Cost-effective Arduino-controlled Flight Simulator 2004/X General Aviation Radio Stack

1st Felipe Vaiano Calderan
Science and Technology Department
Federal University Of São Paulo
São José dos Campos, Brazil
fvcalderan@gmail.com

***Abstract*—Flight simulation has been present in civil and military training for decades now. With them, pilots and student pilots can practice from simple tasks, such as taking off and landing the plane, to rare and complex situations, like very aggressive weather conditions and many types of failures. While dedicated flight-instruction institutions and airlines have the means to buy or build sophisticated hardware for flight simulation, hobbyists and aspiring pilots typically do not have the money to invest in such. Taking into account the demand for more pilots, it is essential to increase the accessibility of flight simulation for everyone, so that potential pilots can develop their skills and love for aviation. With this in mind, here it is proposed a very cost-effective solution to control, via dedicated hardware using Arduino, the radio stack, and autopilot system of most General Aviation aircraft in Microsoft Flight Simulator 2004 and X (with minor tweaks).**

***Index Terms*—cost-effective, embedded systems, arduino, flight simulation, radio stack, autopilot**

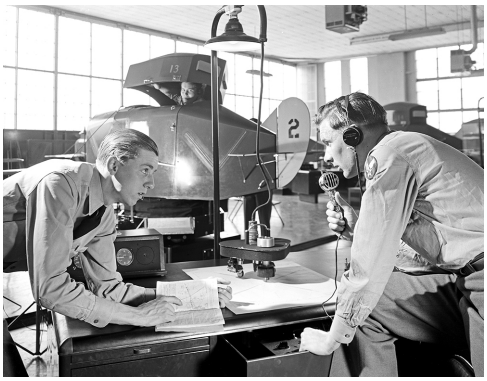

Fig. 1. Link Aviation flight simulator, by Robert Yarnall Richie and DeGolyer Library, Southern Methodist University, No restrictions, via Wikimedia Commons.

I. Introduction

When the term flight simulator is used today, it refers to a combination of systems that allow people to have a digital experience of controlling an aircraft, which would be similar to the real life counterpart. Although it may not sound correct, flight simulators are older than the digital age. The first well-known flight simulator is the Link Trainer (Figure 1), from 1929 [1]. It allowed pilots to operate aircraft-like controls and see the effect through the simulated flight instruments (which is still very relevant today [2]), built using pumps, valves, and bellows.

For the most part, in the early stages of simulation, the focus was more military training, but in the 1950s, airlines were already buying considerable amounts of simulators to train and improve the skill of their pilots. Not only that, but simulators were already being used to check aircraft performance before it left the ground [3], which indicates that there was good accuracy, even 60 to 70 years ago.

Of course, as the technology advanced, so did the simulators and in early 1983, Bruce Artwick created the Sublogic Flight Simulator, which used non-cockpit hardware (everything on-screen) [4]. At this point, digital computers were already popular for simulation purposes, as they could compute very sophisticated models and fit in smaller places than the old analog computers.

In general, from this point onwards, simulators became very important tools to improve flight safety while reducing training costs. Today, they are not only essential to military organization and civil airlines, but also to aircraft makers, to test the physics and behaviors of the plane, before it even exists. With this, standardization of these tools took place throughout the world and tend to become progressively more popular [5].

By 2018, modern professional-grade civil simulator (Figure 2) fleet neared 1300 units [6] and the competition for simulation hardware, software, and training centers became fiercer, with companies like Boeing and Airbus competing with smaller ones for the market [7]. Unfortunately, few people have access to these high-level simulators, which can be detrimental to the arousal of the curiosity of many people who could develop love and great skills for aviation, therefore negatively affecting the amount of potential new pilots.

This fact and the existence of home-use flight simulators, like the Microsoft Flight Simulator[1] and Laminar Research X-Plane[2] series, prompted the scientific and enthusiastic community to create accessible alternatives to full-size professional-grade simulators [8]. Of course, one could always use a keyboard and mouse to control the aircraft in home-use flight simulators, but this results in a subpar experience, even for hobby purposes.

[1]https://www.flightsimulator.com/
[2]https://www.x-plane.com/

Fig. 2. Baltic Aviation Academy Airbus B737 Full Flight Simulator (FFS) in Vilnius, by Tadas1980, Public domain, via Wikimedia Commons.

The system proposed here provides this community with a very cost-effective dedicated-hardware way to control the radio stack and autopilot systems of most small General Aviation (GA) aircraft (with emphasis in those without glass cockpits [9]). It is built using an Arduino Uno (but many other Arduino/Genuino boards could be used in its place) and cheap electronic components.

The system was particularly modeled to operate many of Microsoft Flight Simulator 2004's (FS9) steam gauged GA plane's stacks and implements most of the functionality. It is noteworthy that it is extensible and can be improved to support bigger aircraft features and more realistic input methods. The tests using the device in simulated flights show satisfactory performance (mainly measured in usability), specially considering the low price-point and extensibility.

II. Home flight simulator software today

Simulators were very popular throughout the 1990s and beginning of the XXI century, when acceptable realism in computer graphics and physics was possible for the first time. Of course, the launch of new Microsoft Flight Simulator games have a big positive impact on the genre's popularity. This explains why 2004 and 2006 (launch years of FS9 and FSX, respectively) were very active years for the community. Unfortunately, after 2006, Microsoft went 14 years without publishing a new title and this affected the popularity dramatically, a streak ended by the launch of Flight Simulator 2020, which revived the interest in the genre. Figure 3 shows the Google Trends for the search term "flight simulator", which reflects the popularity of flight simulators on the internet.

