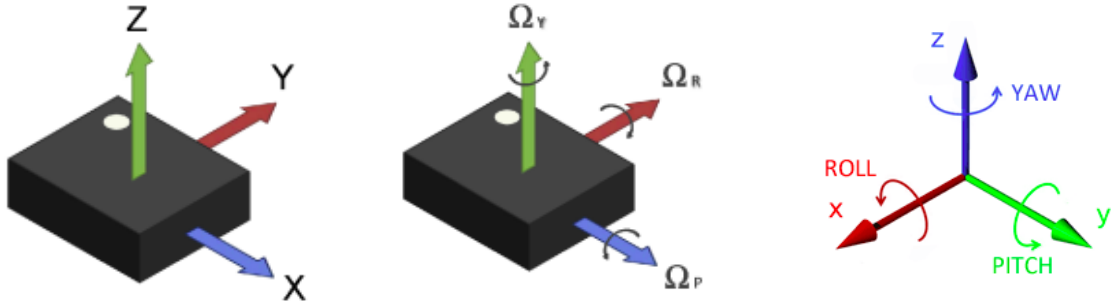
2.1 Deriving angles from angular acceleration (Accelerometer)

The inertial expansion board let us to obtain the acceleration in the three axis of rotation, in general, we can represent these rotations with RPY Angles coordinates, that are a derivation of Euler angles for independent axis rotation, we can observed the axis direction in RPY angles representation in figure (1b) or we can took the Euler angle representation in figure (2) and derive the following equations for each axis angle.

$$\sqrt{\alpha_x^2 + \alpha_y^2 + \alpha_z^2} = g \tag{1}$$

The above equation (1) is important to derive using basic trigonometry the equations for each axis angle, we can take advantages that the magnitude of the three vectors always equals the magnitude of the gravity vector $\vec{g}$.

We take each axis angle rotation independently and not considering as a series of rotation as in the RPY representation, because we can consider each axis rotation individually to meet an specific condition, therefore, by applying basic trigonometry and taking equation (1), we can obtain the equations (2), (3) and (4) below.



(a) XYZ Cartesian coordinates representation on inertial module extension

(b) RPY Rotation representation on inertial module extension for STM32

(c) General RPY Angles rotation representation

Figure 1: Coordinate representation

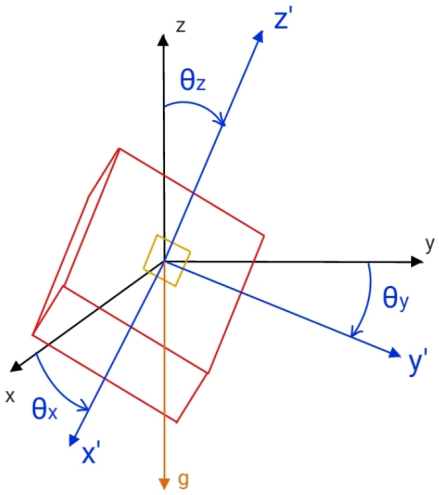


Figure 2: Axis rotation representation with inertial module sketch

$$\theta_x = \tan^{-1}\left(\frac{\alpha_x}{\sqrt{\alpha_y^2 + \alpha_z^2}}\right) \tag{2}$$

$$\theta_y = \tan^{-1}\left(\frac{\alpha_y}{\sqrt{\alpha_x^2 + \alpha_z^2}}\right) \tag{3}$$

$$\theta_z = \tan^{-1}\left(\frac{\sqrt{\alpha_x^2 + \alpha_y^2}}{\alpha_z}\right) \tag{4}$$

By calculating the axis angles with the above equations, we take advantage of not having drift in the medium-long term, that in comparison with a double-integrating process would carry a significant amount of accumulative error, and also adding the noise presented in the signal.

