

# Motorized Throttle Quadrant for Flight Simulation

---

By

Wendi Fu

Ziang Guo

Yuqi Xue

Final Report for ECE 445, Senior Design, Spring 2021

TA: Chaitanya Sindagi

05 May 2021

Project No. 45

## Abstract

This paper discusses the research, development, and evaluation of a design for motorized throttle quadrant peripheral aimed at consumer-grade home flight simulation. The final product provides functionalities that recreate the thrust-following levers found on multiple commercial airliner models (e.g., Boeing B series). The aim of the product is to provide realistic and consumer-friendly flight simulation experiences at prices similar to non-motorized equivalents existent on the market. Large portion of this paper is devoted to the development process behind the software add-on, the motor driver firmware, and the carrier circuit board. Additionally, this paper presents a new communication protocol designed specifically for communication between flight simulator and peripheral hardware. At last, possible improvements to the design and future enhancements are discussed.

## Glossary

**Auto Throttle (A/T)** is a part of the flight automation system that takes full control of the thrust demand of the aircraft once engaged. The pilot only need to specify a desired performance figure and the A/T system will automatically adjust engine outputs to achieve said performance.

**Control Column** is the device for controlling aircraft's pitch and roll placed in front of the pilots on most aircraft models.

**Flight Simulator** is a hardware device, a software program, or a combination of both that artificially re-creates aircraft flight and the environment in which it flies, for pilot training, design, or other purposes.

**FMC/CDU** is short for Flight Management Computer/Central Display Unit. It is the input device for flight information including route, aircraft performance figures, and performance limits. Must be correctly configured for flight automation to function.

**Main Control Panel (MCP)** is the control panel allowing the pilots to select operating modes of the flight automation system and command automated attitude changes.

**Master Caution** is the secondary warning for critical non-emergency events such as A/T disengage.

**Multi-Function Display** is the inboard displays capable of displaying multiple system status including system recall and engine status.

**Primary Flight Display (PFD)** is the primary flight instrument in a digitized cockpit, displaying crucial information including flight automation status and aircraft attitude.

**Recall** is the log-keeping functionality of the FMC capable of summarizing events occurring with the aircraft for improved crew awareness.

**Speedbrake** is mechanised devices on the top-surface of the wings that induces extra drag when deployed for deceleration. Deployment of the speedbrake is controlled either manually with a lever on the throttle quadrant or automatically when the aircraft lands.

**Throttle Quadrant** refers to the unit housing all essential thrust control inputs (buttons and levers). Often placed in between two pilots in the center pedestal for ease of access.

**Thrust Lever** is a lever-like input device with a defined travel range. The position of the thrust lever is mapped to engine thrust demands.

**TO/GA (Take Off/Go Around)** is the maximum rated thrust the engines can generate.



# Contents

1	Introduction . . . . .	1
1.1	Objective . . . . .	1
1.2	Background . . . . .	2
1.3	High-Level Requirements . . . . .	2
2	Design . . . . .	3
2.1	Host Add-On . . . . .	4
2.1.1	The Throttle Quadrant Thread (TQThread) . . . . .	4
2.1.2	The SimConnect Thread (SCThread) . . . . .	5
2.2	ASDF: A Host/Device Communication Protocol for Flight Simulation . . . . .	6
2.3	Electrical & Electronic Parts . . . . .	6
2.3.1	Microcontroller . . . . .	6
2.3.2	Motor Driver . . . . .	7
2.3.3	Stepper Motors . . . . .	7
2.3.4	Power Supply . . . . .	7
2.4	Microcontroller Firmware . . . . .	7
2.5	Circuits & PCB . . . . .	8
2.6	Mechanical Parts . . . . .	9
2.6.1	Levers and Buttons . . . . .	9
3	Design Verification . . . . .	10
3.1	Host Add-On . . . . .	10
3.2	ASDF Protocol . . . . .	10
3.3	Hardware . . . . .	11
3.3.1	Circuit and PCB . . . . .	11
3.3.2	Microcontroller Firmware . . . . .	12
3.3.3	Mechanical Parts . . . . .	12
4	Costs . . . . .	13
4.1	Parts . . . . .	13
4.2	Labor . . . . .	13
4.3	Profit Analysis . . . . .	14

5 Conclusion. . . . .	15
5.1 Accomplishments . . . . .	15
5.1.1 ASDF and Host Add-on . . . . .	15
5.1.2 Microcontroller Firmware . . . . .	15
5.1.3 Circuits . . . . .	15
5.2 Uncertainties . . . . .	16
5.2.1 Power Supply. . . . .	16
5.2.2 Ergonomics . . . . .	16
5.3 Ethical Considerations. . . . .	16
5.4 Future Work. . . . .	17
5.4.1 Motorized Speedbrake . . . . .	17
5.4.2 Circuit Integration . . . . .	17
5.4.3 Replica Levers . . . . .	17
Reference . . . . .	18
Appendix A ASDF Protocol Specification . . . . .	19
Appendix B PCB Schematic and Layout . . . . .	20
Appendix C Exterior Appearance of Final Prototype . . . . .	22
Appendix D Requirement and Verification Table . . . . .	23

# 1 Introduction

## 1.1 Objective

Modern airliners are equipped with multiple flight automation systems to alleviate the workload of pilots during long flights. One of these automations commonly fitted is the Auto Throttle (A/T) system. A/T is able to fully control the engine thrust demand of the airplane in order to achieve constant airspeed cruise and provide crucial safety functionalities. To improve the situational awareness of the pilots, Boeing, alongside many major airplane manufacturers, fitted their airplanes with motorized throttle quadrants. When A/T sends thrust settings to the airplane's engines, the thrust levers will move accordingly to the positions that reflect the real-time engine thrust outputs. Such system allows the pilots to regain control of the throttle at any moment with full awareness of the current thrust settings. This is a crucial control characteristic of these airplanes. Without this consistency between thrust lever positions and actual engine thrust outputs, when A/T is disengaged, the airplane's engines will immediately try to adjust to the current thrust lever positions, which may cause a sudden and unexpected power increase or decrease of the engines.

