POLITECNICO DI TORINO

Department of
CONTROL AND COMPUTER ENGINEERING
(DAUIN)

Master's degree in Mechatronic Engineering
2021/2022

Electronic systems for sensor acquisition
Prof. Vacca

Programming a controller for Mario Kart 64 using STM32

Group 5
Marco Barbon – 287462
Claudio Briganti – 287399
Riccardo Tassi – 286245

1 Objectives

Design and develop a controller to be used as a steering wheel in "Mario Kart 64" game, by means of a proper C code implemented inside a Nucleo board shield able to acquire, elaborate and transmit data to a PC where the game is running.

2 Components required

2.1 Hardware required

- ST-microelectronics Nucleo-F401RE (board)
- X-Nucleo-IKS01A3(sensor shield)
- Mini USB cable

2.2 Software required

C code:

- STM32CubeIDE
- STM32CubeMX

Python code:

- Visual Studio Code(IDE)

3 Structure of the project

The board is intended to be used as a steering wheel, meaning it must be held as in Figure 1. Actions are performed rotating the board left, right, towards player's chest and pressing the blue button. To obtain such results we started acquiring angular velocity and linear acceleration measurements. Such data was then averaged to remove part of the noise it is subject to. Clean measurements were successively processed exploiting trigonometric relations to retrieve the orientation of the board: depending on it commands were generated and sent to the PC to be elaborated by a Python script and converted into game actions. Next subsections thoroughly explain all the steps.

3.1 Data acquisition from sensor

The LSM6DSO accelerometer present in the sensor shield was used to acquire linear acceleration measurements along X,Y,Z axes of the reference frame associated to the board as shown in Figure 1 . This raw data was successively employed to determine the steering angle of our controller, that is the angular position of the board with respect to the horizontal line.
The LSM6DSO gyroscope was used instead to acquire angular velocities measurements along X axis of the reference frame associated to the board. This raw data

was successively used to determine the angular movement of the board toward the player's chest.

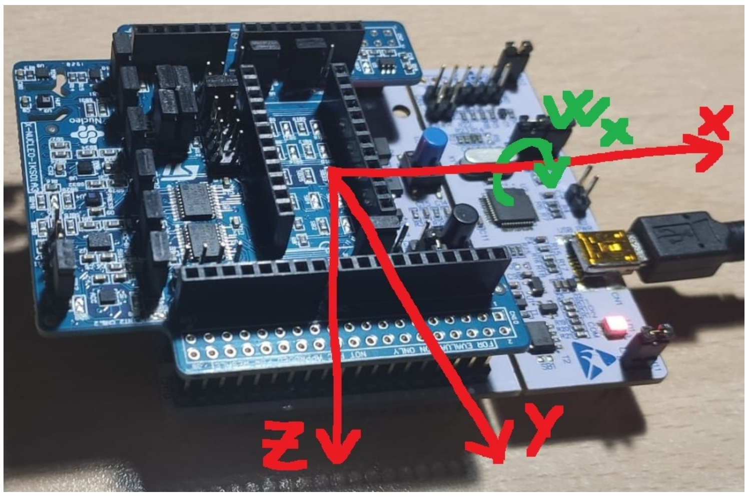


Figure 1: Direction of detectable acceleration and angular velocity (top view)

The information acquired in these way have the features (from the sensor datasheet) shown in Tab. 1:

Sensor characteristic	**Accelerometer**	**Gyroscope**
Power consumption (combo high-performance mode)	0.55 mA (both)	
Measurement input range	±2/±4/±8/±16 g	±125/±250/±500/±1000/±2000 dps
Sensitivity	0.061/0.122/0.244/0.488 mg/LSB	4.375/8.75/17.50/35/70 mdps/LSB
Sensitivity tolerance	±1% (both)	
Sensitivity change vs. temperature (from -40° to +85°)	±0.01%/°C	±0.007%/°C
Zero-g level offset accuracy	±20 mg	±1 dps
Zero-g level change vs. temperature	±0.1 mg/°C	±0.01 dps/°C
Noise in normal/ low-power mode (RMS)	1.8/2.0/2.4/3.0 mg	75 mdps
Operating temperature range	from -40 to +85 °C (both)	
Warm up time	35 ms (both)	
Other	Embedded temperature sensor present (both)	

Table 1: Sensor characteristics of LSM6DSO linear accelerometer and gyroscope

Data acquisition was performed using IKS01A3 motion sensor library, data from sensor is provided to the board through I^2C protocol.