Looking at the trends for other simulators, such as "X-Plane", "Aerofly" and "Prepar3D", it is possible to see that, although they compete with the Microsoft product, they don't really have a significant market share. It is also worth mentioning that at the time of this publication, FS2020 has almost triple the reviews (close to 45000)[3] on Steam compared to X-Plane 11, which is 3 years older[4]. Figure 4 shows the Google Trends for different simulators.

[3]https://store.steampowered.com/app/1250410

[4]https://store.steampowered.com/app/269950

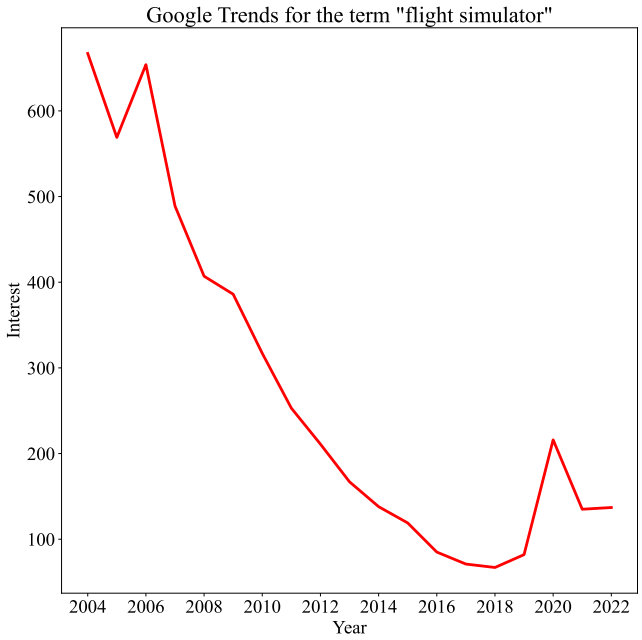

Fig. 3. Google Trends for the search term "flight simulator". The genre became niche in the last years, but 2020 shows a resurgence in interest, with the launch of Microsoft Flight Simulator 2020 (FS2020). The Interest axis represents the global search interest relative to the highest point on the chart. For instance, 100 is peak popularity, while 50 is half of it.

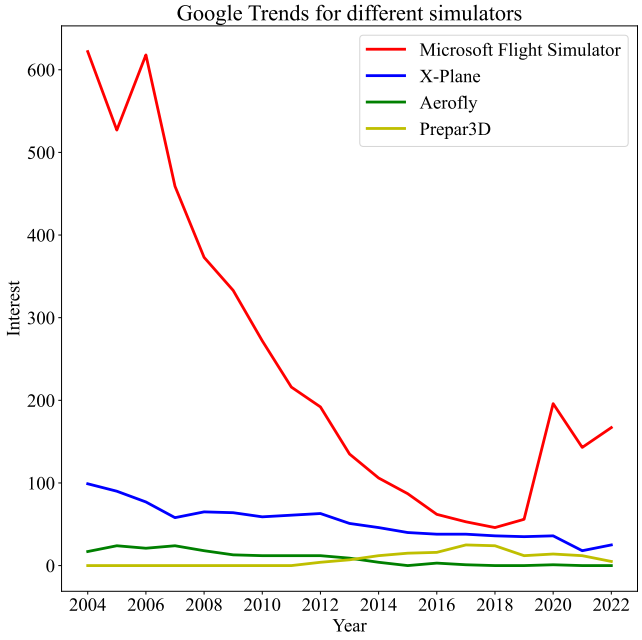

Fig. 4. Google Trends for the games (debatable classification) "Microsoft Flight Simulator", "X-Plane", "Aerofly" and "Prepar3D" reveals that the Microsoft product is ahead of the competition. The Interest axis represents the global search interest relative to the highest point on the chart. For instance, 100 is peak popularity, while 50 is half of it.

It is clear that FS2020 is able to cater not only to advanced simulation users, but also to novice and gamers in general. This is done by adding missions and competitive elements to the software, attempting to turn it into a game [10], while maintaining respectable physics and the characteristics expected from a simulator. Also, it is the only available simulator for console, which further increases its reach for the casual users.

This is very important, because, although not everyone that tries a simulator will develop a passion for aviation, some will, which can lead to potential new pilots. This is the public that needs the tools to have a more simulator-like experience, while keeping down the costs. Generally, these tools fall under the category of Human Interface Devices (HID) [11]. Some of them are: joysticks, rudder pedals, power quadrants, radio stacks, eye trackers, and virtual reality goggles.

Unfortunately, FS2020 cost and hardware requirements can be prohibitive to many users, therefore, this work takes a step back and develops for FS9 and FSX. FS9 is now considered abandonware and is available in digital museums for preservation purposes, while FSX is still sold to this day, with over 20000 reviews on Steam[5]. This also guarantees a work that is easily reproducible. Furthermore, it can be modified to work with other simulators, the only requirement is that there must be a way to read from/write to the simulator and perform serial communication with the device.

III. Related Works

Although there are many related works, this section focuses on some more recent works that study varied topics surrounding HID development for flight simulators.

In 2015, Aslandere, Dreyer and Pankratz [12] developed a way for a person to interact with an aircraft's cockpit using VR goggles and hand motion controllers. Essentially, the hands are tracked and represented in the virtual world as a hand 3D object, then this object has its collision box checked against the collision boxes of switches, knobs, and other cockpit items. If an adequate collision happened, the respective item is toggled/changed. Even though VR is still a very expensive item in many countries, it already massively reduces the cost to obtain realism in a simulator.

Still in the same year, Lee and Kwon [13] developed an Arduino PCB module for large commercial jet simulation. It basically does in a much larger scale and complexity what this work proposes: an intermediate hardware that enables the communication between the user and the simulator.