2.2 Deriving angles from angular velocities (Gyroscope)

To obtain the angles from the angular axis velocities, we have to use a different approach than with the accelerometer, in this case, we have to do a integration process to obtain the corresponding angles for each axis. Theoretically, angular displacement (similar as obtaining linear displacement) is obtained trough equation (5):

$$\theta = \theta_0 + \int_{t_i}^{t} \omega \, dt \tag{5}$$

In practice, to be able to compute the integral process, we have to execute an approximation process, knowing that integrate a given curve in time is obtaining the area under this one. In our case, we use the trapezoidal area (Piecewise linear approximation) approach as we can observe in figure (3):

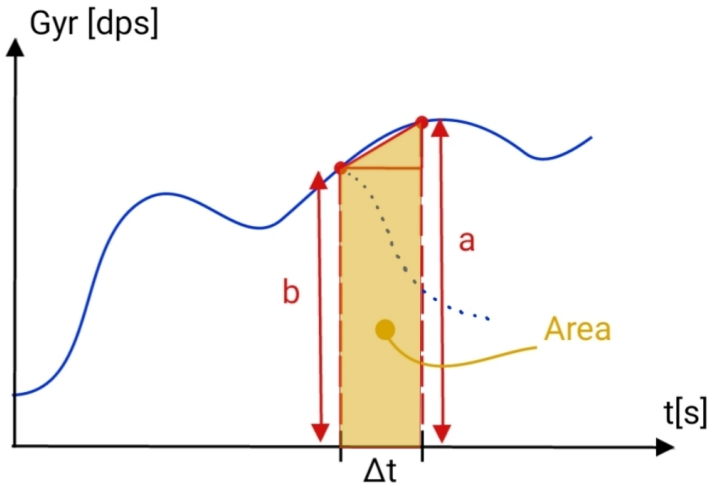


Figure 3: Gyroscope signal sketch over time

Recalling the equation for an area of a trapezoid:

$$A = \frac{(a+b)}{2} \cdot \Delta t \tag{6}$$

where,

- a and b: Corresponds to the angular velocities (ω_t, ω_{t-1}) obtained from the gyroscope.

- Δt: Corresponds to the time interval between one angular velocity and the other.

It is important to highlight that the accuracy of the equation (6), also depends on the fact that $\Delta t \to 0$, at least in theory, but experimentally we can get a good approximation with the timers of the STM32 due to the working speed around 84MHz, fast enough and realiable. We will also face a drift problem due to the accumulative error in the integration process, for this issue, we use the angles obtained from the accelerometer and, a soft filtering and calibration process is needed.

3 Design Methodology

As it was mentioned before, we worked with the output signals obtained from the 3-axis accelerometer and 3-axis gyroscope with the purpose to obtain the angular position in the x, y and z axes. In order to extract the information from the sensors and transmit the instruction properly to the computer keyboard, we follow the approach shown in the figure 4:

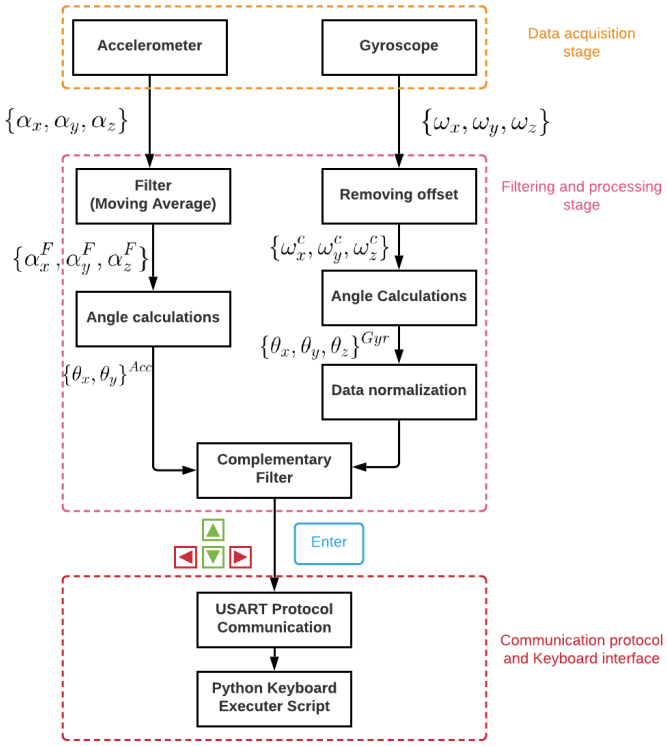


Figure 4: Block diagram of the general process

3.1 Signal Acquisition

The first step was placed the microcontroller in a core that emulates the steering wheel, as is shown in figure (5):

