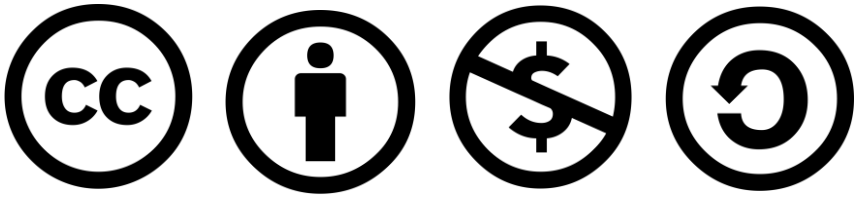

EMBEDDED SYSTEMS PROJECT

Air mouse - Fingers

Lecturer: Francesco Moscato - fmoscato@unisa.it

Group members:

Ranieri Speranza	0622701687	s.ranieri4@studenti.unisa.it
Mancusi Ernesto	0622701668	e.mancusi2@studenti.unisa.it
Parrella Giuseppe	0622701665	g.parrella9@studenti.unisa.it
Sonnessa Francesco	0622701672	f.sonnessa@studenti.unisa.it

Summary

Introduction

The aim of this document is to describe the design and development of an Air Mouse, carried out as the final project of the Embedded Systems course.

This document is attached to the final project as official documentation, in order to provide a complete description of the functional and non-functional requirements of the system, as well as the hardware components and software architecture used.

Requirements

High level description

It is required to carry out the design and implementation of an Air Mouse that allows, through two accelerometers placed on two fingers of one hand, to simulate a mouse.

Specifically, it is required to reproduce, through gestures, the following commands:

- Up/Down/Left/Right movements of the cursor.
- Left Click and Double Click.
- Zoom In and Zoom Out.
- Alt+Tab.

In addition to the hand motion, the logging of gestures made by the user is required. Specifically, when an event is detected after the movement of the accelerometers, a log string must be generated (e.g.: `[from:(x,y,z); to:(x,y,z); accel: (x,y,z); gyro: (x,y,z)]`) taking, from the RTC module, the timestamp related to the gesture to record the instant when the event is detected/executed. This string will need to be uploaded on a Micro SD card, through its reader module.

Functional Requirements

FR.1. The system must make the cursor move in the right, left, up, and down directions when the corresponding motion is detected.

FR.2. The system must make the left click command when the associated gesture is detected.

FR.3. The system must make the Zoom In command when the associated gesture is detected.

FR.4. The system must make the Zoom Out command when the associated gesture is detected.

FR.5. The system must make the ALT+TAB command when the associated gesture is detected.

FR.6. The system must be able to switch between the current open windows of the O.S., after sending the ALT+TAB command and when the associated gesture is detected.

FR.7. The system must perform the ALT+TAB release command (i.e., close the currently open windows screen) after sending the ALT+TAB command and when the associated gesture is detected.

FR.8. For each of the above events, the system must save a log string inside the Micro SD card with the following format:

[dd-mm-yy HH:MM:SS] EVENT: *description of the detected event* ========

Acc. Values:{T: Gx,Gy,Gz Ax,Ay,Az \|\| I: Gx,Gy,Gz Ax,Ay,Az}

The log must be saved in a file named **`dd_mm_yy.TXT`**

Non-functional requirements

NFR.1. Movements that are detected on the computer must appear smooth to the human eye on a screen of 60Hz.

Proposed Solution

An Air Mouse has been created, the physical implementation of which is shown in Figure 1.

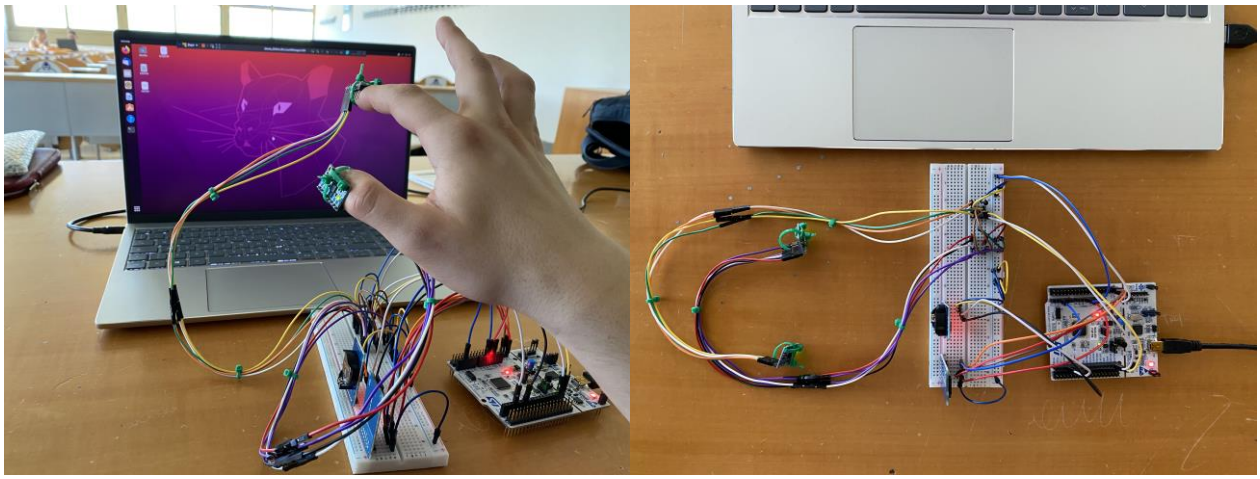

Figure 1: Air Mouse - physical implementation

Several components were used, in particular

- **STM32F401RE6**: the main microcontroller based on ARM technology which almost all operations are carried out.
- **MPU6050**: two 6-axis gyroscope and accelerometer sensors used to capture the hand gestures.
- **RTC DS3231**: used to get the timestamp of an event.
- **Micro SD card Module**: used, with the RTC, to write on the Micro SD Card a representation of the event (e.g.: `[from:(x,y,z); to:(x,y,z); accel: (x,y,z); gyro: (x,y,z)]`)
- **SanDisk Micro SD card**: 2GB, FAT file system, 7 MB/s writing speed, 10 MB/s reading speed.