Current main-stream consumer-level flight simulator peripherals use spring tension to simulate the weight felt by the pilot when operating the control column. However, no effort is made to simulate the synchronous movement of the throttle quadrant with A/T commands. Home-based simulation systems often have the pilots to manually adjust the thrust levers to match the actual thrust setting inside the simulator software program, hence deteriorating the experience. Being able to recreate the motorized throttle will bring the realism of the flight simulation experience to a whole new level.

With this project we aim to design and create a motorized throttle quadrant for casual flight enthusiasts. The throttle quadrant can interface with mainstream simulation software (Microsoft FSX [1] or Lockheed Martin Prepar3D [2]) via universal protocol (e.g., USB 3.0) and synchronize thrust levers' positions with A/T commands. Figure 1 is an illustration of such process in action which vaguely represents the final product's main functionality.



Figure 1: Illustration of device operation. Multi-function display (MFD) of the aircraft (left). Real looking of the throttle quadrant (middle). Flight simulator throttle quadrant (right). The MFD, real throttle quadrant, and simulator throttle quadrant on the same row represent the same the engine thrust setting.

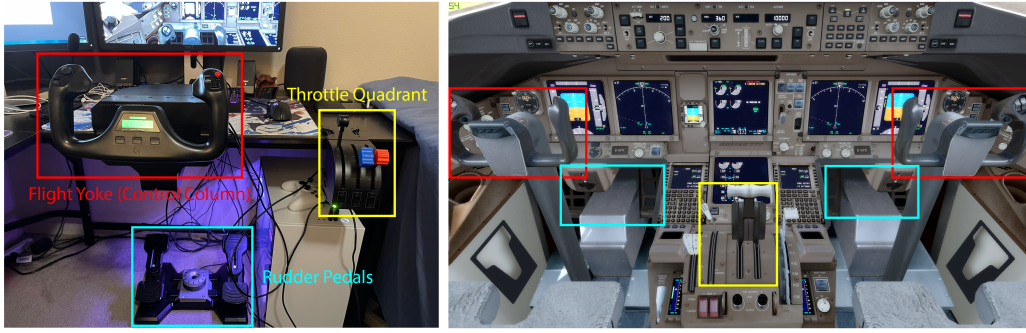


Figure 2: Comparison of a casual in-home flight simulator (left) and the real cockpit view in Prepar3Dv4 (right). Corresponding components are labelled by the same color (for example, red boxes mark the control columns in both pictures).

## 1.2 Background

Despite commercial full-scale flight deck simulators for professional pilot training purpose, a common choice for casual flight enthusiasts is Logitech Flight Yoke and Throttle Quadrant [3, 4], often purchased together with a Rudder Pedal System, as shown in Figure 2. While Logitech’s solution is affordable and popular, it does not have a motorized throttle quadrant, thus deteriorating the realism of flight simulation.

In the domain of driving simulators, wheel and pedal systems such as Logitech G920 [5] already support force feedback, and existing solutions are affordable to casual players. These simulators use a motorized wheel to simulate the actual behavior of a car’s wheel. Our project aims to build motorized throttle levers for flight simulators.

Flight Illusion has motorized flight yokes on sale [6], but their solutions cost more than \$1200 and are too expensive for casual players. No ready-to-use motorized throttle quadrant under \$1000 is available on the market as far as we have discovered.

## 1.3 High-Level Requirements

To better evaluate the outcome of the project, three major requirements must be met:

- The throttle quadrant must be recognized by the OS (i.e., be listed in Device Manager in Windows 10) and flight simulator software can detect user inputs from the throttle quadrant. We will be targeting Windows 10 and Prepar3Dv4. Supporting more OSes and simulators is possible but out of the scope of this work.
- If A/T is engaged and commands a thrust change, the throttle quadrant must reflect the movement (speed and direction of position change) of the thrust levers shown in the simulator’s virtual cockpit with a latency less than 100ms, which is an unnoticeable delay comparable to that of a wireless game console controller [7]. The thrust levers must stop within 2% of the commanded position. We assume that the simulator software is reliable for sending appropriate A/T commands so that the speed of the levers under A/T can be realized by the motors.
- When the A/T Disengage button is pressed, the throttle quadrant must disengage motor torque output in less than 50ms and allow for the pilot to regain control of the thrust levers. The motor must not output any torque to hinder the movement of the levers until the A/T is engaged again by the pilot.

## 2 Design

Figure 3 is an overall view of our design, which is divided into two parts that works in tandem to achieve the requirements listed in Section 1.3. The software part includes an add-on for selected flight simulation software (Section 2.1). The add-on gathers airplane status information from APIs provided by the flight simulation software and allows bidirectional communication between the flight simulator and the throttle quadrant device. We develop the ASDF protocol as the host/device communication protocol (Section 2.2). The hardware part includes a custom control board carrying a microcontroller and several stepper-motor driver chips (Section 2.3). The microcontroller has firmware mapping functions that translate between thrust lever position and throttle input. Linear potentiometers are used for detecting lever positions. The mechanical appearance of our product is also presented. (Section 2.6).