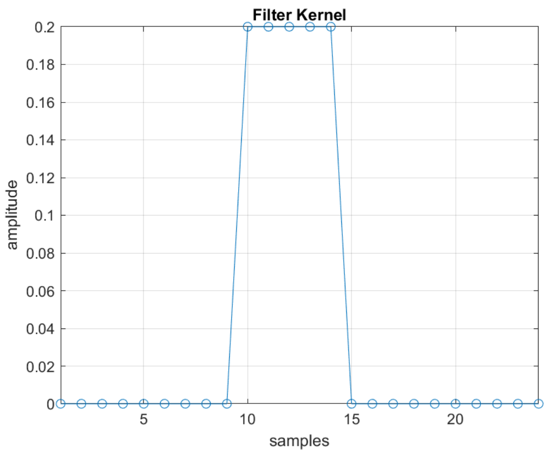
```
#include "iks01a3_motion_sensors.h"

// Initialize and enable the accelerometer in X-MEMS
IKS01A3_MOTION_SENSOR_Init(1,MOTION_ACCELERO);
IKS01A3_MOTION_SENSOR_Enable(1,MOTION_ACCELERO);
// Initialize and enable the gyroscope in X-MEMS
```

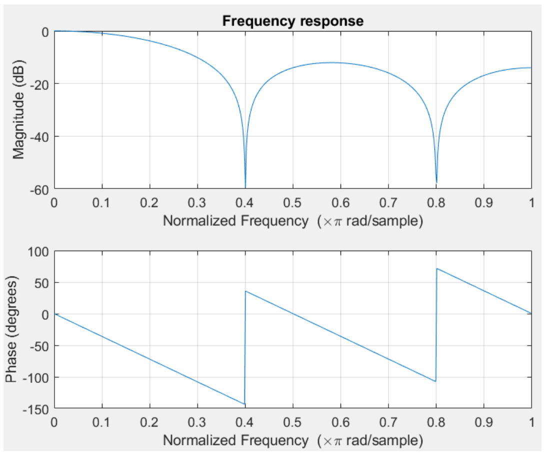
```
    IKS01A3_MOTION_SENSOR_Init(0,MOTION_GYRO);
    IKS01A3_MOTION_SENSOR_Enable(0,MOTION_GYRO);
```

3.2 Digital filter

The sensor data measured in this way are obviously corrupted by a high frequency noise, then in order to attenuate it we decided to apply a digital filter. We chose a Finite Impulse Response (FIR) Moving Average Filter that is the lighter filter able to accomplish our purposes. We implemented it using the recursive approach since it has a fast algorithm and since it avoids the signal's phase shift. We chose a window of 5 points allowing already to obtain a good attenuation, in fact the noise is reduced of $\sqrt{5} = 2.24$ times (attenuation of about -7 dB). Figure 2 shows the kernel and the frequency response of the filter