To see how the air mouse works in a video tutorial you can go here while you can find the source code here.

Design Choices

This section will explain the design choices made to realize the project.

It was chosen to place the two accelerometers respectively on the thumb and index finger of the user's right hand, as shown in *Figure 1*.

The accelerometer on the index is used to move the cursor on the PC screen, as well as for clicking and double-clicking. The accelerometer on the thumb, on the other hand, is used for making the gestures associated with the Zoom In and Zoom Out commands and, in conjunction with the one on the index, for the ALT+TAB commands.

About mouse movements, it was chosen to send only the relative displacement to be made with respect to the current position of the cursor.

Selected Gestures

The design choices made, regarding the mapping between finger movements and mouse actions for each of the commands to be implemented, are illustrated below.

Right movement

We have encoded the right movement of the mouse with the movement of the index's accelerometer shown in the in Figure 2.

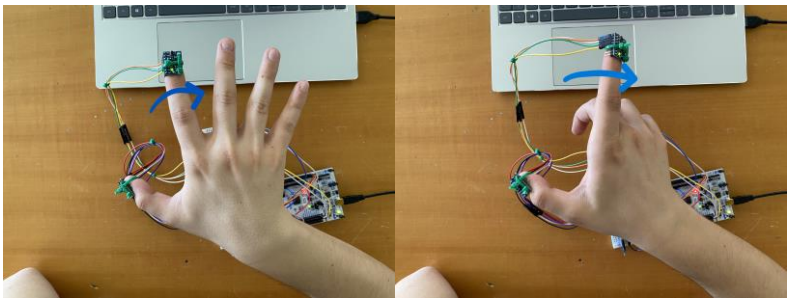

Figure 2: Right movement

Left movement

We have encoded the left movement of the mouse with the movement of the index's accelerometer shown in the in Figure 3.

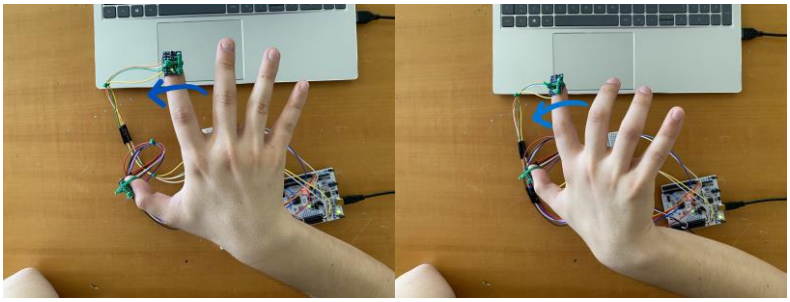

Figure 3: Left movement

Up movement

We have encoded the up movement of the mouse with the movement of the index's accelerometer shown in the in Figure 4.

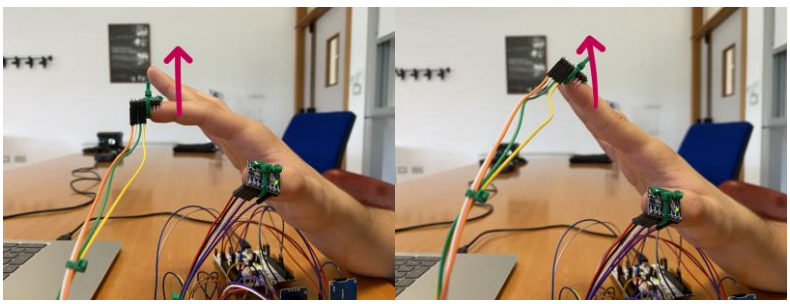

Figure 4: Up movement

Down movement

We have encoded the down movement of the mouse with the movement of the index's accelerometer shown in the in Figure 5.

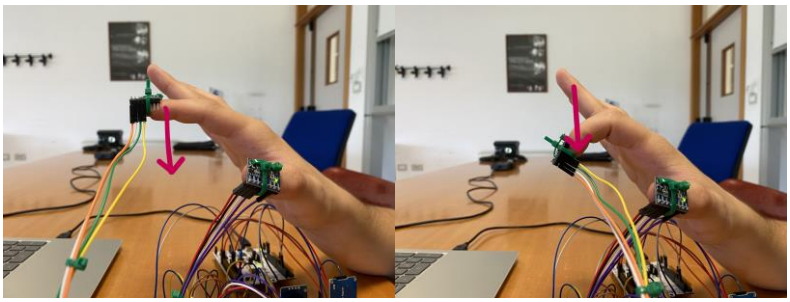

Figure 5: Down movement

Left Click

We chose to encode the left click with a sequence of movements of the index's accelerometer: first downwards and then upwards, returning to the initial position as shown in Figure 6.