Moving forward 3 years, in 2018 Benedikovic, Hamernik and Brozek [14] presented Arduino as an interlink between arbitrary Input/Output (I/O) devices. This interlink is low-latency and fault-tolerant, which shows that the board is well suited for the job. They also argue that combining this fact with the very low cost of an Arduino, it may as well be the best solution for small and medium-sized projects, which can, of course, benefit hobbyists and even simulation enthusiasts.

[5]https://store.steampowered.com/app/314160

Also in 2018, Redei, Dascalu and Harris [15] built a framework (on top of another study by Redei and Dascalu [16]) for virtualizing joystick controls, where they tackle the problem of two pilots concurrently sending converging or diverging inputs to one single plane. This conflict can happen in real life, in planes with a pilot and a copilot. The authors evaluate different conflict resolution techniques and are able to provide a seamless experience for standard one joystick operation and weight the input of different joysticks, depending on the situation, to resolve conflicts. The project was also built using Arduino as the core system.

Going another 3 years into the future, Othman et al. [17] built a reasonable and affordable flight simulation device. It consists of a structure made of metal and wood containing a base and a bench, with 2 yokes (the aircraft's ailerons and elevator controller) mounted. Each yoke's movements are captured by two potentiometers that are connected to yet another Arduino, this way it is possible to control the aircraft's pitch and roll. To satisfy the most GA aircraft's control surfaces, they are missing the rudder (specially), flaps and possibly the speedbrakes controllers.

Finally, in 2022 Suarez [18] published his undergraduate Thesis titled HID interface program based in Arduino for connecting flying simulator controls to a PC. This work uses an Arduino to translate physical inputs into a Windows compatible HID device that is able to interact with flight simulators. It basically provides a blank canvas to build any sort of controller out of the core device, having the possibility to become a cost-effective way to build dedicated hardware for simulation.

As it is possible to notice, the use of Arduinos is very common due to 3 main reasons: they are cheap, responsive, and reliable. Also, there are multiple versions of Arduinos, and they are programmed very similarly, so it is possible to adapt or extend the code to a bigger Arduino board, with a more powerful microcontroller to deal with bigger projects. This work will be no different, as it also uses an Arduino Uno (Figure 5) to intermediate the communication between the human and the simulation running in a computer.

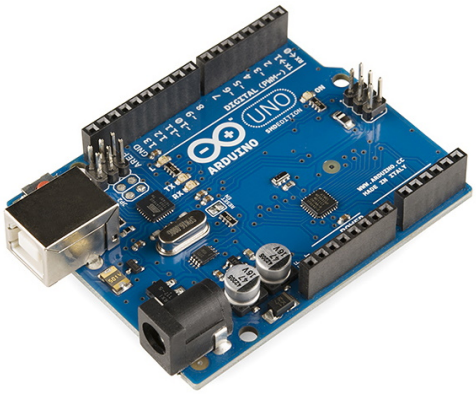

Fig. 5. Arduino Uno board, by SparkFun Electronics from Boulder, USA, CC BY 2.0 <https://creativecommons.org/licenses/by/2.0>, via Wikimedia Commons

IV. Radio Stack

Fig. 6. Microsoft Flight Simulator 2004's generic radio stack. It is modeled after Bendix/King Silver avionics system.

The device proposed in this work controls most of the FS9's Cessna C172SP Skyhawk's, Cessna C182S Skylane's, Cessna C208 Caravan Amphibian's, Cessna C208B Grand Caravan's and Mooney Bravo's radio stack, which is modeled after Bendix/King Silver avionics system. Figure 6 shows a screenshot of the in-game stack.

Going from top to bottom of the radio stack, ignoring the Morse code activators (first row of buttons), we have the following modules:

- Communication 1 (COMM1) and Navigation 1 (NAV1) frequency selectors. COMM 1 chooses which frequency to hear from/communicate to using the radio. NAV1 picks which Very High Frequency Omnidirectional Range (VOR) station the Course Deviation Indicator (CDI) and Instrument Landing System (ILS) glide slope (if supported) gauge will be tracking.
- COMM2 and NAV2 frequency selectors. Same functionality as COMM1 and NAV1, but for a different radio channel and CDI/ILS gauge.
- Automatic Direction Finder (ADF) frequency selector. Chooses which Non-Directional Beacon (NDB) station to track with the ADF gauge.
- Distance Measuring Equipment (DME). If NAV1 or NAV2 is pointing to a Doppler VOR (D-VOR) station instead of a simple VOR, this instrument shows the distance (in Nautical Miles) the aircraft is to the station and the speed (in Knots) in which it is moving. A switch chooses which NAV to pick up.
- Transponder (XPDR). Sets the Squawk Code for identification and flight following.
- Autopilot. Controls the aircraft's autopilot system. This model supports the basics, such as heading, altitude and vertical speed hold, as well as navigation and approach following. It is worth noting that there are two extra controls of the autopilot system that are not in this stack: the heading selector and GPS/NAV switch, which are positioned in different parts of the cockpit.

V. Hardware Assembly

A. Parts

The complete device is composed of 3 essential and 1 optional part:

Arduino Uno board (Figure 5)

It is the brain of the system and has as its core function to intermediate the communication between the human and the simulator. It reads the physical inputs, then sends them to the simulator, which will change the state of the world. Not only that, but also reads the simulation state and presents the information to the user.

LCD Keypad Shield (Figure 7)

This shield contains 6 push-buttons, of which 5 are customizable and 1 is a reset button, and also a 16 by 2 Liquid-Crystal Display (LCD). Although the shield is highly convenient for this device, it is not needed and the components can be individually plugged in the board, with minor changes to the code.