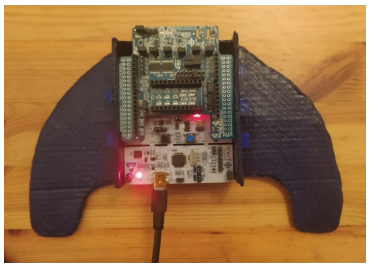


Figure 5: Microcontroller core

To be able to detect if the user wants to move forward or to move the car right or left, we needed to decided the movements for the steering wheel that were going to be available for the

user. The selection is shown in figure (6). Once that these information was set, the approach taken was to work with the angular position in the three axes.

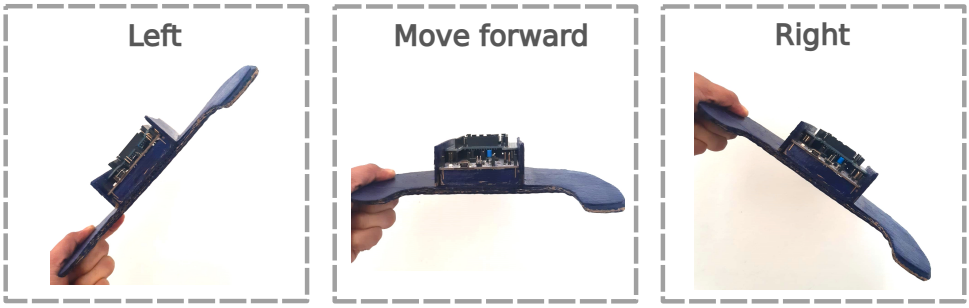


Figure 6: Movements allowed for the user.

For this purpose we used the sensors LIS2DW12 and LSM6DSO integrated in the X-NUCLEO-IKS01A3 sensor expansion board. For capturing the data we needed to use and configure the I2C protocol to allow the connection between the microcontroller and the expansion board, as well as load the IKS01A3 drivers to the Nucleo board [8].

3.2 Signal Processing

Once we acquired the signal the next step was transform those signals in angular position. To do that we took the two sensors separately and according to their behaviour we selected different signal processing strategies. The figure 8 shows the procedure follow for each sensor. For the case of the accelerometer, first we applied a moving average filter that is Low-pass Finite Impulse response filter (FIR) which allows to eliminate the white noise and obtain a smoother output signal. In this case it takes a sample of five data from the acceleration of each axis and average those samples to produce a single output point. The comparison of the acceleration signals for each axis before and after the filter can be observed in figure (9). As is possible to notice, the quality of the signal improve after the filter was applied which allows to manipulate more accurate signals. After the filtering we compute the angles with the procedure explain in the section 2.1.

For the gyroscope, as a first step we put the core with the microcontroller in a rigid table to measure the offset that each axes presents. The results were:

1. x-axis: 315
2. y-axis: 70
3. z-axis: -350

These values were subtracted from the output signals of the gyroscope, obtaining the results shown in figure (10). After that we computed the angles with the procedure explain in the section 2.2. To implement the integration in the STM32 we configured the TIM3 in interruption mode and set the values of the prescaler and the counter period to have a delta time of 10 ms. Which means that each 10 ms the microcontroller is capturing data and performing the integration. To find the values of these parameters we used the next equation:

$$\Delta t \cdot \left(\frac{84MHz}{Prescaler} \right) = CounterPeriod \tag{7}$$

Where 84 MHz is the clock frequency. In figure (7), is shown the configuration for the timer