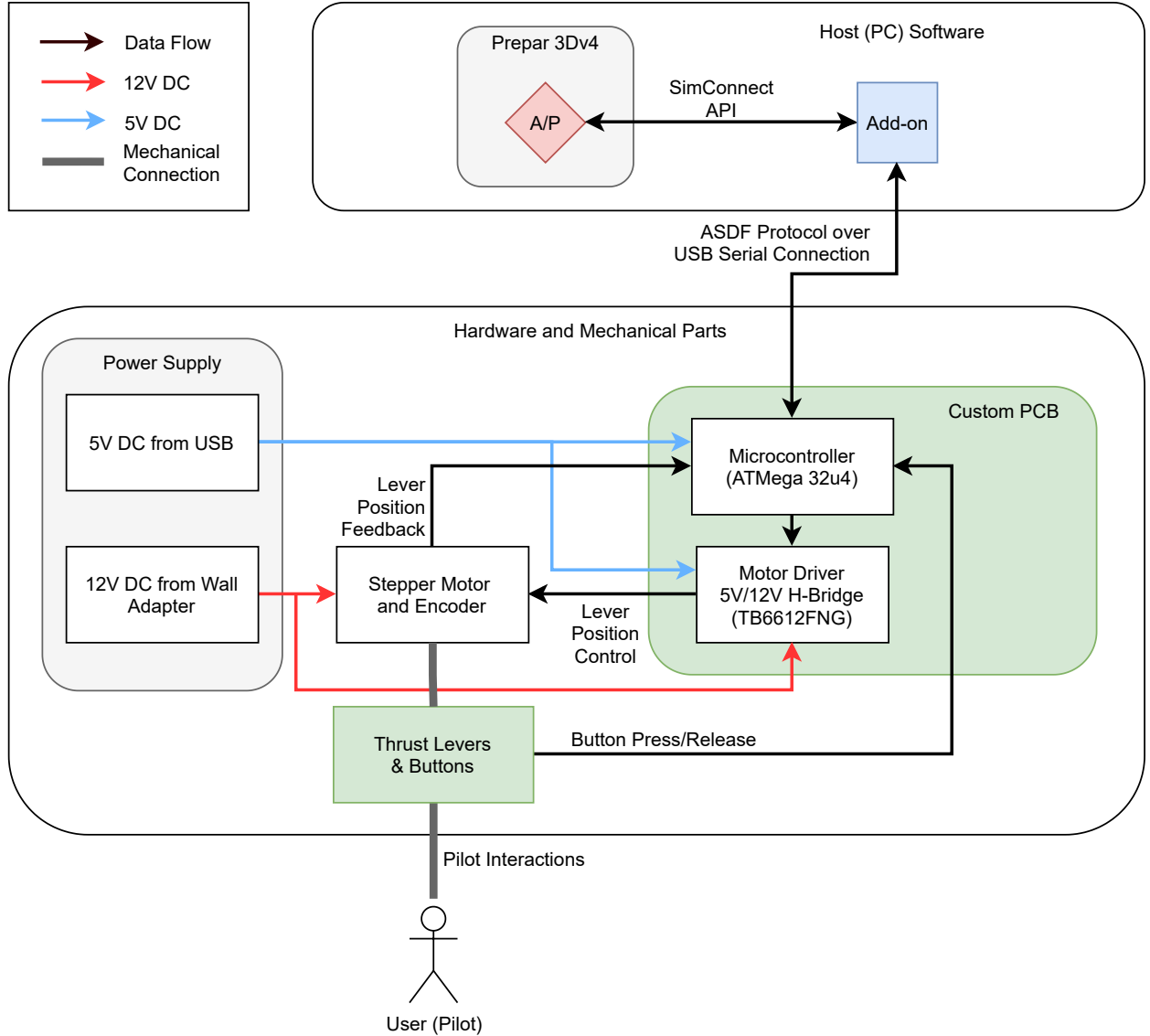


Figure 3: Block Diagram of the Design.

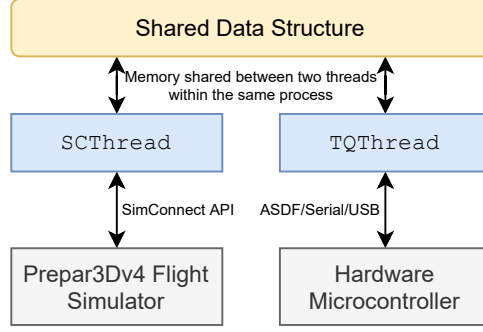


Figure 4: High-level Architecture of the Flight Simulator Add-On.

## 2.1 Host Add-On

The software subsystem running on the host computer consists of an add-on that is responsible for interacting with the flight simulator program and for commanding the throttle quadrant device. Specifically, the add-on needs to read user inputs from the device and transmit user inputs to the flight simulator if A/T is not engaged, or get simulation data from the simulator program and command the device accordingly if A/T is engaged.

The add-on is developed with Prepar3Dv4 SDK [8]. In particular, the SDK specifies the SimConnect API, which exposes a network-based communication interface for third-party developers to access flight data and simulation events from outside of the simulator program. Taking this convenience, we only need to develop our own program using SimConnect API and link to the pre-compiled library provided by the SDK. As a standalone program, the add-on can be easily installed by the user and configured to auto-start with the flight simulator.

The communication between the add-on and the hardware device is realized with the ASDF protocol, as discussed in Section 2.2. In particular, the ASDF protocol operates over a serial connection with USB channel as the physical media. The add-on uses the conventional Win32 Serial API to maintain serial connection to the device and implements the ASDF protocol based on the serial communication [9].

Since the add-on needs to address two challenges — interacting with the flight simulator and commanding the device — a natural design arises. In particular, the add-on is split into two separate threads, as shown in Figure 4. Multithreading avoids interference between the two functionalities and gives a more decoupled design for ease of debugging. The shared data structure between the two threads follows a single-producer single-consumer synchronization model to guarantee data consistency, and which thread gets the write permission is determined by the A/T Engage status.