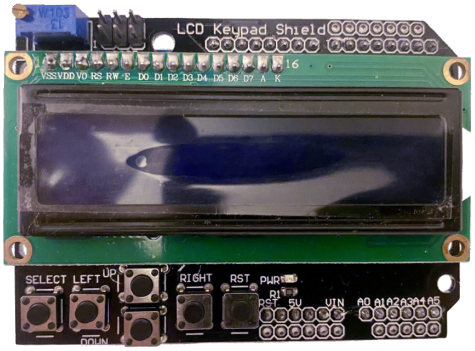

Fig. 7. LCD Keypad Shield for Arduino. Contains 6 buttons and an LCD.

USB type B to USB type A cable (Figure 8)

The Universal Serial Bus (USB) cable lets the Arduino connect to the computer in Serial mode, which enables the

bidirectional data transmission between the devices. This cable can be swapped by any other alternative that permits the same kind of communication.

Fig. 8. USB type B to USB type A cable for the communication between the Arduino board and the computer.

3D printed case for LCD Keypad (Figure 9)

This whole item is optional, as it can not only be replaced, but also completely ignored. The case's main functionality is to be a rudimentary protection against dust and liquids. It makes the device as more visually appealing, too.

Fig. 9. 3D printed case to accommodate an Arduino with a LCD Keypad shield connected

B. Assembly

The assembly itself is very simple: first plug the LCD Keypad on top of the Arduino Uno board, as show in Figure 10. Then, fix the ensemble inside the 3D printed case, just like depicted in Figure 11. Finally, close the case by screwing its top on its base, the same way as shown in Figure 12.

It is very important that the case has a hole for the USB cable to be plugged in the Arduino board. The power supply hole in the case used here is optional, since the computer's USB port should be able to supply enough power.

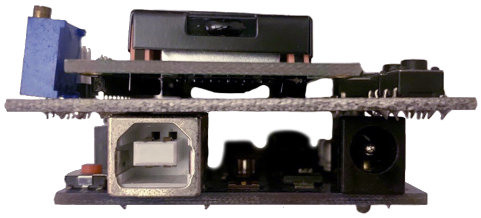

Fig. 10. LCD Keypad Shield plugged on top of Arduino Uno board

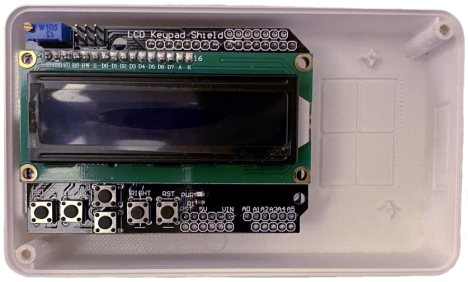

Fig. 11. Arduino and Shield fixed inside the 3D printed case

Fig. 12. 3D printed case closed

C. LCD Keypad connections

Although it is convenient to use the LCD Keypad shield, it can be bypassed by adequately connecting the equivalent components directly to the Arduino board. Figure 13 shows how to perform this connection. The complete schematic of the shield would also contain pass-through connectors between the board and the shield. These have been omitted in the figure, since they would not be useful in this particular case.

The buttons array (excluding the reset) was built in a way to occupy a single analog port, instead of 5 digital ports. Although this small port optimization is optional, it is important to keep in mind that changing to the digital ports alternative leads to necessary small changes in the code.

Another point to be made is that the actual LCD used in the particular shield of this work is the TC1602, not HD44780, which includes extra LED ports that are taken care of by the shield itself. These are not essential for the core interpretation of the circuit and that is why, for simplification purposes, the figure shows an HD44780.

Finally, the microcontroller pins used by the LCD are not the typical ones used in many Arduino examples (12, 11, 5,

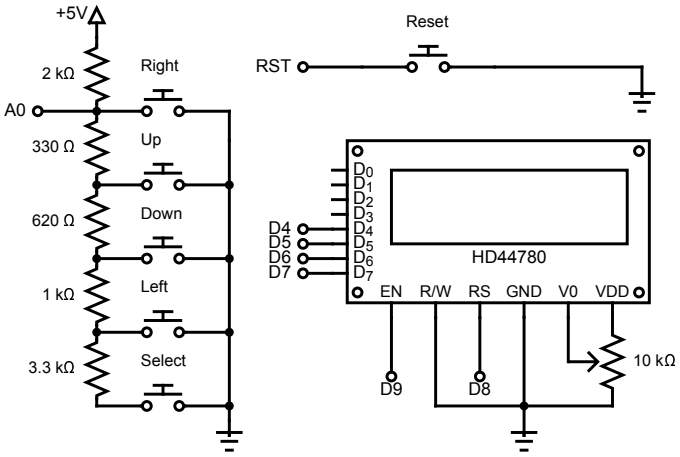

Fig. 13. LCD Keypad Shield's most important components (buttons and LCD screen) equivalent circuit connected to the ATmega328.

4, 3, 2), but rather (8, 9, 4, 5, 6, 7). Changing this, of course, means extra changes in the code.

VI. Software Architecture

To facilitate the communication between the Arduino and the Flight Simulator, a third party program called FSUIPC[6], by Pete and John Dowson [19], is used. The version necessary for FS9 is no longer supported, therefore it is free of charge. This software is able to perform serial communication and can read from/write to the correct memory addresses from the flight simulator to access/modify most of the information available.

This means that the core architecture is composed by 3 main parts: the Arduino program, FSUIPC and FS9, where the first two communicate by serial and the last two, by sharing information stored inside the Random Access Memory (RAM). Figure 14 illustrates the described system.

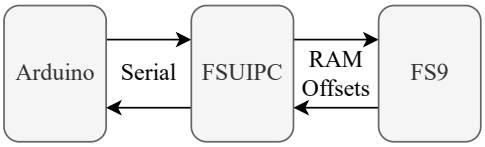

Fig. 14. Diagram representing how the communication happens between the Arduino and the Flight Simulator. FSUIPC is a software by Pete and John Dowson [19] used to facilitate the communication.