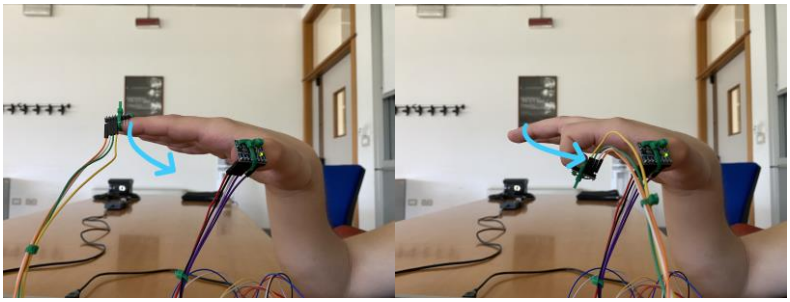

Figure 6: Left Click

Double Left Click

We chose to encode the double click with the sequence of movements associated with the single click repeated twice in at most 600ms, as shown in Figure 7.

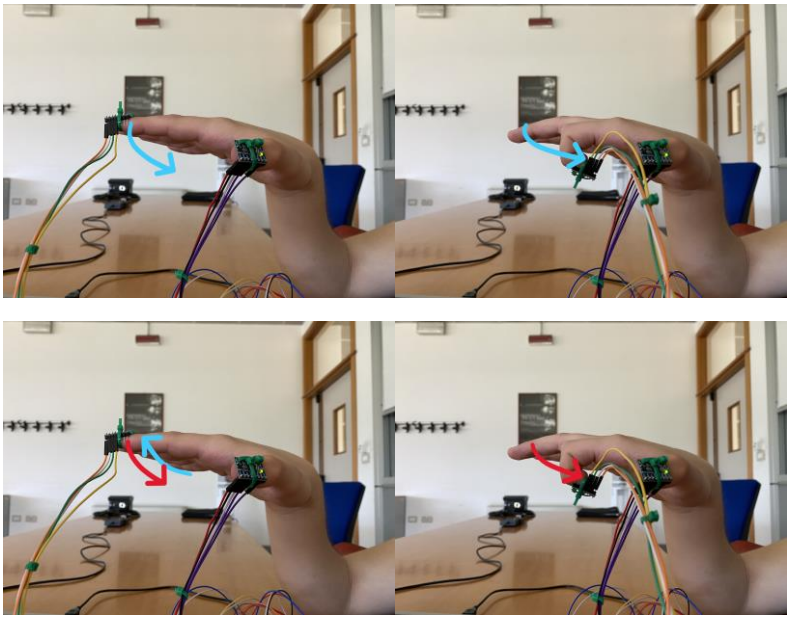

Figure 7: Left double click

Zoom In

We chose to encode the Zoom In command by moving the thumb's accelerometer towards the inside of the palm of the hand, and then returning to the initial position, as shown in Figure 8.

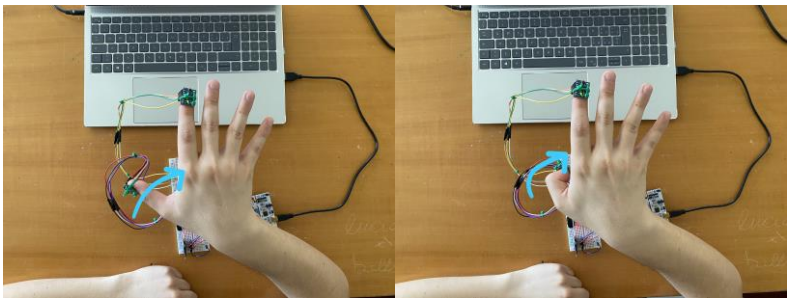

Figure 8: Zoom In

Zoom Out

We chose to encode Zoom Out command with the sequence of movements associated with Zoom In command repeated twice in at most 600ms, as shown in Figure 9.

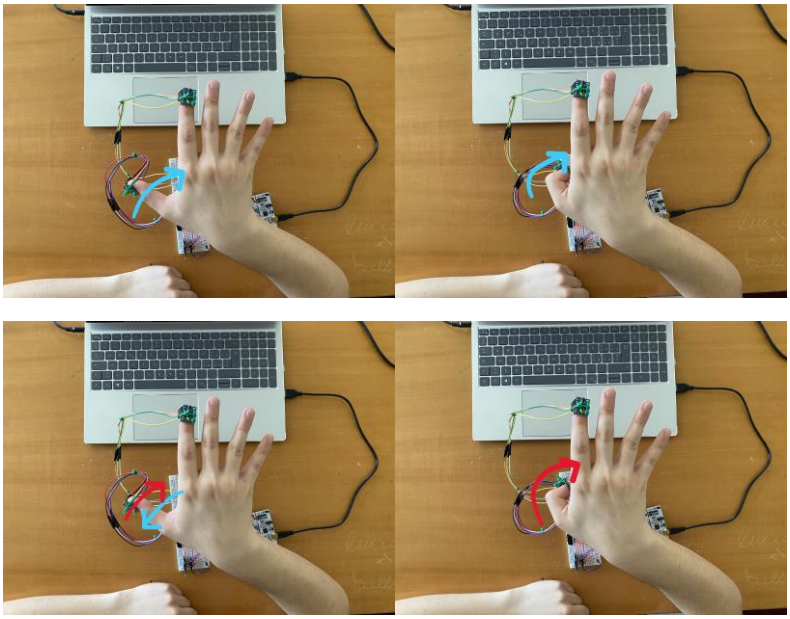

Figure 9: Zoom Out

Alt+Tab

We chose to encode the ALT+TAB command with a hand transition from horizontal to vertical, as shown in Figure 10.