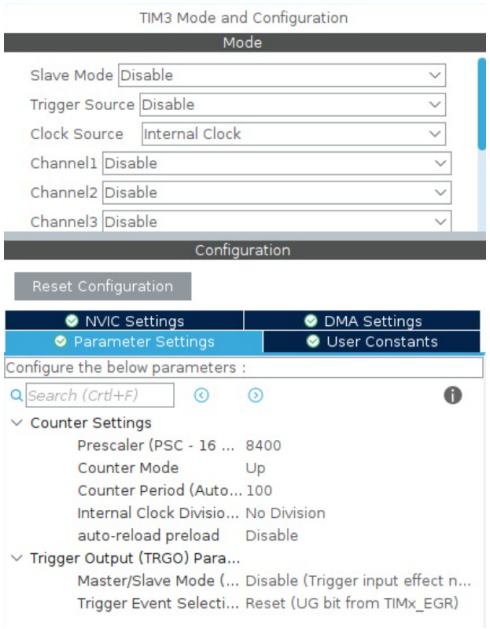


Figure 7: Time 3 configuration

Once we obtain the angles around the axes from the accelerometer and gyroscope, we applied a complementary filter. The complementary filter takes advantage of the features of each sensor considering for instance, that in the case of the accelerometer it does not present drift in medium or long term because it takes the absolute measure of the angle that forms the sensor with the vertical direction label by the gravity. However, it can be affected by the movement of the sensor and the noise in short term. In the case of the gyroscope it considers that it works good for short or exaggerated movements, but when we extract the angle by the integration with respect to time it can accumulate noise in short term. For these reasons the complementary filter behaves as a High-pass filter for the gyroscope measurements and Low-pass filter for the accelerometer signal, which means that in short term the dominant signal is from the gyroscope and in long term is from the accelerometer. The equation used to implement the complementary filter is:

$$\theta = A \cdot (\theta_{gyro}) + B \cdot (\theta_{accel}) \tag{8}$$

As is indicated in figure (8) from this filter we extracted the angular position in x and y axes, and the z-axis angular position obtained from only the gyroscope.