The Arduino is responsible to send the pushed buttons and current operation mode to FSUIPC and, in return, receives the values that should be displayed on-screen, given the operation mode and actual state of the simulation universe. A button press causes can cause the state to change, which immediately reflects in the information being received from FSUIPC, giving the impression that the values are being locally processed by the Arduino.

For instance, supposing the system is in mode 0 (would be COMM1 frequency MHz edit enabled), value change button presses in the Arduino lead to information being sent to FSUIPC that indicates the MHz of COMM1 should increase or decrease by 1. FSUIPC, then, send the corresponding command to FS9, which changes the state of the simulation. FSUIPC reads the changed state and, since the operation mode is 0, sends the COMM1 current frequency back to Arduino, which displays it in the LCD.

A FSUIPC program is a LUA script [20]. To inform FSUIPC which control should be manipulated inside the simulator, the programmer must inform the control number. Table I list all the control numbers used in this work to manipulate COMM1. A complete list of supported controls can be found in FSUIPC documentation.

TABLE I
Control numbers to manipulate COMM1 radio frequencies

Number	Control description
65636	COMM1 Mhz range Down
65637	COMM1 Mhz range Up
65638	COMM1 Khz range Down
65639	COMM1 Khz range Up
66372	COMM1 Standby Frequency Swap

To read from/write to FS9 memory addresses, FSUIPC needs to be informed of the memory offset to be used. The software provides functions to read various types of data, but the values obtained from the memory addresses may not immediately resemble what is being shown in the simulator. This leads to some calculations having to be made to send Arduino a properly formatted value to be shown. For instance, Equation 1 shows how to properly format the autopilot altitude hold value, so that it is the same in the simulator and in the LCD (ignoring the zeros to the left that must be occasionally added).

$$f(x) = 100 \left\lfloor \frac{x}{1997600} + \frac{1}{2} \right\rfloor \quad (1)$$

Table II shows the offsets used to read various autopilot values. A complete list of all the possible offsets can be obtained in the software's documentation.

The proposed device uses a total of 50 control numbers and 22 memory offsets. Since FSUIPC provides many more, it is

TABLE II
Memory offset for various autopilot values

Offset (Hex)	Value description
0x07BC	Master state
0x07C4	Nav hold state
0x07C8	Heading hold state
0x07CC	Heading value
0x07D0	Altitude hold state
0x07D4	Altitude value
0x07EC	Vertical speed hold state*
0x07F2	Vertical speed value
0x07FC	Approach hold state
0x132C	Nav mode (NAV or GPS)

*unavailable for FS9, therefore unused here.

[6]http://www.fsuipc.com/

possible to extend this device to control many other parts of the plane and read many other values. An example of possible extension would be to include visualizers for the various flight instruments present in an aircraft, such as the six-pack [21].

VII. Device usage

Here it will be presented the device's interface and how to use it. As already mentioned, the device has 5 programmable buttons, 1 reset button and a 16 by 2 LCD. This means that all the interface must take this very limited set of resources into account. A very good upgrade to the current device, without much additional cost, would be to use a 20 by 4 LCD and more buttons.

To work with the limitations, the device remaps buttons differently (except for the left and right buttons), depending on the operation mode. The way the operation modes work is the following: for each item in the radio stack, an exclusive interface screen was developed. For each screen, there can be multiple operation modes, for instance, the ADF screen has 4 operation modes, one for each digit of the frequency that can be changed. For each one of the ADF's screen operation mode, the up and down button have a unique functionality, which would be to change the corresponding digit.

Some screens behave the same, in respect to the intuitiveness of what each button will do, but some work in a quite different manner. A summary of how to operate the device is now presented. All LCD screens show simulation data for a recently spawned Cessna C172SP Skyhawk in Seattle-Tacoma airport, in FS9, without any changes or movement.

A. COMMs and NAVS

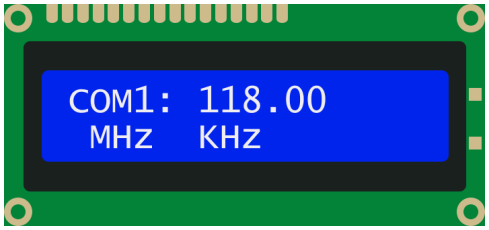

Fig. 15. Graphical representation of the LCD screen when in COMM1 operation modes

Everything here applies to COM1, COM2, NAV1 and NAV2 screens, since they are identically formatted. The left and right button change operation mode, where left subtracts 1 and right adds 1 to the operation mode value. Up and down, increase and decrease, respectively, from MHz or KHz range (depending on the operation mode). Select (the button on the extreme left) swaps the current frequency by the standby one.

To choose between MHz and KHz, the user must change the operation mode. So if the operation mode is 0, up and down change the MHz range. If the right button is pressed, the operation mode is changed to 1 and now the up and down buttons change the KHz range. Pressing right again changes the mode to 2, which would change the screen to COM2 MHz mode, and so on. To aid the user, the LCD cursor blinks before the "MHz" or "KHz" text, depending on the mode. Figure 15 shows the COM1 screen.

B. ADF and XPDR

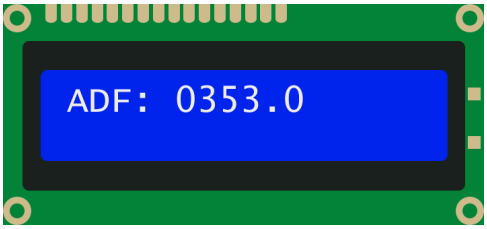

Fig. 16. Graphical representation of the LCD screen when in ADF operation modes

The ADF and XPDR screens behave the same. As previously mentioned, the left and right buttons are not remapped and always change mode (this holds true for all the following cases, therefore will not be further mentioned). In this case, changing the mode, means changing a different digit of the ADF/XPDR systems. Therefore, if the mode is 8, up and down adds and subtracts, respectively, 1 to/from the first ADF digit, whereas 9 deals with the second one.