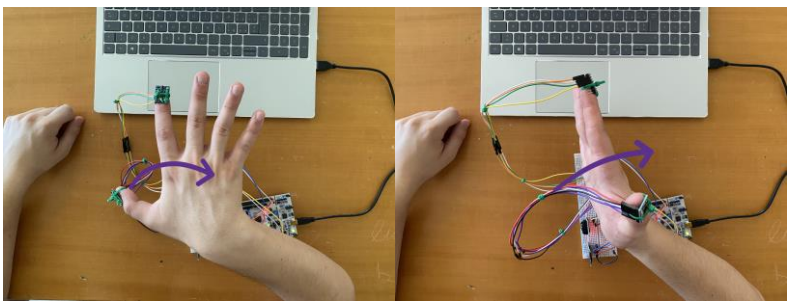

Figure 10: Alt+Tab

Alt+Tab – Windows Switch

We chose to encode the command to switch between open windows, displayed after the ALT+TAB command, with a movement of the index finger like left clicking (it is as if one were making a left click but with the horizontally hand), as shown in Figure 11.

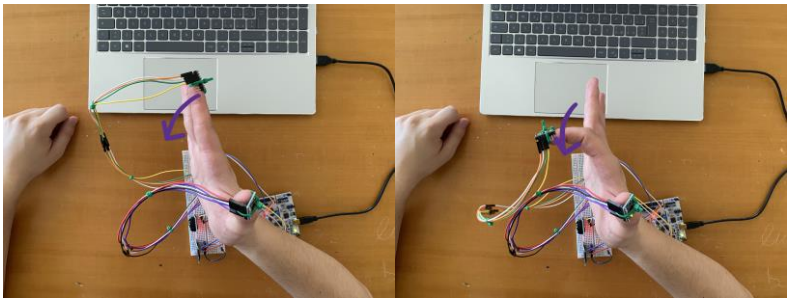

Figure 11: ALT+TAB Windows Switch

Alt+Tab – Release

We chose to encode the command to release the ALT+TAB, and thus open the currently selected window, as a hand transition from vertical to horizontal, as shown in Figure 12.

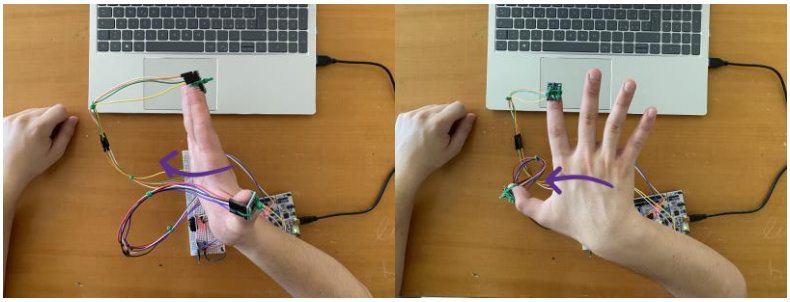

Figure 12: ALT+TAB release

Hardware architecture

The following table shows the main hardware connections, which are also schematized in Figures 13 and 14.

STM32		
Component	**Pin**	**Protocol**
MPU6050 (x2) *	SDA:PB9 – SCL:PB8	I^2C
RTC DS3231	SDA:PB3 – SCL:PB10	I^2C
SPI Card Reader	SCK:PC10 – MISO:PC11 – MOSI:PC12	SPI
Power supply	3.3 V – 5 V**	-

***The accelerometer on the thumb has AD0 pin connected to 3.3V in order to change his address from 0x68 to 0x69.
****The 5 V power supply is used only for the SD Card sensor.