(a) *Filter kernel*



(b) *Filter frequency response*

Figure 2: Filter characteristics

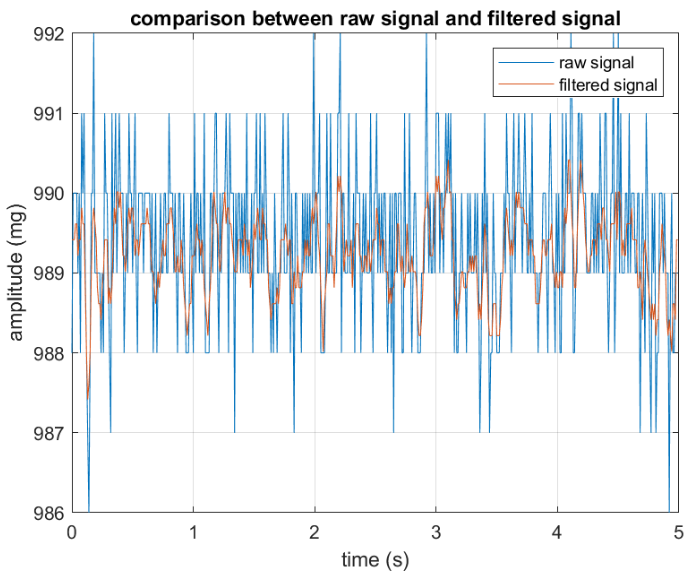


Figure 3: Raw and filtered signal

In order to test the filter, we recorded through MATLAB (the file is called 'filter_design.m') a signal of 5 seconds from the serial port (acceleration on the Z axis when the board is still on the table) and we filtered it. Figure 3 shows the filtering result, as can be seen a 5 point moving average gives already an acceptable result without burdening too much the processor. The filter was then implemented in C as follows:

```c
// Three vectors to store data from sensor and filter them
int32_t xAxisReading[6] = {0};
int32_t yAxisReading[6] = {0};
int32_t zAxisReading[6] = {0};
int32_t xAxisReadingGyro[6] = {0};
// Vector storing X,Y,Z acceleration and X angular rate filtered data
int cleanData[4] = {0};

void getCleanData(int cleanData[], int32_t xAxisReading[], int32_t yAxisReading[],
    int32_t zAxisReading[], int32_t xAxisReadingGyro[]){
/*Implementation of Moving average filter */

  // Initialization of the struct to store sensor measurements
  IKS01A3_MOTION_SENSOR_Axes_t axes;
  IKS01A3_MOTION_SENSOR_Axes_t axesGyro;

  // Initialization of check of good acquisition
  int8_t returnStatus = 0;
  int8_t returnStatusGyro = 0;

   // Start of data acquisition and filtering
   // Accelerometer measures in mg (1000 mg = 9.81 m/s^2), Gyroscope measures in
       millidegree/s


  // Reading from sensors using IKS01A3 library
   returnStatus = IKS01A3_MOTION_SENSOR_GetAxes(1,MOTION_ACCELERO,&axes); // I am
       acquiring data from the accelerometer in 3 directions
   returnStatusGyro = IKS01A3_MOTION_SENSOR_GetAxes(0,MOTION_GYRO,&axesGyro);// I
       am acquiring data from the gyroscope in 3 directions

   // An update of previous measures is performed only if both sensor readings were
        successful
   if(returnStatus == BSP_ERROR_NONE && returnStatusGyro == BSP_ERROR_NONE){

      // Acquisition of new measurement
      xAxisReading[0] = axes.x;
      yAxisReading[0] = axes.y;
      zAxisReading[0] = axes.z;
      xAxisReadingGyro[0] = axesGyro.x;

     // Implementation of the recursive moving average filter (window of 5 points)
      cleanData[0] = cleanData[0] - xAxisReading[5]/5 + xAxisReading[0]/5;
      cleanData[1] = cleanData[1] - yAxisReading[5]/5 + yAxisReading[0]/5;
      cleanData[2] = cleanData[2] - zAxisReading[5]/5 + zAxisReading[0]/5;
      cleanData[3] = cleanData[3] - xAxisReadingGyro[5]/5 + xAxisReadingGyro[0]/5;

      // The old value is cancelled and the new value is now ready to be acquired
      dataShiftRight(xAxisReading, yAxisReading, zAxisReading, xAxisReadingGyro);
      }
}


void dataShiftRight(int32_t xAxisReading[],int32_t yAxisReading[],int32_t
    zAxisReading[], int32_t xAxisReadingGyro[]){
```

```
  /* This function will perform the right shifting of data contained inside four
      unidimensional arrays
   * of dimension 6. The shifting will not involve the first element which is left
       unmodified  */

  // Counter initialization
  int8_t i;

  // Actual data shifting
  for(i=5; i>0; i--){
    xAxisReading[i] = xAxisReading[i-1];
    yAxisReading[i] = yAxisReading[i-1];
    zAxisReading[i] = zAxisReading[i-1];
    xAxisReadingGyro[i] = xAxisReadingGyro[i-1];
  }
}
```

3.3 Computation of board orientation

The goal of our measurements is, as declared in subsection 3.1, to find the value of the angle between X axis of the board reference frame and the horizontal line while playing. To do this we assumed that the acceleration caused by the user was relatively small with respect to the gravity, thus the linear acceleration measured by the accelerometer on the three main directions of the board can be assumed to be the gravity itself. Gravity vector is constant and directed accordingly to Z* axis of a fixed reference frame. Such vector can be decomposed into two components, one directed along X axis and one along Y axis of the board reference frame. The angle between Z* (fixed reference frame) and Y (board reference frame) is exactly the orientation of the board with respect to the fixed reference frame: it can trivially be retrieved according to Eq. 1.