The select button in this screen jumps to the next screen, independent of the digit the user is currently on. On the ADF screen, this means that a select button press always change the operation mode to 12 (DME) and, on XPDR screen, to 17 (Autopilot altitude). This is basically a shortcut, so the user do not have to go through all the digits, if they do not want to. To help the user know which ADF/XPDR digit is being edited, a blinking cursor appears below the current digit. Figure 16 shows the ADF screen.

C. DME

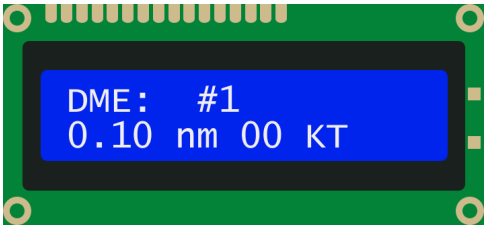

Fig. 17. Graphical representation of the LCD screen when in DME operation mode

The DME screen is very simple (as are all the following ones). It has a single operation mode: up, down and select buttons swap the DME being monitored by the radio stack. If it is monitoring DME number 1, #1 will show, alongside the distance in nm and the relative speed in KT. Pressing an action button (up, down or select) will change the DME being monitored to #2, with the respective information also being shown for this one.

In this mode, there is a cursor blinking before the number of the DME being monitored, as a reference for the user to know what will be changed. Figure 17 depicts the screen in DME operation mode.

D. Autopilot Altitude and Heading Hold

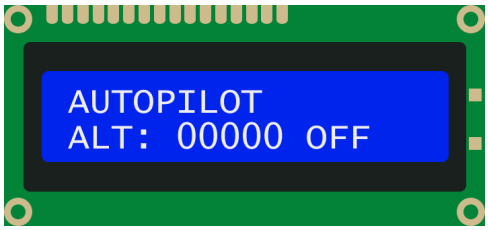

Fig. 18. Graphical representation of the LCD screen when in Autopilot Altitude operation mode

These two modes work the exact same, where there is one operation mode for each. Considering the altitude screen, up and down buttons increase and decrease, respectively, the hold altitude of the autopilot by 100 and the select button activate or deactivate the hold, changing the text in the screen from OFF to ON and vice-versa.

In the case of the heading hold, up and down buttons change the heading by 1 degree, and select activates or deactivates the hold. It is noteworthy that the heading adjust knob is not in the radio stack, but rather mounted together with the Heading Indicator. Since it directly affects the autopilot system, the proposed device still lets the user manipulate this dial directly.

Both screens have a blinking cursor before the numbers, but since they work in a single operation mode, there can be no cursor blinking before the HLD or OFF text. Figure 18 shows the autopilot altitude screen.

E. Autopilot Vertical Speed

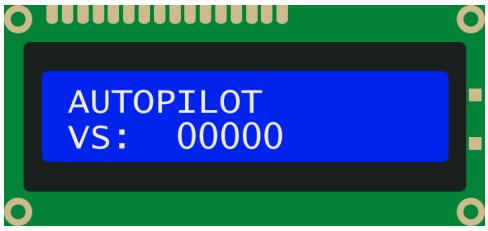

Fig. 19. Graphical representation of the LCD screen when in Autopilot Vertical Speed operation mode

This mode works identically to Autopilot Altitude and Heading hold modes. The exception is that the select button does nothing, since FS9 does not support Vertical Speed hold toggle, as it is automatically activated, depending on the current aircraft altitude and current autopilot target altitude, if altitude hold is enabled. This can be programmed to work properly in FSX and Prepar3D.

F. Autopilot Navigation Mode

In this operation mode, any action button will change the navigation mode from NAV to GPS or vice-versa. The navigation mode decides if the autopilot should (if Navigation Hold is enabled) follow a configured VOR station, typically the one set in NAV1, or the programmed GPS route. The button to switch the modes is not in the radio stack, but rather elsewhere in the plane. Since it still directly changes the autopilot behavior, the proposed device has the switch functionality built-in.

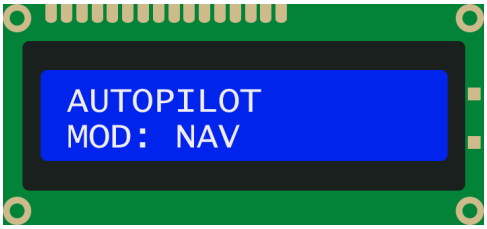

Fig. 20. Graphical representation of the LCD screen when in Autopilot Navigation Mode operation mode

A blinking cursor should appear to the left of the NAV or GPS text, indicating, to the user, what is being edited in this operation mode. Figure 20 reveals the Navigation Mode screen in the LCD.

G. Autopilot Master, NAV hold and APR hold

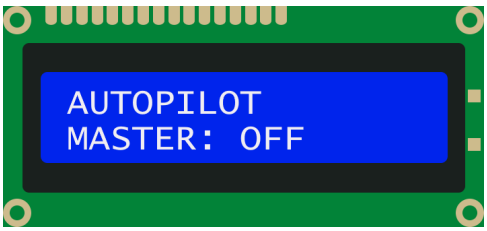

Fig. 21. Graphical representation of the LCD screen when in Autopilot Master operation mode

Autopilot Master, Navigation hold and Approach hold screens work in a very simple manner: any action button will turn on or off the screen's corresponding functionality. When the navigation hold is enabled, the autopilot will follow either NAV1 or GPS, depending on the navigation mode. When the approach hold is enabled, the autopilot will follow the set ILS direction and glideslope (typically set on NAV1). Finally, the autopilot master control turns on or off the whole autopilot system.