### 2.1.1 The Throttle Quadrant Thread (TQThread)

For simplicity and for low latency, we use a dedicated **TQThread** that runs an infinite loop to poll throttle quadrant data from the hardware controller of the throttle quadrant device. In addition, it also initializes the device when the simulation begins and handles any error reported by the device.

The control flow of **TQThread** is shown in Figure 5. If A/T is not engaged, the user input data will be stored in a pre-allocated structure in memory and shared with **SCThread**, which is responsible for sending user inputs to Prepar3Dv4 using the SimConnect API. Conversely, if A/T is engaged, **SCThread** will write throttle lever positions into the shared structure and **TQThread** will send these data to the device via the ASDF protocol. In addition, **TQThread** always forwards button inputs to the flight simulator.

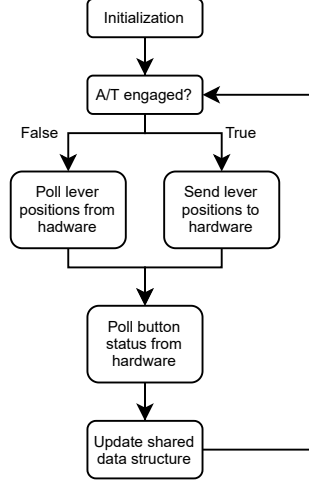


Figure 5: Control Flow Diagram of **TQThread** (error handling not shown).

### 2.1.2 The **SimConnect Thread** (**SCThread**)

The **SCThread** is composed according to the **SimConnect** API and the **Prepar3Dv4** SDK [8]. The API specifies an event-based communication with the flight simulator. When there is a change in **Prepar3Dv4** that is of our interest — for example, when the A/T changes the throttle level — **Prepar3Dv4** triggers a simulator event. This event is forwarded to **SCThread** via the **SimConnect** API.

For each simulator event, we create an event handler function. In particular, we are only interested in events that signify the change in throttle level. The event handler sets the corresponding fields in the shared data structure, and the **TQThread** can then forward the throttle level change to the hardware controller by sending corresponding **ASDF** packets. In practice, we are only interested in simulator events that signify A/T engage/disengage, throttle level change, or speed brake level change, and **SCThread** is configured during initialization such that only events of interest will be sent to the add-on.

When the **SCThread** receives an event from the simulator, it proceeds by calling

```
SimConnect_RequestDataOnSimObject()
```

to get the data (A/T status, throttle level, etc.) from the simulator, and then sets the corresponding field in the shared data structure so that **TQThread** can send **ASDF** commands to control the hardware throttle quadrant accordingly. Conversely, **SCThread** calls

```
SimConnect_SetDataOnSimObject()
```

to set status (A/T status, throttle level, etc.) for the airplane under simulation. Since this API call will pause the simulation until it returns to the **SCThread**, we cannot directly use an infinite loop and let the simulator “poll” data from the add-on, or the “polling” will cause significant simulation lag and degrade flight experience. Instead, in the infinite loop, we add a condition check to guarantee that we only call the set data API when the data are actually changed. The control flow of **SCThread** is shown in Figure 6.

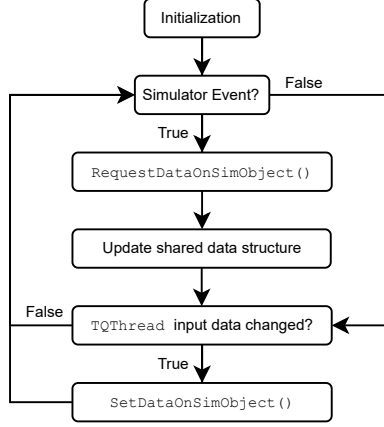


Figure 6: Control Flow Diagram of SCTesthread.

## 2.2 ASDF: A Host/Device Communication Protocol for Flight Simulation

The host program and the device microcontroller will communicate over a USB Serial connection, which provides a streaming interface for bidirectional reads and writes. We use existing USB Serial libraries for the Windows OS and the ATmega chip to implement our communication protocol.

We present the packet-based communication protocol, the Aviation Simulator Data Format (ASDF) Protocol, for the host to poll data from the thrust quadrant and send commands to set thrust position. The packet sent by the host consists of a command opcode and optionally several bytes of data. The opcode is 1 byte with the most significant bit set to 1. Each data byte will have the most significant bit set to 0 to be distinguished from opcode bytes. This scheme limits the number of opcodes and the maximum value of one data byte to  $2^7 - 1 = 127$ , which is more than enough for our purposes. The details of each command are specified in Table 2.

The response packet sent by the thrust quadrant microcontroller contains one or more bytes depending on the command received from the host. The response formats are listed in Table 3.

## 2.3 Electrical & Electronic Parts

The hardware subsystem consists of a microcontroller, three stepper motor driver chips, three stepper motors, the user interfaces (levers and buttons), and three potentiometers. The E&E subsystem makes up most of the functionalities of the throttle quadrant. The levers move at a constant rate smoothly whenever A/T commands a thrust adjustment. The levers will stop exactly and firmly at the designated position without slipping or drifting when they arrive at the designated position. The motors actively apply resisting torque to external disturbances when they are set in position until the A/T disengage signal is issued.

### 2.3.1 Microcontroller

The ATmega 32u4 microcontroller is the processing center on the device side. This particular model of microcontroller is selected due to its convenient integrated USB controller and support for the Arduino IDE environment. By flashing the onboard ROM with Arduino’s open-sourced bootloader, we are able to convert the 32u4 into an Arduino Leonardo equivalent and have access to multiple design tools exclusive to the platform.