$$\alpha = \arctan \frac{Y_g}{X_g} \tag{1}$$

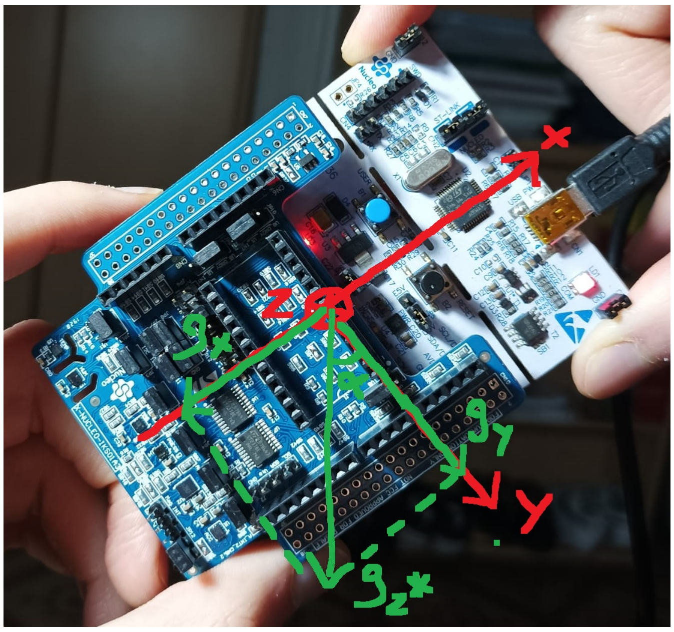


Figure 4: Representation of gravity vector decomposition in board's reference frame

Such operation is implemented with the following C function:

```c
#include "math.h"
float getXYorientation(int32_t cleanData[]){
  /* This function computes the XY orientation of the board starting from
   * Acceleration filtered data of X axis and Y axis: 0 degrees corresponds to
       board positioned
   * horizontally with MEMS board on the left side when facing someone's chest   */

  // Output variable initialization
   float angle;

   // Orientation computaiton in radiants
   angle=atan2(cleanData[1],cleanData[0]);

   // Orientation conversion in degrees
   angle=angle*180/3.1415;

   return angle;
}
```

This computation is meaningful only when the board has its Z-axis pointing towards player's chest: in case it is lying on the table, Z-axis of the board and of the fixed reference frame almost coincide, hence gravity vector has almost zero components on the board's X and Y axes and then their measurements are highly corrupted by the offset noise, this leads to completely wrong angle orientation.

Remarks: We chose this method to compute the angle, among all the others, because it required only few computational resources (in contrast to Sensor fusion techniques or Neural networks techniques) and it allows us to perform static computation of the desired value (in contrast to integration techniques of the angular acceleration that are dynamic computations and thus are affected by data drift problems).

3.4 Data transmission to PC

Later we needed to translate the information acquired and computed up to now, into the desired commands to be sent to the pc game.

We decided to follow a variable-threshold approach: to perform a gentle steering action, the value of the angle has to be in the range $\pm 10°$and $\pm 40°$; to perform a more aggressive steering action, the angle has to overstep $\pm 40°$.

We decided that leaving the board on the table corresponds to put the game in pause: this means that the acceleration on Z axis of the board will be approximately equal to the gravity acceleration; in this case a pause flag is set to 1.

In order to exit to the pause mode it is sufficient to rise the board and hold it again in playing position: doing so, the acceleration on Z axis of the board becomes significantly less than gravity acceleration; in this case we reset pause flag.

Jumping action can be performed rotating quickly the board towards player's chest (the threshold is set at 100000 mdps by trial and error) : this corresponds to a positive angular velocity along X axis.

Moreover, we decided that the blue button on the board could be exploited to use objects. Blue button pressing detection was performed using a EXTI interrupt, that would subsequently set button flag to 1.

Lastly, we set a neutral command: in case some error in acquisition stage occurred or the game was set to pause and we wanted to avoid to send any meaningful command to the PC, a "0" would be sent.