All of these screens have a blinking cursor before the ON/HLD or OFF text, to demonstrate that action buttons will activate or deactivate the functionality. Figure 21 shows the autopilot master screen, which is the last operation mode, so it can be easily accessed from operation mode 0, by pressing left (cycles back to 23, which is the master mode number).

H. Operation Modes table

Since there are many screens, operation modes and action buttons functionalities, Table III contains a thorough description of what each action button does for every possible operation mode.

TABLE III
ACTION BUTTONS FUNCTIONALITIES FOR EACH POSSIBLE OPERATION MODE

OP Mode	Buttons		
	Select	Up	Down
00	Swap current and standby frequencies	+ COMM 1 Mhz	- COMM 1 MHz
01		+ COMM 1 KHz	- COMM 1 KHz
02		+ COMM 2 Mhz	- COMM 2 MHz
03		+ COMM 2 KHz	- COMM 2 KHz
04		+ NAV 1 Mhz	- NAV 1 MHz
05		+ NAV 1 KHz	- NAV 1 KHz
06		+ NAV 2 Mhz	- NAV 2 MHz
07		+ NAV 2 KHz	- NAV 2 KHz
08	Skip ADF editing	+ ADF 100ths	- ADF 100ths
09		+ ADF 10ths	- ADF 10ths
10		+ ADF units	- ADF units
11		+ ADF decimals	- ADF decimals
12	DME Swap		
13	Skip XPDR editing	+ XPDR 1000ths	- XPDR 1000ths
14		+ XPDR 100ths	- XPDR 100ths
15		+ XPDR 10ths	- XPDR 10ths
16		+ XPDR units	- XPDR units
17	Toggle ALT hold	+ target altitude	- target altitude
18		+ vertical speed	- vertical speed
19	Toggle HDG hold	+ heading angle	- heading angle
20	Toggle navigation hold		
21	Toggle approach hold		
22	Toggle navigation mode (NAV or GPS)		
23	Toggle autopilot master		

VIII. In-simulator GUI

After installing FSUIPC, copying **fsradio.lua** to the **Modules** folder and configuring a way to start the device's LUA script, in FSUIPC settings (Figure 22), start it. The first time, a window should appear indicating that it could not open Arduino's COM port (Figure 23. After that, a screen asking for the COM port to read the Arduino from will appear (Figure 24). Inform the port and if everything is correct, a new success message will show up (Figure 25), otherwise, a new attempt to start the system will have to be made.

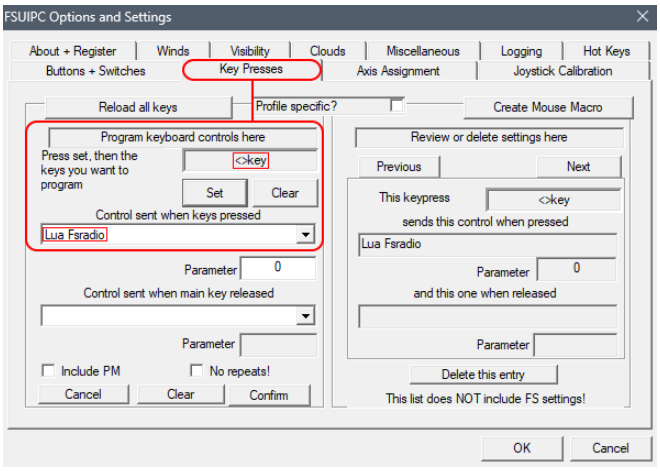

Fig. 22. FSUIPC configuration screen. To use the proposed device, it is necessary to initialize its LUA script, this can be done by binding it to a key, as shown here. See FSUIPC's manual for more details on how to start a script.

Fig. 23. Window showing an error, which tells the user it could not open Arduino's COM port.

Fig. 24. Window asking for the COM port where FSUIPC can find the Arduino

Fig. 25. Window telling the user that the Arduino has been found and is, now, communicating with FSUIPC.

This Graphical User Interface (GUI) is made using FSUIPC's display API, which tells FS9 to show a window with contents to the user. It is a very basic GUI, but considering the basic nature of the proposed device's functionalities and that most of the time the device will be using its own screen to communicate with the user, it is enough.

IX. Tests

The device was thoroughly tested, in relation to its capabilities, and various pieces of information about its strong and weak points were collected. Since it is a peripheral intended to be operated by humans, the evaluation of its performance can be objective in some cases, but subjective in others. The user experience may vary.

A. Computational environment

- **CPU:** Intel(R) Core(TM) i5-10210U
- **RAM:** 2x4GB LPDDR3
- **GPU:** Intel(R) UHD Graphics 620
- **OS:** Windows 11 Home Single Language 22H2
- **Simulator:** Flight Simulator 2004: A Century Of Flight

B. Objective results

The device works as intended, as it successfully performs all the operations it was designed to be able to, without errors, if its limitations are respected. It was observed that

it can fail to register very fast button presses, due to the very primitive debounce-avoidance system in place, which can be much improved in future versions.

Although failing to register a button press is not ideal, it is not catastrophic, since the user can see if the button press reflected in the simulator, by looking at the device's and/or simulator's screens.

A video showing the testing of many of the device's features can be found in the work's GitHub repository, in the README.md file or inside the main page of the repository: https://github.com/fvcalderan/fsradio.

The main positive points about using the device are its price, compactness, and extensibility. The price is how much an Arduino plus the components to build the circuit represented in Figure 13 cost, which would be below US$50,00, in the United States, as of January 2023. It is a small box capable of controlling the whole radio stack and can be programmed to deal with even more parts of the aircraft, without any change in size,

C. Subjective results

Although a 16 by 2 LCD and 5 programmable buttons were enough to be able to operate all the proposed functionalities, it is very limiting in some occasions, specially when speed is needed.