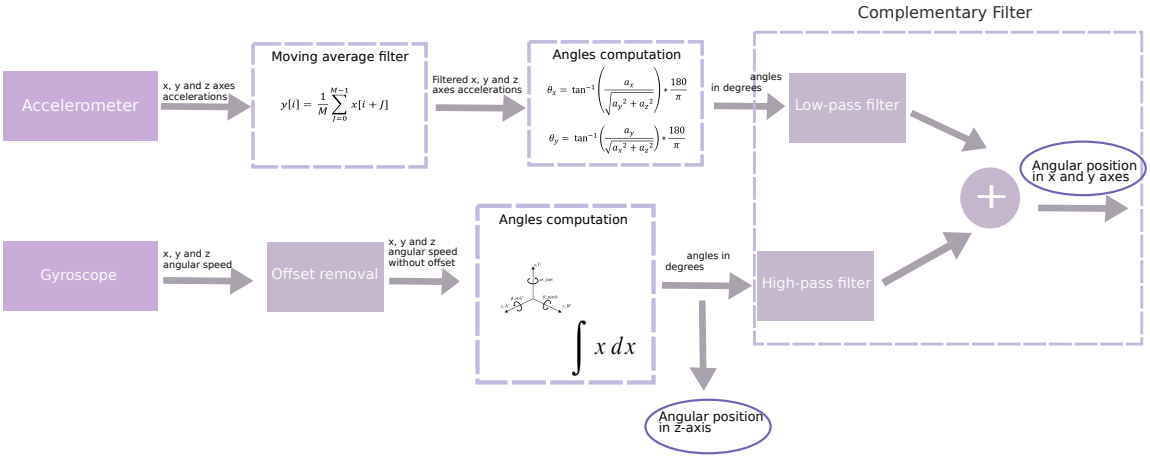


Figure 8: Methodology for signal processing

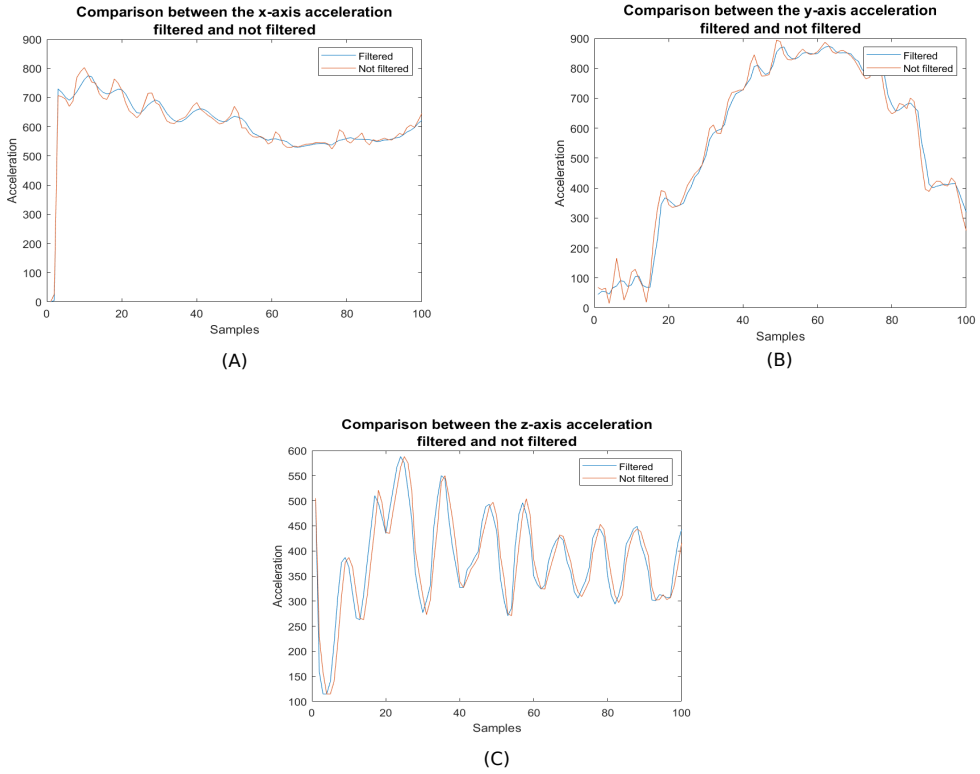


Figure 9: Comparison between the 3-axes acceleration filtered and not filtered

3.3 Sensor Calibration

After the angular position was obtained for the three-axis, we proceed to establish the range for each angle to send the instruction to the keyboard. To do this we did some trials according to the movement previously selected, shown in figure 6. Since the microcontroller together with the core was going to act as the steering wheel for the user and considering the instructions that allow the game, there were three important movements: forward, turn left and turn right.

Once the firsts trial were done, we find out that for the situations when the user wants to turn left or right and the car was moving, was better to create only one condition that evaluates and sent the instruction to, for instance, move forward and turn left or right. This was because