The corresponding C code is shown below:

```c
static void MX_GPIO_Init(void);

// Initialization of global flag to handle blue push-button through interrupt
volatile uint8_t button_flag = 0;

int main(void){

  // Flag activated when game is in pause
  uint8_t pause_flag = 0;

  // Flag activated to avoid conflicting commands
  uint8_t jump_flag = 0;

  // Series of variables to send commands to python script
  char command[5] = {0};
  char special[5] = {0};
  char jump[5] = {0};
  /* Initialize all configured peripherals */
  MX_GPIO_Init();

  // Initialize these two strings as they will not be modified inside the main
  strcpy(jump, "e");
  strcpy(special,"r");

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
// Acquisition and filtering of sensor data stored in cleanData
   getCleanData(cleanData, xAxisReading, yAxisReading, zAxisReading,
       xAxisReadingGyro);

   // Computation of orientation on XY plane of out board
     orientation = getXYorientation(cleanData);

 // Neutral value, this will not lead to any action in the game (unmodified only in
     case of malfunctioning)
   strcpy(command,"0");

   /* COMMANDS COMPUTATION */

 // In case the board is laying on the table and program is not yet in pause, pause
     command is sent
   if (cleanData[2] > 970 && pause_flag == 0){
     strcpy(command,"p");
     pause_flag = 1;
    }

   // In case game is in pause and board is no longer on the table, game is
      restarted
    if (cleanData[2] < 900  && pause_flag == 1){
      strcpy(command,"p");
      pause_flag = 0;
    }

    // If while playing board is accelerated towars player's chest, jump command
        is instantly sent
    if (cleanData[3] > 100000 && pause_flag == 0){
      jump_flag = 1;
      HAL_UART_Transmit_IT(&huart2,jump,strlen(jump));
      HAL_Delay(50);
      }

    // If while playing blue button is pressed, command to use objects is
        immediately sent
    if (button_flag == 1 && pause_flag == 0){
      button_flag = 0;
      HAL_UART_Transmit_IT(&huart2,special,strlen(special));
      HAL_Delay(50);
    }
    /* If while playing board is turned left, command to turn left is sent. If the
         orientation is between [-10,-40] degrees will be send in python the
        command to steer softly, instead if it is grater then -40 degrees it will
        be send the command to steer hardly (If jump command is sent, the
        orientation is such that also an unwanted turn left command is sent,
        jump_flag avoids this)*/
    if (orientation<=-10 && pause_flag == 0 && jump_flag == 0){
      if(orientation>=-40 && orientation<-10){
       strcpy(command,"l");
      }
      else if (orientation<-40) {
        strcpy(command,"f");
      }
    }

    /* If while playing board is turned right, command to turn right is sent. If
        the orientation is between [10,40] degrees will be send in python the
        command to steer softly, instead if it is grater then 40 degrees it will
        be send the command to steer hardly */
    else if(orientation>=10 && pause_flag == 0){
      if (orientation>10 && orientation<=40){
       strcpy(command,"k");
      }
      else if (orientation>40) {
        strcpy(command,"h");
      }
    }

    // If while playing board is not turned, command to accelerate is sent
   else if(orientation < 10 && orientation > -10 && pause_flag == 0){
      strcpy(command,"z");
    }

  // Flag is reset
  jump_flag = 0;
}
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  /* Interrupt created to handle blue push-button (B1_Pin), in case it is pressed
   * the global flag is set to 1 and an action will be initiated in the main*/

  if(GPIO_Pin == B1_Pin)
  {
    button_flag = 1;
  }
  else{
    __NOP ();
  }
}
```

Then we transmitted these commands across an USART protocol to the PC(setting a delay of 50 milliseconds not to make data stream hard to read) using the following HAL function:

```
// Sending commands through serial port
HAL_UART_Transmit_IT(&huart2,command,strlen(command));

HAL_Delay(50);
```

Eventually the obtained results are summarized in Tab. 2.

DISCLAIMER: we have changed some of the predefined keyboard button commands in order to avoid problems of misinterpretation of some commands, these features can be modified from the control setting icon on the left corner of the game screen.

Action	**Command**	**Conditions**
Pause	"p"	Acceleration on z $>$ 970mg Game not paused
Play	"p"	Acceleration on z $<$ 900mg Game paused
Jump	"e"	Angular velocity on x $> 100000\ \frac{mdeg}{s}$ Game not paused
Use object	"r"	Blue button pressed
Move forward	"z"	$-10° <$ Horizontal angle $< 10°$ Game not paused
Turn left(soft)	"l"	$-40° <$ Horizontal angle $< -10°$ Game not paused No jump
Turn left(hard)	"f"	Horizontal angle $< -40°$ Game not paused No jump
Turn right(soft)	"k"	$10° <$ Horizontal angle $< 40°$ Game not paused
Turn right(hard)	"h"	Horizontal angle $> 40°$ Game not paused
No action	"0"	No other condition met