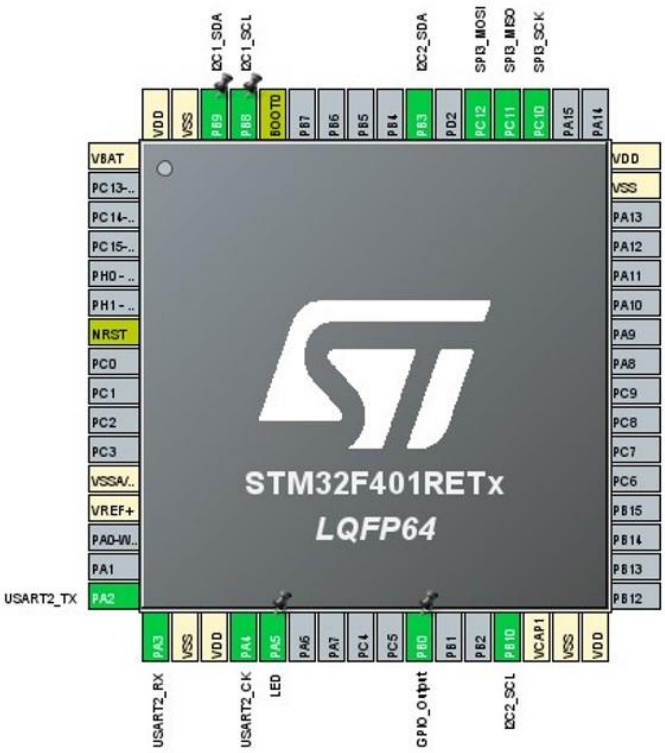

Figure 13: ioc scheme

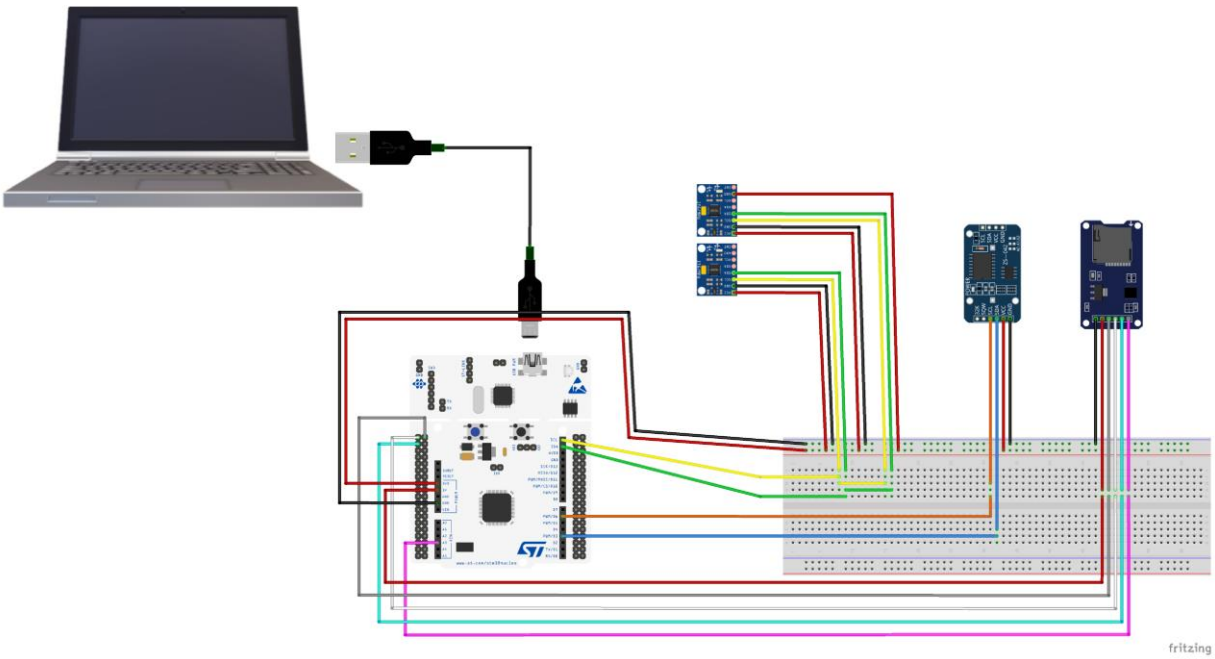

Figure 14: Fritzing scheme

Software architecture

High-level architecture

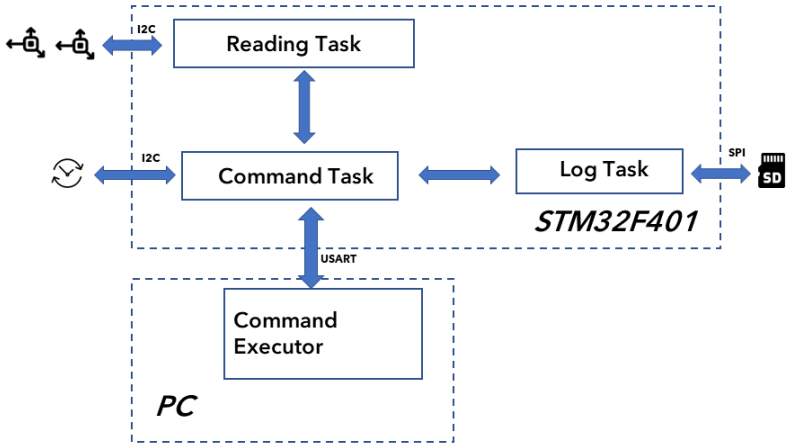

Figure 15: High-level Architecture

Figure 15 shows a high-level software architecture of the system, based on tasks. It can be noted that regarding the organization of the software in the STM32F401RE6 board, three main tasks have been identified:

1. *Reading Task*: reading of accelerometers measurements.
2. *Command task*: detection of commands carried out by the user.
3. *Log Task*: writing log strings to the Micro SD card.

The figure also shows the communication between the tasks and the communication between them and external devices, indicating the protocols used for this purpose.

On the PC (equipped with Ubuntu O.S.) side there is a bash script, indicated with *Command Executor*, which communicates with the MCU through the USART protocol, interprets the received strings as commands, and then executes them through the tool `xdotool`[1].

[1] https://manpages.ubuntu.com/manpages/trusty/man1/xdotool.1.html

Detailed design

As for the organization of the code, in Figure 16 it is possible to see the main libraries used within the project.

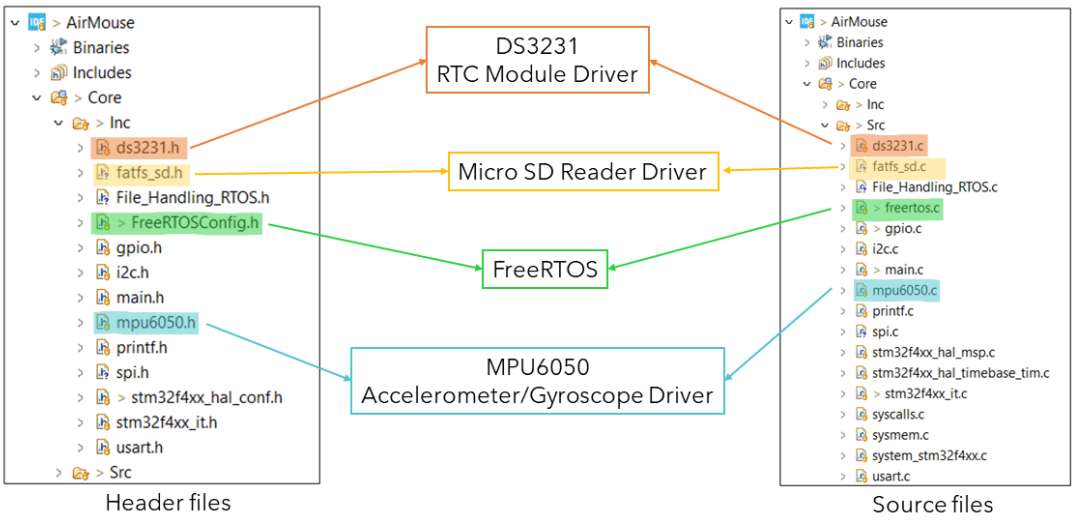

Figure 16: main libraries

It is possible to note that there are libraries relating to the drivers of the physical components used within the project, while the fundamental part of the application logic is present within the `freertos.c` library. The initialization activities of the necessary components for the project are carried out within the `main.c` file before the tasks described below are started.