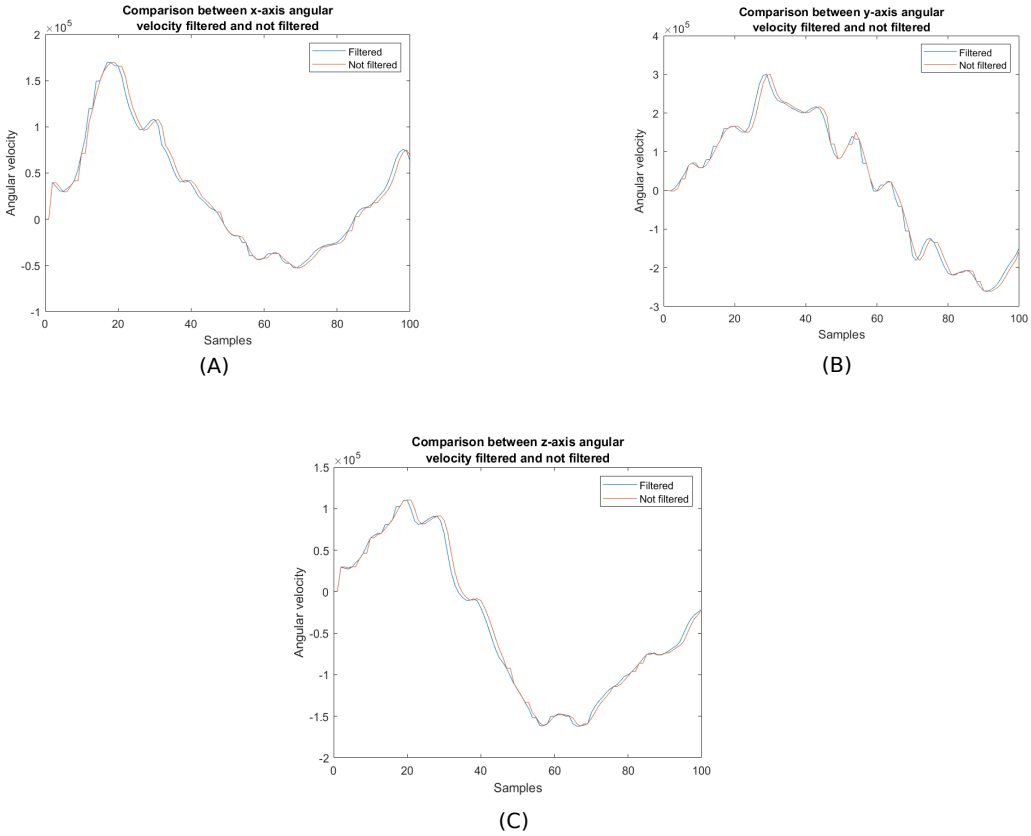


Figure 10: Comparison between the 3-axes angular velocity filtered and not filtered

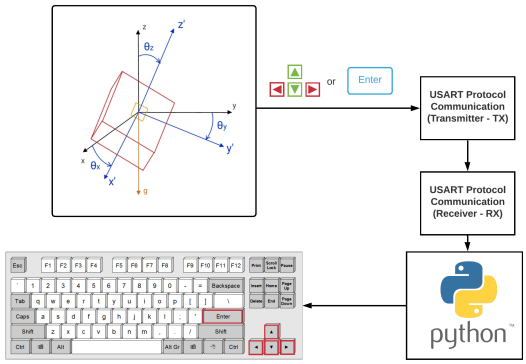
the time of reaction for the user's movements of the microcontroller was faster than with the evaluation of two conditions (one for moving forward and another for turning left or right) which allows having a more accurate implementation. We added also the feature to be able to select the character by moving the "steering wheel" and we activated an interruption to the button of the STM32 that press the enter in the keyboard. The results of the calibration are shown in table 1:

Angle x	**Angle y**	**Angle z**	**Instruction**
[-10,10]	[-20,30]	-	Move forward
[10,30]	[-20,30]	-	Move Forward and turn left
<-15	[-20,30]	-	Move Forward and turn right
-	>30	>20	Turn left
-	>30	<-20	Turn right

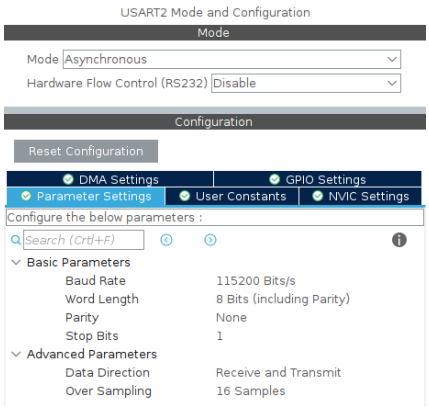
Table 1: Sensor Calibration

Another calibration process that was done is the calibration of the complementary filter. What is recommended in the literature [4] is to start with A and B as 0.98 and 0.02 respectively. Considering that the sum of these coefficients must be 1 always, we obtained better results decreasing a little B and consequentially increasing A. The values selected for this case were 0.08 for A and 0.92 for B.

3.4 Keyboard Interface



(a) Keyboard interface block diagram



(b) USART Protocol Configuration in STM32CubeIDE

Figure 11: Keyboard interface and USART configuration

As we can observe in figure (11a), the first part is already discussed, once we obtain the commands generated by the STM32, we send it through USART protocol @115200 bits/s, to be received and read by the Python script. We have to take into account the speed we're employing with the USART, because in the Python script the reading must be in the same speed in order to receive reliable information. In figure (11b), we can see the configuration to be used with STM32, and we can observed the following implementation in the Python script:

```
while 1:
    #Decoding information from the serial port
    if serial.in_waiting:
        key = serial.read().decode('ascii')
        #Conditions to press and release Keyboard keys
        if key == 'e':
            #Special case for 'enter'
            keyboard.press(Key.enter);
            time.sleep(0.005);
            keyboard.release(Key.enter);
            key = '\a';
        #Press-Release condition for Left and Right
        if key == 'a':
            keyboard.release('d');
        elif key == 'd':
            keyboard.release('a');
        #Release condition in case it's sending nothing
        if key == "\n":
            keyboard.release('w');
            keyboard.release('a');
            keyboard.release('d');
        #Press command activation
        keyboard.press(key)
```