Connected to the host PC via USB 3.0 and powered by USB, the microcontroller interfaces with the host



add-on to receive commands, data, and to report device status. One functionality of the microcontroller is to encode and decode necessary information in compliance with the ASDF protocol, as discussed in Section 2.2. The microcontroller firmware is also responsible for collecting analog values from lever position potentiometers, read button status, and issuing PWM control signals to the motor drivers. More information on the firmware running on the microcontroller can be found in Section 2.4.

### 2.3.2 Motor Driver

The 3 TB6612FNG dual H-bridge motor drivers used provide the necessary circuits to run the bipolar stepper motors in both forward and reverse direction. The motor driver takes three inputs: Ain, Bin, and PWM. Notice that the TB6612FNG consists of two H-bridges per chip, but our design requires both of them to control the same motor. Ain and Bin regulates the operational modes of the motor driver and can be used for special functionalities such as fast stop or halt. The PWM signal comes from the microcontroller's digital output pins.

### 2.3.3 Stepper Motors

Three NEMA 17 stepper motors are used to physically move the levers. Each stepper motor will be responsible for only one lever. The stepper motors are powered externally by a power adapter and controlled by the stepper motor driver. Each stepper motor is directly connected to a single-turn linear potentiometer to determine the location of the lever. The stepper motors' wings remain energized by the HALT function to resist external disturbances such as vibration.

### 2.3.4 Power Supply

The power supply is the only OEM block component in the E&E subsystem. Purchased from external sources, the power supply provides a steady 12 V potential to the motors' powerline, and possesses the capability of delivering a current of >5 A to power the motors at maximum torque.

## 2.4 Microcontroller Firmware

The firmware on the ATmega 32u4 MCU serves as an interface between the host device and the throttle quadrant. The microcontroller firmware receives ASDF commands from the serial connection and performs the requested operation. Analog readouts from the potentiometers and buttons are mapped to their corresponding ranges and then packaged into 8-bit data packets to be sent back to the host via ASDF. The microcontroller is also responsible for monitoring the status of the hardware system and reports device error or spontaneously reset should catastrophic errors occur.

$$\text{position} = (x - \text{in}_{\min}) \times \frac{\text{out}_{\max} - \text{out}_{\min}}{\text{in}_{\max} - \text{in}_{\min}} + \text{out}_{\min} \quad (1)$$

A mapping function is built into the firmware to convert the variable analog input range to the various data types required by the simulator and to stepper motor movement instructions for the H-bridge controllers. The 32u4 MCU is capable of 10-bit analog input using the `analogRead()` function provided by the Arduino standard library. The 10-bit data must be mapped on a 1:1 basis to narrower ranges required for the simulator to understand. For example, the lever inputs accepted by Prepar3Dv4 only ranges from 0 to 127. The stepper motor control function only accepts integers between 0 and 50. More discussion on the choice of the mapping function can be found in: Equation (1).

The microcontroller also facilitates the motor control functions. The motor control function takes integer inputs between 0 and 50 that indicates the number of steps the motor must rotate to position the levers in

```

int throttle_level_1_curr = analogRead(L_POT);

int throttle_level_1_step = map(throttle_level[0], 0, 128, 0, 50);
int throttle_level_1_curr_step = map(throttle_level_1_curr, TL_max, TL_min, 0, 50);

int throttle_level_1_diff = PosTracking(throttle_level_1_step, throttle_level_1_curr_step);

if (throttle_level_1_diff > 0)
    L_Stepper.step(-1);

```

Figure 7: Motor Control Code Example.

the correct spot. During the motorization process where the motors are running, the potentiometer inputs from the levers are constantly checked against the desired position. This feedback loop ensures that the levers are positioned as accurately as possible. The feedback loop also self-adjusts to occasional slipping of the motors. Since the `step()` function in the standard Arduino library is a blocking function that prevents the execution of any other instructions before its completion, we cannot run the stepper motors in a continuous fashion. In order to move multiple motors together and ensure all button presses during the motorization process are still immediately recognized, the motors move alternatively one step at a time. The function is looped until both the step count and the potentiometer feedback indicates that the lever is in the correct position. Since the 32u4 is operating at 16 MHz, there should be no identifiable pauses or asynchronizations between levers.

## 2.5 Circuits & PCB

The PCB is the main carrier of E&E components inside the throttle quadrant. It consists of soldering points for the ICs and integrated power/signal lines. The circuit also hosts a collection of filtering and supportive peripheral circuits required for the ATmega 32u4 to function correctly and stably

The circuit is designed in compliance with ATmega’s recommendations and checked against Arduino’s open-sourced Leonardo design. We have added additional power rail filtering capacitors to further stabilize the system. This is crucial to the stability of the E&E system as the stepper motors can generate considerable amounts of noises when turned on. A polarized capacitor is used to further rectify fluctuations in the Vcc rail. Several LEDs are also added to indicate the physical working state of the serial port and 5 V Vcc. All ATmega I/O pins and motor driver I/O pins are routed to fly-wire headers to increase firmware design flexibility as well as to make the circuit easier to be diagnosed should an error occur. A soldering joint is used on the  $\overline{HWP}$  pin to allow for hardware bootloader override. This can be used in commercialized versions to prevent the firmware from being tampered with by the user. Figure 14 is the schematic of our final circuit design. A copy of the .sch file can also be found on the project homepage.

The PCB used in the final product is created from the circuit schematic shown above. When laying out the components, special care is taken to the sensitive filtering capacitors and noisier components. The filtering capacitors for the voltage lines are placed next to the respective power inlets to minimize the coupling capacitance brought by the PCB traces. The filtering capacitors for signal lines and the crystal oscillators are placed close to the microcontroller to maximize signal integrity. Fly-wire pin connectors are placed on the edges of the PCB for easy access and the traces are carefully wired to minimize the use of vias. Ground-planes are added to cover all components and the two ground-planes are connected by a 60-mil trace capable of hosting >10 A current with <10 °C temperature rise. The 12 V power rail also features thicker traces for higher current capacity and less temperature rises. Figure 15 is the final PCB layout. A copy of the .brd file can also be found on the project homepage.