Additionally, an external library has been included through the `printf.h` and `printf.c` files. This library contains the implementation of the `sprintf()` function and other functions related to string management, specifically designed for embedded systems. Among the features of this library, we note a good management of memory resources (already poor on the MCU), with the absence of dependencies on other modules (the common `sprintf()` requires the entire `stdio` lib and all their dependences), and the ability to operate with different threads. Other features of the library are reported in the indicated repository[2].

Tasks

For the implementation of the general architecture, shown in Figure 14, three tasks were used, subsequently scheduled with FreeRTOS:

1. `ReadTask`: it reads the measurements of the accelerometers periodically, every 25 ms.
2. `CommandTask`: after reading by `ReadTask`, it detects the command requested by the user, and writes the event into a queue for the logging activity.
3. `SDTask`: it writes the log strings on the Micro SD card, after the detection of each command carried out by `CommandTask`. This task is performed periodically every 100 ms, so that it empties the queue containing the log messages not yet written on the Micro SD.

[2] https://github.com/mpaland/printf

As regards the scheduling of the three tasks, sequential scheduling was chosen, as they are performed progressively, in the order indicated in the previous list. This sequential scheduling is imposed by disabling the *preemption* within the kernel settings of FreeRTOS and assigning the following priorities to the tasks:

- `ReadTask: osPriorityNormal1;`
- `CommandTask: osPriorityNormal;`
- `SDTask: osPriorityNormal2;`

This results in a sequential scheduling that involves 4 executions of ReadTask and CommandTask in an alternate fashion, and the execution of SDTask, which empties the log queue. So, at each execution new measurements are read, any detected movements are sent to the PC and the gestures are logged into a file on the Micro SD.

ReadTask

The first task reads from the two accelerometers, after which it pauses for 25 ms. This choice made it possible to comply with the NFR1 requirement relating to the fluidity of cursor movements: the movements and commands sent by the user are captured correctly, without there being a delay in reading that affects the fluidity of the mouse. In fact, by increasing the period of this task, significant accelerometer measurements may not be detected between two consecutive readings, causing the system to respond more slowly to the user's wishes.

CommandTask

The second task is scheduled immediately after the ReadTask and, after its execution, it pauses for 25 ms. This task, in accordance with the design choices of the mentioned above gestures, detects the commands that the user sends to the PC. Each time a command is detected, it is sent to the PC through the USART and, at the same time, the log string associated with the same command is saved within a queue called `EventQueue`. This queue is managed through the features offered by FreeRTOS (the CMSIS-RTOS2 Version 2.1.3 API was used for this purpose). Subsequently, the contents of the queue will be processed by another task that has the purpose of writing log strings into the Micro SD.

The strategies used for detecting the commands sent by the user, starting from the values of the accelerometers, are presented below.

Mouse movement

Regarding the mouse movement on the screen, the gravitational acceleration components measured on the x and y axes of the index's accelerometer were used. In fact, with these values, it is possible to calculate the relative displacement, in pixels, of the cursor along the same axes of the screen with respect to its last position. The calculation of displacement is made consistent with the movements in Figures 2, 3, 4 and 5.

The adopted convention of the screen axes is shown in Figure 17.

Finally, the command relative to this displacement is sent.

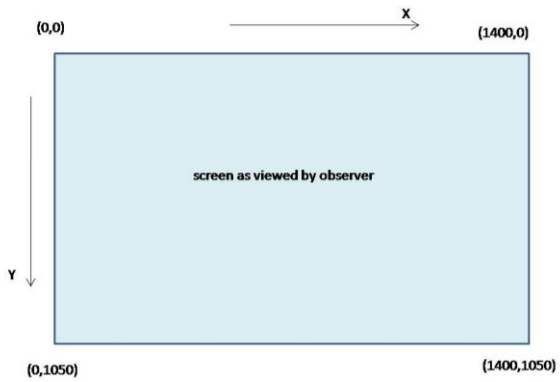

Figure 17: axis convention example

Click and double click

Regarding the click detection, an analysis of the accelerometer values associated with the gesture chosen for this command, previously shown in Figure 6, was carried out[3].

In the following figure you can see which are the variations of the values of the index's gyroscope when the gesture of the click or double click is made.

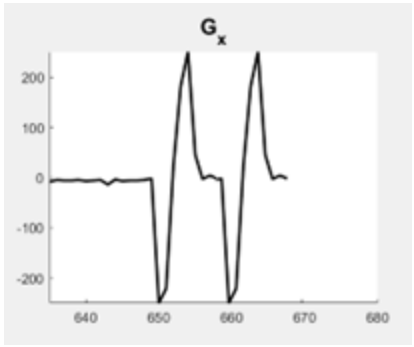

Figure 18: gyroscope variations for click/double click gestures

Analyzing the figure, it is possible to notice that when a click is made significant changes occur of the values on the x axis, with respect to the initial position, whose value is 0.

From the figure it is clear that to carry out a click and to return to the initial position, corresponds to having a negative variation (finger movement downwards) followed by a positive variation (the finger returns to the initial position).