Generally, in civil aviation, there are not many cases of fast-paced sequences of actions a pilot must take during a cruise, so surfing through the device's screens and adjusting the values is not usually a problem. Unfortunately, taking-off and landing (specially when an emergency or route alterations are in place) can be very demanding, which means the device can hinder the pilot's performance.

For sure, very welcome upgrades to the device include a bigger LCD screen (at least 20 by 4) and more input methods, mainly buttons, but also potentiometers (they would make much easier to reach the wanted values). The bigger screen could fit more information at once, making it so that fewer button presses are needed to find the correct information.

X. Conclusion

Flight simulation has proven to be an essential tool for military and civil pilots training. Today, home simulators are a great way to make aviation accessible to more people, increasing the amount of potential pilots. The main problem is that the hobby is not always accessible in terms of hardware and even software.

This work proposes a very cost-effective device to control the radio stack of various GA aircraft in FS9 and FSX (with some tweaks). It only consists of an Arduino, some buttons and an LCD. The operation is very simple, and the program is very easily extensible, making it a good start point for bigger projects or upgrades. The source code for the project can be found at https://github.com/fvcalderan/fsradio.

Tests show that it is a capable device and that it fulfills its purposes, but it certainly has limitations. The small amount of buttons and small LCD lead to a not ideal experience when fast radio manipulation is needed, like when taking off or landing (specially in emergencies).

Possible next steps with this device include making it faster to use, by adding buttons, potentiometers and a bigger LCD. It is also possible to port the whole device to more modern simulators, such as X-Plane 11 (or the recently launched X-Plane 12) and Microsoft Flight Simulator 2020.

References

[1] J. De Angelo, "The link flight trainer," *ASME*, jun 2000.

[2] L. Ross, P. Slotten, and L. Yeazel, "Pilot's evaluation of the usefulness of full mission IFR simulator flights for general aviation pilot training," *The Journal of Aviation/Aerospace Education and Research*, 1990. [Online]. Available: https://doi.org/10.15394%2Fjaaer.1990.1024

[3] H. Magazines, "Airline pilots fly anywhere in world - without leaving the ground," *Popular Mechanics*, vol. 102, no. 3, p. 87, 1954.

[4] F. A. Simulations, "Flight simulator technology through the years," *Fly Away Simulations*, 2010.

[5] D. J. Allerton, "The impact of flight simulation in aerospace," *The Aeronautical Journal*, vol. 114, no. 1162, pp. 747–756, dec 2010. [Online]. Available: https://doi.org/10.1017%2Fs0001924000004231

[6] A. Fafard, "Analysis: Civil simulator fleet nears 1,300 mark," *Flight Global*, 2018.

[7] M. Morrison, "Analysis: Civil simulator manufacturer strategies compared," *Flight Global*, 2018.

[8] B. Cameron, H. Rajaee, B. Jung, and R. Langlois, "Development and implementation of cost-effective flight simulator technologies," in *Proceedings of the 3rd International Conference on Control, Dynamic Systems, and Robotics (CDSR'16). Canada, Ottawa*, no. 126, 2016, pp. 1–8.

[9] W. Sweet, "The glass cockpit [flight deck automation]," *IEEE Spectrum*, vol. 32, no. 9, pp. 30–38, 1995.

[10] P. Crogan, "Gametime: History, narrative, and temporality in combat flight simulator 2," in *The video game theory reader*. Routledge, 2013, pp. 297–324.

[11] *Device Class Definition for Human Interface Devices (HID): Firmware Specification – Final 1/30/97*. USB Implementer's Forum, 1997. [Online]. Available: https://books.google.com.br/books?id=bQZoHQAACAAJ

[12] T. Aslandere, D. Dreyer, and F. Pankratz, "Virtual hand-button interaction in a generic virtual reality flight simulator," in *2015 IEEE Aerospace Conference*, 2015, pp. 1–8.

[13] E. Lee and Y. Kwon, "Development of an expanded arduino interface board pcb module for large commercial jet simulator," *Int. J. Mech. Eng. Rob. Res*, vol. 4, no. 4, pp. 309–313, 2015.

[14] M. Benedikovic, D. Hamernik, and J. Brozek, "Arduino wrapper for game engine-based simulator output," in *International Conference on Applied Physics, System Science and Computers*. Springer, 2018, pp. 145–156.

[15] A. Redei, S. Dascalu, and F. C. Harris Jr, "A framework for virtualizing joystick controls in a flight simulator training environment," *INTERNATIONAL JOURNAL OF COMPUTERS AND THEIR APPLICATIONS*, p. 30, 2018.

[16] A. Redei and S. Dascalu, "A method for handling multi axis input for a motion based flight simulator," in *27th International Conference on Software and Data Engineering*, 2018.

[17] M. A. Othman, M. S. M. Hanif, N. A. N. K. Amilin, and M. L. A. Saing, "Development of flight simulator control trainer (fsct) for aviation education," *Multidisciplinary Applied Research and Innovation*, vol. 2, no. 3, pp. 095–101, 2021.

[18] P. Suárez García *et al.*, "Hid interface program based in arduino for connecting flying simulator controls to a pc," 2022.

[19] P. Dowson, "Fsuipc: Application interfacing module for microsoft flight simulator," 2012.

[20] R. Ierusalimschy, L. H. de Figueiredo, and W. C. Filho, "Lua—an extensible extension language," *Software: Practice and Experience*, vol. 26, no. 6, pp. 635–652, jun 1996. [Online]. Available: https://doi.org/10.1002%2F%28sici%291097-024x%28199606%2926%3A6%3C635%3A%3Aaid-spe26%3E3.0.co%3B2-p

[21] G. McKay, "Six pack – the primary flight instruments," mar 2010. [Online]. Available: http://www.learntofly.ca/six-pack-primary-flight-instruments/