## 2.6 Mechanical Parts

The mechanical portion of our design consists exclusively of user interfaces (levers and buttons). The three levers are connected to the output shaft of the three stepper motors. The two throttle levers on the right are mapped to the two engine of a twin-engine airplane, and the left most lever is mapped to the speedbrake handle. The arrangement is chosen to resemble the layout of a Boeing-spec airliner cockpit center pedestal. Two push buttons are located on the side of the body of the throttle quadrant and are mapped to TO/GA and A/T disengage.

### 2.6.1 Levers and Buttons

The three levers are fixed on one end with a hard-limited travel range enforced by slots cut into the throttle quadrant body. The angle at which the levers sit corresponds to the value of inputs made to the respective axis and is determined by single-turn linear potentiometers connected to the root of the levers. Moving the levers either by hand or through the stepper motors will alter the corresponding input inside the simulator. The levers are designed with a travel range close to that on a Boeing 737-800. The lever travel are mapped linearly to the allowed value ranges for respective inputs in the simulator by a linear mapping function in the microcontroller firmware, as discussed in Section 2.4. Two buttons are also present on the throttle quadrant. The first button is mapped to the TO/GA button that, once pressed, will engage A/T and immediately demand maximum allowed thrust from both engines. The second button is the A/T disengaged button. Pressing it once will disconnect the A/T system. Figure 8 is an illustration of the concept look of our product and a close-up picture of the final prototype is in Figure 16 in the Appendix.

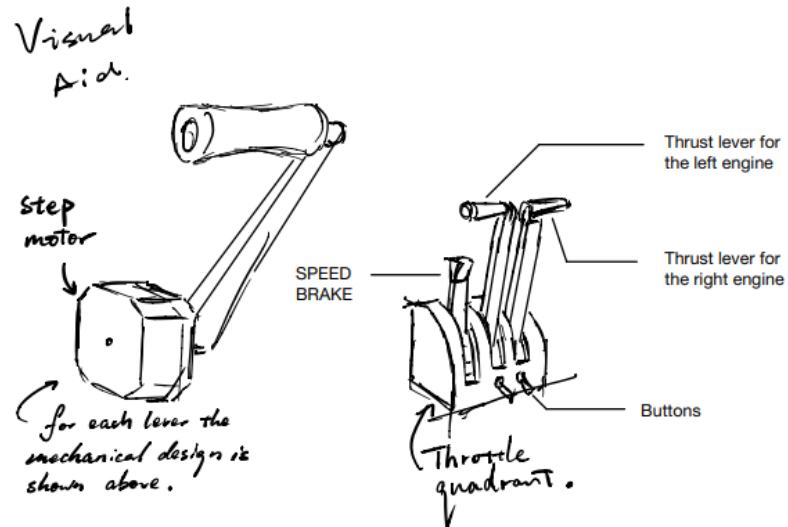


Figure 8: Conceptual Physical Design.

## 3 Design Verification

### 3.1 Host Add-On

The R&Vs of the host software are listed in Table 4. There are four major functionalities to be verified: 1) **SCThread** correctly sends user inputs to the flight simulator; 2) **SCThread** correctly receives simulation events of interest such as a lever position change or A/T status change; 3) **TQThread** correctly reads user inputs from the throttle quadrant hardware; 4) **TQThread** correctly sends lever positions to the hardware. Correctly implementing these four functionalities guarantees the connectivity between the flight simulator and the throttle quadrant hardware. In addition, the add-on should also correctly map between different data representations. For example, the flight simulator requires a throttle lever position to be a **double** number between 0 and 100, while the ASDF protocol encapsulates lever position as a 7-bit unsigned integer ranging between 0 and 127.

In practice, we first verify the implementation of ASDF protocol in the device firmware, as discussed in Section 3.2. Then, we use the verified firmware to test **TQThread**. Once **TQThread** is verified, we can directly test the whole system and examine any misbehavior and debug messages from **SCThread**. While directly verifying **SCThread** is possible, it requires significant amount of work just to develop a test bench due to the complexity of the Prepar3Dv4 SDK. We estimated that developing a test bench for **SCThread** requires comparable amount of effort as testing the system as a whole after all other components are verified.

In addition to correctness, we are also interested in the performance of the host add-on. In particular, we want to estimate the latency between the flight simulator and the throttle quadrant device. This latency approximates the time needed for a user input to be reflected in the virtual cockpit and for a visual change in the virtual cockpit to be reflected on the actual hardware.

Instead of profiling the latency directly by a global timer, which is almost impossible due to the natural complexity of the Prepar3Dv4 SDK, we split the critical path into three separate parts: 1) simulator to add-on latency; 2) add-on internal latency; 3) add-on to device latency. We estimate part 1) by measuring the average latency of a SimConnect API call. Part 2) is dominated by the synchronization between **SCThread** and **TQThread**, i.e., the time taken to read or write the shared structure. Part 3) is the time taken between sending an ASDF command and receiving the corresponding response. Empirically, part 1) takes on average 0.5 ms and at most 1 ms, part 2) takes 10  $\mu$ s to 20  $\mu$ s, and part 3) takes around 1 ms. Putting these numbers together, we calculate that the latency between the flight simulator and the throttle quadrant device is on average 1.5 ms and at most 2 ms.