For this purpose, the negative variation was detected and reported via a flag named $clickFlag$. This flag prevents those measurements, taken after the negative variation, from being interpreted as mouse movements, thus avoiding unwanted cursor movements.

At this point, if a positive variation is detected, it means that the user's finger has returned to the initial position, and this movement must therefore be interpreted as a click. A variable called $numClick$ is used to consider the number of times the movement just described has been made.

In the case of double clicking, this "pattern" is repeated twice in a short time. For this reason, its detection, once a single click is detected, the command is not immediately sent to the PC, but

[3] We have carried out this analysis with MATLAB real time plot. The script used is also attached to this project.

it remains waiting always for that short time, to check if to the single click will follow another one and then interpret the entire movement as double click.

In particular, to detect a double click, after the movement related to the single click a timer, realized by taking advantage of the timing features offered by FreeRTOS[4], is started. This timer expires after 600 ms, after which a callback is performed, that:

1. If it detects a click, it sends the click command.
2. If it detects a double click, it sends the double click command.

So, this callback will launch the appropriate command.

Zoom In and Zoom Out

Regarding the Zoom In detection, an analysis of the accelerometer values associated with the gesture chosen for this command, previously shown in Figure 8, was carried out.

In the following figure it is possible to notice the variations of the values of the thumb's gyroscope, when the gesture of Zoom In or of Zoom Out is carried out.

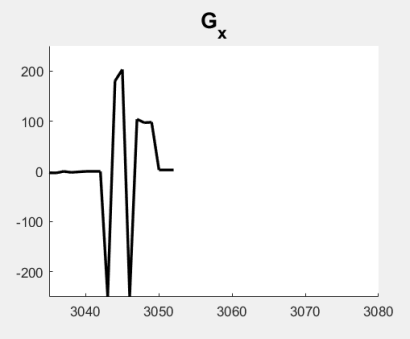

Figure 19: gyroscope variations for zoom in/zoom out gestures

Since the movement and, consequently, the detected variations are very similar to those just analyzed by click and double click, the logic used for the detection of such gestures is the same as the one described in the previous paragraph. Also in this case, for their detection, the timing functions offered by the FreeRTOS API were used.

ALT+TAB, window switching and ALT+TAB release

ALT+TAB

Regarding the detection of the ALT+TAB command, an analysis of the values of the accelerometers associated with the gesture chosen for the command, previously shown in Figure 10, has been carried out.

In the following figure it is possible to see the variations of the values of the thumb's gyroscope when the gesture in examination is carried out.

[4] https://www.keil.com/pack/doc/CMSIS/RTOS2/html/group__CMSIS__RTOS__TimerMgmt.html

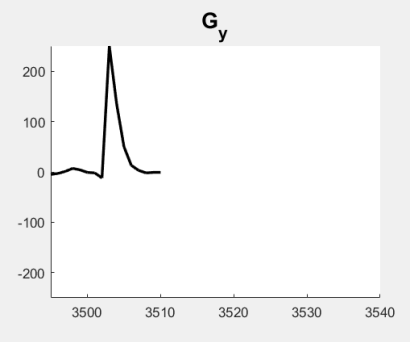

Figure 20: gyroscope variations for ALT+TAB gesture

Analyzing the figure, it is possible to notice that, when the gesture associated with this command is made, a positive variation occurs on the y-axis, with respect to the initial position whose value is 0 (movement of the hand from horizontal to vertical with a stretched thumb).

For this purpose, this change was detected and reported through a flag called ALT_TAB_flag. This represents the entry in the ALT+TAB mode that corresponds, on the PC side, to the pressure of the relative command with attached display of the open windows and the possibility of executing the commands, later described, of Windows Switching and ALT+TAB release.

It was also introduced another flag called ALT_TAB_move which is used for different purposes:

- When entering ALT+TAB mode ($ALT\_TAB\_flag \rightarrow 1$), the purpose of this flag is to ensure that during the transition to this mode, some possible movements are not interpreted as window switch commands that the user has not voluntarily made. In this case, $ALT\_TAB\_move = 0$ is set and it is set again to 1 by a timer after 120 ms, determining the end of this transition and the possibility to accept windows switch commands.

- When you are already in ALT+TAB mode ($ALT\_TAB\_flag = 1$), this flag is intended to prevent the windows switching command from being detected multiple times. This is because this command is detected by examining a negative variation on the x-axis of the index's gyroscope: the values may remain negative for several consecutive measurements.

- When exiting the ALT+TAB mode ($ALT\_TAB\_flag \rightarrow 0$) the flag is intended to prevent some movements from being inadvertently considered as cursor movements when making the vertical to horizontal hand transition, interdicting them until the stabilization of the hand in the horizontal position.

The use of these flags in these different ways, along with the timers and their callbacks, allows you to simply manage the possible transitions: considering that the time interval between one measurement and the other is very narrow, a complete transition (e.g. a gesture made) could be captured in a larger measurement window, in which case the flags are used to capture these windows appropriately.

Window switching

As for the windows switching in ALT+TAB mode, an analysis of the values of the accelerometers associated with the gesture chosen for this command, previously shown in Figure 11, has been carried out.

In the following figure it is possible to see which are the variations of the values of the index's gyroscope when the gesture is carried out.