4 Code Implementation

Data acquisition

```
//Obtaining data from axis acceleration
xAxisReading_A = axes_A.x;
yAxisReading_A = axes_A.y;
zAxisReading_A = axes_A.z;
//Obtaining data from axis velocities
xAxisReading_G = axes_G.x;
yAxisReading_G = axes_G.y;
zAxisReading_G = axes_G.z;
//Return status for each module
returnstatus_A = IKS01A3_MOTION_SENSOR_GetAxes(1,MOTION_ACCELERO,&axes_A);
returnstatus_G = IKS01A3_MOTION_SENSOR_GetAxes(0,MOTION_GYRO,&axes_G);
```

Signal processing

Accelerometer

```
//Conditional For-Loop to add new data to the accumulative vector of acceleration
for(int i=0; i<5; i++){
    if(i==4){
        accx_f[i] = axes_A.x;
        accy_f[i] = axes_A.y;
        accz_f[i] = axes_A.z;
    }else{
        accx_f[i] = accx_f[i+1];
        accy_f[i] = accy_f[i+1];
        accz_f[i] = accz_f[i+1];
    }
}
//Applying Moving Average Filter function
xAxisReadingA_Fil = filter(accx_f);
yAxisReadingA_Fil = filter(accy_f);
zAxisReadingA_Fil = filter(accz_f);
//Moving average filter with 5 samples
int filter(int acc[5]){
    //Accumulative filter variable initially set to 0
    int accf = 0;
    //Summation and division process to obtain the average
    for(int i = 0; i < 5; i++)
        accf += acc[i]/5;
    return accf;
}

  int arctangent(int ax, int ay, int az, int y){
      int angle;
      //Trigonometric equations to obtain the angles given axis components
      //And conversion from radians to degree
      thetax = atan(ax/sqrt((ay*ay)+(az*az)))*(180/pi);
      thetay = atan(ay/sqrt((ax*ax)+(az*az)))*(180/pi);
      //Recursive condition to return different angles with same function
      if(y==1){
          angle = thetax;
      }else if(y==2){
          angle = thetay;
      }
      return angle;
  }
```

Gyroscope

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim3){
    //Operations to be performed each time the Timer counter reaches the value of the ARR register
    //Removing offset from raw data
    //Assign to actual variable
    zVector2 = (zAxisReading_G+350);
    xVector2 = (xAxisReading_G-315);
    yVector2 = (yAxisReading_G-70);
    //Conditional to assign previous and actual Gyroscope velocity values for integration process
    i2++;
    if(i2 > 0){
        Valueprez = zAxisReading_G;
        Valueprey = yAxisReading_G;
        Valueprex = xAxisReading_G;
    }else if(i2 > 1){
        zVector1 = (Valueprez+350);
        xVector1 = (Valueprex-315);
        yVector1 = (Valueprey-70);
        i2 = 0;
    }
    //Accumulative and integration process to obtain angle x,y and z from the gyroscope data
    positionZ += position_int(zVector1, zVector2);
    positionX += position_int(xVector1, xVector2);
    positionY += position_int(yVector1, yVector2);
    //Filter processing to normalize data into readable values
    anglez = positionZ/500;
    anglex_g = positionX/500;
    angley_g = positionY/500;
    anglez_g = positionZ/500;
}



  int position_int(int zVector1_1, int zVector2_1){
      //The equation for the integration is: position = (zVector2+zVector1)*deltaTime/2
      int32_t sum = zVector2_1+zVector1_1;
      //In this case the time interval is 10 ms
      //Therefore, we have to divide by 200 including the division by two of the formula
      int32_t resultPos = sum/200;
      return resultPos;
  }
```