### 3.2 ASDF Protocol

The verification of the ASDF protocol follows from the specifications in Table 2 and Table 3. To verify that the device correctly implements the ASDF protocol, we develop a Python script that accepts user inputs in terminal and sends corresponding ASDF commands to the device. When the user types `poll`, the script will send `CMD_POLL` to the device and displays any responses it receives (Figure 9). Note that the printed poll rate does not represent the actual poll rate in our implementation, which is written as a C++ program. To send `CMD_LVR_SET`, the user needs to type in `lvrset` followed by a bitmask indicating which lever to set followed by values to be set (Figure 10). To reset the device and the serial session, the user inputs `reset`, and the script will send `CMD_RESET` to the device and wait for the `ASDF_RESET` response (Figure 11). Other commands can be verified in a similar manner.

```

Provide command: poll
Command: 129
write size: 1
[2, 0, 106, 31, 8]
rate = 500.21514609421587 polls/sec
redundant: b''

```

Figure 9: Using Python script to verify CMD\_POLL.

```

Provide command: lvrset
Command: {'000': 130, '001': 146, '010': 162, '011': 178, '100': 194, '101': 210, '110': 226, '111': 242}
Input Lever Bitmask ({speed brake, thrust 1, thrust 2}): 011
Input Lever Values: 50 60
Lever Values: [50, 60]
Command: 178
write size: 1
write size: 1
write size: 1
b'\x00'
redundant: b''

```

Figure 10: Using Python script to verify CMD\_LVR\_SET.

```

Provide command: reset
Command: 128
write size: 1
Device reset done
redundant: b''

```

Figure 11: Using Python script to verify CMD\_RESET.

After verifying the ASDF implementation in the device firmware, we can use this verified firmware to test that the **TQThread** (Section 2.1.1) correctly implements the ASDF protocol. At the same time, we write microbenchmarks to measure the average latency for **CMD\_POLL** and **CMD\_LVR\_SET** by sending each command consecutively for more than one thousand times in C++. The average round-trip latencies for both commands are around 1 ms, implying that the host add-on is able to poll user inputs from the device at a rate of 1000 Hz and send lever position data to the device at a comparable rate. We also measured the latency of **CMD\_RESET** in a similar way, and the average latency of running 50 trials is 1.9 s.

### 3.3 Hardware

The R&Vs of the EE hardware are listed in Table 5.

#### 3.3.1 Circuit and PCB

The circuit is first compared to the open-sourced Arduino Leonardo reference board design to confirm that all the necessary supporting circuits are present for the microcontroller to work. For every revision of PCB manufactured, components are welded on to the board and immediately checked for accidental connection/disconnection between nodes. The microcontroller is then connected to a PC running AVRdude and Arduino IDE to confirm that it can be recognized by the driver as a valid ATmega 32u4 microcontroller. The frequency of the 16 MHz crystal oscillator is then checked with an oscilloscope to confirm that a steady 16 MHz signal is being fed into the 32u4’s external clock pin. To this point, the microcontroller and peripheral circuit is considered to be functional. The 12 V power is then introduced to the Vmotor line. A simple script provided by the Arduino stepper.lib library is used to run the motors. Heating and instability issues are observed for a few minutes to confirm that the PCB traces and soldering can withstand the current demanded by the motors. Then the Arduino bootloader is flashed onto the microcontroller. Figure 12 shows our ATmega 32u4 being successfully recognized by Windows 10 with correct device name and driver.

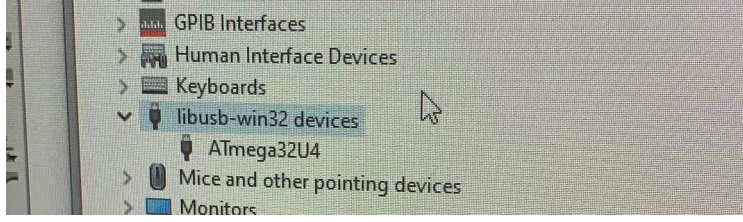


Figure 12: ATmega 32u4 Being Recognized by Windows 10.

### 3.3.2 Microcontroller Firmware

The firmware is tested once a stable serial connection can be established between the host and the microcontroller. First, the firmware is flashed with the Arduino bootloader, and we can see the microcontroller appears as an Arduino Leonardo at the COM port. Then we upload the firmware to the board and open the debug terminal. We can verify the functionality of the firmware with the verification of ASDF protocol, which follows the specifications in Table 2, Table 3 and Table 4. The functionalities of instruction decoding and data processing is verified by examining the debug terminal printout when buttons are pressed and levers are moved. The levers are moved, first individually then in groups, from the lowest position to the maximum position. The potentiometer readout is constantly printed on the serial debug monitor to verify the analog input taken from the potentiometers and buttons is correctly mapped to the accepted data range. The motor actuation firmware's verification follows the specifications in Table 5 and Table 6.

### 3.3.3 Mechanical Parts

The mechanical parts are verified by a Python test script we developed to test the functionality of ASDF protocol (Section 2.2). A CMD\_POLL command is issued to the microcontroller which gathers, maps, and encodes the readings of potentiometers and buttons to send back to the host PC to be printed in the debug console. The potentiometer range is checked in the serial monitor to be correctly mapped to the required 0 to 127 input range with a linear relation. The three levers are individually calibrated to satisfy the requirements as discussed in: Table 6. Any differences of the lever readouts caused by variations in the potentiometer, variations during the assembly, and the simulator's requirement of different data is accounted for, as discussed in: Figure 13.

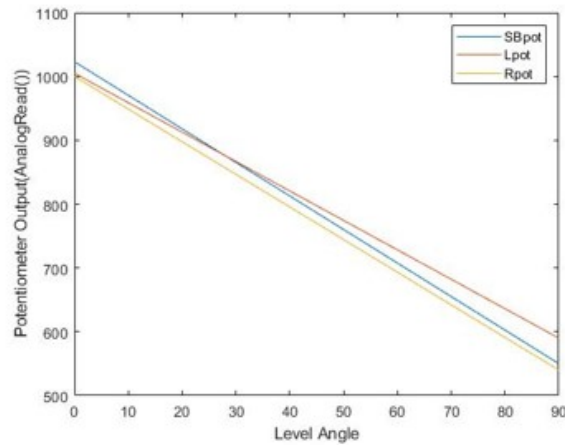


Figure 13: Voltage Output for Different Lever Angles and Different Levers.