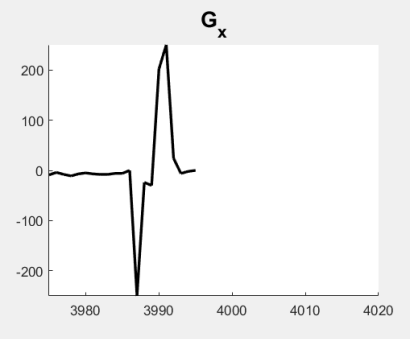

Figure 21: gyroscope variations for window switching gesture

Analyzing the figure, you can see that when this gesture is made, significant changes occur, with respect to the initial position whose value is 0, on the x axis. In addition, from the graph it is possible to notice that the pattern associated to the action of window switching coincides with that used for the detection of the single click.

There is a negative variation (movement of the index towards the inside of the hand) and to be interpreted as a window switching command, requires it to be in the ALT+TAB mode (and so we have $ALT\_TAB\_flag = 1$).

After executing the command, the ALT_TAB_move flag described above, is forced to the value 0 to prevent multiple consecutive windows switch commands from being detected for user movement. This could happen because, given the high reading frequency, among several contiguous measurements there are several that meet the condition and that could be interpreted as windows switch.

ALT+TAB Release

Regarding the release of the ALT+TAB, an analysis of the values of the accelerometers associated to the gesture chosen for the command, previously shown in Figure 12 has been carried out.

In the following figure it is possible to see which are the variations of the values of the index's gyroscope when the gesture in examination is carried out.

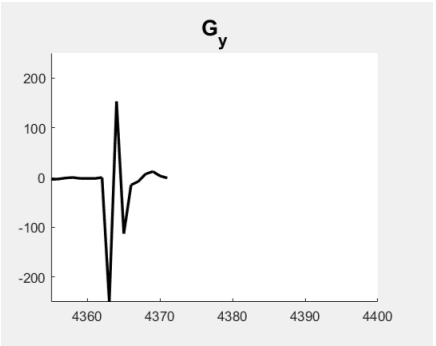

Figure 22: gyroscope variations for ALT+TAB release gesture

Analyzing the figure, it is possible to notice that when the gesture associated to such command is carried out, significant variations happen, regarding the initial position whose value is 0, on the axis y.

There is a negative variation (movement of the hand from vertical to horizontal), and to be interpreted as a command related to the ALT+TAB Release, requires it to be in the ALT+TAB mode.

After executing the command, the *ALT_TAB_flag* is forced to the value 0 to indicate the exit from the ALT+TAB mode, while the *ALT_TAB_move* flag is forced to the value 0 for the same reason described above; At this point a timer is started at the end of which the flag is returned to its initial value, indicating the end of the command.

SDTask

The third task has a period of 100 ms. We have chosen this time value to not degrade the fluidity of the mouse. It was found that writing an event immediately after its detection generated a mouse lag due to the time required by writing operation on Micro SD. Therefore, it was decided to buffer events in a queue and write them in batches.

Considering that the `CommandTask` writes in the queue every 25 ms, after 100 ms there will be at most 4 events (not necessarily to every cycle an event is generated, e.g. the user remains stationary), already executed but not yet registered on the Micro SD, which will constitute the batch.

Then this third task empties the queue, using the `f_write` operation to write the batch of events inside the log file. At the end of these writes, the `f_sync` function is invoked so that writing is finalized and then the batch is written to the Micro SD card. Note, in fact, that the `f_write` function does not write to files immediately, except after a calling of `f_close` or `f_sync` function.

We have chosen the `f_sync` function to avoid continuously closing and reopening the file, saving time and, at the same time, respecting the writing specification on the Micro SD card.

The file is then opened only once, during initialization, with the `f_open` function. Then the SDTask task empties the queue containing the event logs every 100 ms, by using the `f_write` and `f_sync` functions to write correctly on Micro SD.

In conclusion, it was noted that the logging of events carried out in this way does not slow down the fluidity of the mouse, allowing compliance with the requirements FR10 and NFR1.

Software Protocols

The following communication protocols were used in the project:

I²C

I²C protocol is used for communication with the two MPU6050 accelerometers and with the RTC DS3231.

SPI

SPI protocol is used for communication with Micro SD card to write event log.

USART

The USART protocol is used to define the software protocol used by the MCU to send commands to the PC.

We chose to use the synchronous version of this protocol to prefer a fast and effective communication between MCU and PC, so that it does not slow down the execution of commands sent by the MCU. Specifically, we used the USART with a baud rate of 115200 Bits/s and the

`Transmit Only` mode, as the MCU must not receive messages from the PC, creating a one-way channel.

The detection part of the commands to be carried out, as well as the logging activity, is left to the MCU while the PC executes the received commands.

The MCU, after detecting an event, sends the string ***"xdotool ..."*** which represents, in the form of a Linux command, the command that the user wishes to execute.

On the PC side, this string is interpreted as a Linux command and executed. We have chosen to send directly strings in form of a Linux command to avoid wasting time on PC side due to string pre-processing before the execution of the command.

This has allowed to avoid the use of programs written in high-level language that, once received an encoded version of the command to be executed, send said command to the O.S. through an interpretation of that string.

Finally, a bash script was used to receive commands from the serial port and execute them as Linux commands, avoiding the additional overhead given by libraries of high-level programming languages.

How to Run

It's necessary to clone the repository in the STM32CubeIDE and load the project on the STM32 board.

On PC side, you can run the `commandScript.sh` script with the following command:

```
sudo ./commandScript.sh
```

After that, the commands received from the serial port are executed (note that in the script the default serial port is `/dev/ttyACM0`[5], you can modify it by inserting the correct serial port detected in your PC for STM32).

You can also execute the mouse commands by opening a Linux terminal and executing the following commands:

```
1. sudo bash
2. exec<>/dev/ttyACM0
```

[5] Note that ACM0 could be ACMx with x=1,2, …