```
//Complementary filter
anglex = arctangent(xAxisReadingA_Fil, yAxisReadingA_Fil, zAxisReadingA_Fil, 1)*92/100+anglex_g*8/100;
angley = arctangent(xAxisReadingA_Fil, yAxisReadingA_Fil, zAxisReadingA_Fil, 2)*92/100+angley_g*8/100;
```

Keyboard instructions

```
//Enter:
if(exticounter > 0){
      HAL_UART_Transmit_IT(&huart2,(uint8_t *)enter,strlen(enter));
      HAL_Delay(2);
      exticounter = 0;
}
//Drive forward:
if(anglex > -10 && anglex < 10 && angley < 30 && angley > -20){
      HAL_UART_Transmit_IT(&huart2,(uint8_t *)forward,strlen(forward));
      HAL_Delay(2);
//Move forward and turn left:
}else if(anglex > 10 && angley < 30 && angley > -20){
      HAL_UART_Transmit_IT(&huart2,(uint8_t *)forward,strlen(forward));
      HAL_Delay(2);
      HAL_UART_Transmit_IT(&huart2,(uint8_t *)left,strlen(left));
      HAL_Delay(2);
//Move forward and turn right:
}else if(anglex < -15 && angley < 30 && angley > -20){
      HAL_UART_Transmit_IT(&huart2,(uint8_t *)forward,strlen(forward));
      HAL_Delay(2);
      HAL_UART_Transmit_IT(&huart2,(uint8_t *)right,strlen(right));
      HAL_Delay(2);
//Turn left
}else if(anglez > 20 && angley > 30){
      HAL_UART_Transmit_IT(&huart2,(uint8_t *)left,strlen(left));
      HAL_Delay(2);
//Turn right
}else if(anglez < -20 && angley > 30){
      HAL_UART_Transmit_IT(&huart2,(uint8_t *)right,strlen(right));
      HAL_Delay(2);
//Stop
}else{
      HAL_UART_Transmit_IT(&huart2,(uint8_t *)stop,strlen(stop));
      HAL_Delay(2);
}
```

Figure 12: Code implementation in STM32CubeIDE

5 Conclusions

- Implement a filtering stage in the manipulation of signals greatly improves the performance and allows to obtain more accurate information from it. These effect can be observed in the smooth process of the signal, that allows the system to not have any abnormal fluctuation and become reliable and repeatable.
- By the using of the external interruption in polling mode we get a not relevant delay in the command 'enter' sending, and also we have to consider the noise contained in the hardware of the button, that cause a not always reliable 'pressing', but in general, this work fine along all the interruption and conditional requirement in the while loop.
- Finally, we observe that the interface between the Transmitter-Receiver of the USART communicating the CubeIDE project and the Python script do not present any difficulty or relevant delay when sending the information and allows the user to play the game smoothly.

References

[1] B. Bona, Dynamic modelling of mechatronic systems. Torino: Celid, 2013, pp. 39-41.

[2] B. Siciliano, Robotics - Modelling, Planning and Control. 2010, pp. 48-54.

[3] "LSM6DS0 - STMicroelectronics", STMicroelectronics, 2022. [Online]. Available: www.st.com/en/mems-and-sensors/lsm6ds0.html#documentation

[4] "Medir la inclinación con IMU, Arduino y filtro complementario", Luis Llamas, 2022. [Online]. Available: www.luisllamas.es/medir-la-inclinacion-imu-arduino-filtro-complementario

[5] "Complementary filter design", Gait analysis made simple, 2022. [Online]. Available: https://gunjanpatel.wordpress.com/2016/07/07/complementary-filter-design

[6] "STM32 32-bit Arm Cortex MCUs - STMicroelectronics", STMicroelectronics, 2022. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html

[7] "STM32F4 Series", 2022. [Online]. Available: www.digikey.com/es/product-highlight/s/stmicroelectronics/stm32-f4

[8] "X-NUCLEO-IKS01A3 - STMicroelectronics", STMicroelectronics, 2022. [Online]. Available: www.st.com/en/ecosystems/x-nucleo-iks01a3.html

[9] "What is a Microcontroller and How Does it Work?", IoT Agenda, 2022. [Online]. Available: https://internetofthingsagenda.techtarget.com/definition/microcontroller