Table 2: Commands and corresponding actions

3.5 Commands elaboration on the PC

A python script was written to convert the commands computed by the Nucleo board into actually pressing and releasing keyboard buttons. First of all, we import the libraries to acquire data from the serial COM port of the pc, to control the keyboard and to manage time functions:

```
import serial
from pynput.keyboard import Key, Controller
import time
```

Then we create a controller object named keyboard:

```
keyboard = Controller()
```

and we set the serial port setting:

```
# Configuration of serial communication
serial_port = serial.Serial()
serial_port.baudrate = 115200
serial_port.port = 'COM3'
serial_port.timeout = 1
# Initialize error in data reading to False
error = False
```

A delay of 5s is set to allow the user to open “Mario Kart 64” game after this python code is launched:

```
# Gives you time to open the desired target for the inputs
time.sleep(5)
```

We try to access the serial data provided by the Nucleo board:

```
# Open serial port communication
try:
    serial_port.open()
except:
    error = True
```

In case something went wrong either in data transmission or in port configuration, we exit the script:

```
# Check for error presence in communication
if not serial_port.is_open or error:
    print("Something went wrong!")
    exit()
```

If this operation is succeeded the script reads the data from the COM port and decodes it as ASCII encoded data. These data are saved in the “key” variable. Using the methods “keyboard.press(parameter)” and “keyboard.release(parameter)” the code presses and releases the parameter as the user would manually press and release the relative keyboard button. Our code works in two steps:

- Perform the command transmitted by the board: The key value is pressed, then the code maintains the button pressed for 0.01s, eventually releases it. Concerning the steering commands, if the orientation angle is between ±10°and ±40°the soft steering is implemented by pressing the left (”f”) or right (”h”) button for 0.01s, instead if the angle is above ±40°the hard steering is implemented by pressing them for 0.03s (these values were found by trial and error).

- Continuously accelerate: The same operation is done after for the “z” command (acceleration command) In this way the kart always accelerates, and at the same time performs the commands provided by the Nucleo board.
  Initially the blue button was intended to be used as the throttle command, but we changed our mind as it revealed to be very uncomfortable, so we left the throttle command to be ”automatic”: doing so, the outcome of the controller was not compromised as the game does not allow neither gradual accelerations nor braking.

```
while 1:
    try:
    # In case some data is sent from serial port, store it in key
        if serial_port.in_waiting!=0:
            key = serial_port.read().decode('ascii')
            # If pause command is sent, press it for 0.1 s
            if key == "p" :
                keyboard.press(key)
                time.sleep(0.1)
                keyboard.release(key)
            # If 0 is passed, do nothing
            elif key == "0":
                pass
            # If soft steer left command is passed, steering
            #button is held pressed for shorter time (together with
            #acceleration button)
            elif key == "l":
                keyboard.press('z')
                keyboard.press('f')
                time.sleep(0.01)
                keyboard.release('f')
             # If soft steer right command is passed, steering
             #button is held pressed for shorter time(together with
             #acceleration button)
            elif key == "k":
                keyboard.press('z')
                keyboard.press('h')
                time.sleep(0.01)
                keyboard.release('h')
            # All other commands are performed keeping buttons
            #pressed for 0.03 seconds
            else:
                keyboard.press('z')
                keyboard.press(key)
                time.sleep(0.03)
                keyboard.release(key)
    except:
        pass
```

4 Game tutorial

To play the game, as already said, we need first of all to run the python script then we have to switch immediately the screen to the Mario Kart 64 game, prepared previously (the game can be found at Super Mario Kart 64). As already anticipated in subsection 3.4, the Nucleo board allows to perform commands shown in Tab. 3 on the kart.

Command	**Action**
Pause	Lay the board on the table
Play	Lift the board from the table
Jump	Rotate the board toward player's chest
Use object	Press the blue button
Turn left	Turn the board left
Turn right	Turn the board right
Acceleration	Hold the board still (Keep it horizontal)

Table 3: Game commands achievable using Nucleo board

5 Conclusion

Overall we can deem satisfied of the outcome of the project: we managed to actively re-elaborate what we had done during the laboratories and develop a simple, yet effective, code for the controller. After several attempts performed during the test stage, the kart has behaved as expected on the game, with a obvious small delay due to the intrinsic limit of the online game itself. The movements of the board are all well captured and the commands are properly translated.