## 4 Costs

### 4.1 Parts

Table 1 is the total component costs for a complete build of our product. This does not include the costs of damaged/extra components we purchased during R&D. The ECE Machine Shop actively contributed in the manufacturing of the mechanical parts of our product. It is possible to contract third-party manufacturers with the manufacturing of these parts and may alter the cost.

**Table 1: Parts Costs**

Part	Manufacturer	Retail Cost	Bulk Purchase Cost	Actual Cost
ATmega32u4-AUR	ATmega	\$ 4.66	\$ 3.66	\$ 4.66
17HS4401S	Usongshine	\$ 14.97	\$ 14.97	\$ 14.97
TB6612FNG 5V/12V H-bridge	Sparkfun	\$ 1.98	\$ 0.863	\$ 1.98
SMD Mini-B USB Female Connector	Amphenol ICC	\$ 1.11	\$ 1.11	\$ 1.11
16mm Push Button Switch	APIELE	\$ 9.66	\$ 7.74	\$ 9.66
12V 10A Adapter Power Supply	BINZET	\$ 19.98	\$ 15.98	\$ 19.98
PCB	JLPCB	\$ 12.6	\$ 0.10	\$ 12.6
Mechanical Parts	Machine Shop	\$ 35.0	\$ 5.0	\$ 35.0
<b>Total</b>			\$ 49.36	\$ 99.96

### 4.2 Labor

Given an average hourly wage of \$ 25 /hr for a graduate-level electrical engineer, we can estimate the rough total R&D labor cost of this project as shown in Equation (2). Our calculation is based on three development members working 12 hours per week for a total duration of 10 weeks. The parameters are estimated by the total workload experienced by our team members divided by the total length of this course after approval of this project.

$$3 \text{ engineers} \times \$ 25 / \text{hr} \times 12 \text{ hr/week} \times 10 \text{ weeks} = \$ 9000 \quad (2)$$

After applying the 2.5x modifier adjusting for actual personnel expenses, the total cost of this project in terms of labor is \$ 22500.

### 4.3 Profit Analysis

With the unit cost of production and labor cost of R&D determined, we can analyze if our project has sufficient room for profit if commercialized and marketed. Searching the key word "Flight Yoke System" on amazon.com shows 4 similar products at the same targeted \$ 200 price bracket as our product. The total rating count on these four products is 1584. Since none of these 4 competing products are motorized, we expect to be able to take away at least 50% of purchases with the motorization functionalities if our product is listed. We estimate that, with sufficient optimization of the process, the time required for a skilled engineer to assemble a complete product and perform basic quality assurance is 1.5 hours. Then we can estimate the listing price for our product assuming a 30% profit as shown in Equation (3).

$$\left[ \$ 49.36 \text{ /unit} + \left( \frac{\$ 22500}{1584 \text{ units} \times 50\%} \right) + 2.5 \times \$ 25 \text{ /hr} \right] \times 1.3 = \$ 182.35 \text{ /unit} \quad (3)$$

This estimated MSRP falls right in our targeted consumer-grade price bracket and can be a strong competitor with the innovative motorization features.



## 5 Conclusion

This paper has discussed our design of a motorized throttle quadrant aimed to improve the realism of consumer-grade desktop flight simulation. The software add-on written in C++ interfaces with the flight simulator via the SimConnect API and receive/send raw data corresponding to events within the simulator. The software add-on also interfaces with the microcontroller using the proprietary ASDF protocol. The ASDF protocol provides a compact and robust communication scheme for bidirectional data and command transfer between the host and the device. The microcontroller runs firmware designed to work alongside the ASDF protocol and control the activity of the stepper motors through H-bridge motor controllers. The immediate and direct communication pipeline allows for accurate following and setting of the thrust levers when A/T is engaged, and allows for responsive disengagements and overrides when the appropriate button is pressed<sup>1</sup>.

### 5.1 Accomplishments

#### 5.1.1 ASDF and Host Add-on

The ASDF protocol presented was successfully implemented and achieved  $> 1000$  poll-per-second transfer rate. The empirical latency between the transmission of an ASDF command to the receiving of all data packets is measured to be less than 2 milliseconds. We consider such performance to be above our initial expectation and the requirement of all possible operating scenarios.

The host add-on can be recognized by Prepar3Dv4 as a valid modification and can successfully control related events inside the simulator. The multi-threaded design scheme adopted by the host add-on prevents blocking events in the operation of the system and can achieve an input delay of less than 5 millisecond. This exceeds our delay tolerance of 100 ms.

#### 5.1.2 Microcontroller Firmware

The communication firmware is able to interface with the host add-on via serial connection over USB. The firmware uses standard serial connection and does not require the installation of additional proprietary driver. The device can be signaled to reset spontaneously through instruction timeout, and automatically performs setup again upon boot-up. This enables plug-and-play capabilities that does not require a restart of the simulator for the throttle quadrant to function.

The ASDF parsing and processing firmware is able to efficiently convert between raw data from potentiometers and buttons to standardized ASDF data packets. The mapping functions enable the levers to perform in a predictable, linear manner. The switch-case based parsing structure minimizes the round-trip response time of the microcontroller to allow the completion of a poll-type request in less than 3 ms.

The motor actuation firmware is able to control all motors via PWM signal and directional control signals issued to the motor drivers. The levers can move in synchronization with their virtual counterparts, The firmware can also perform hard stopping and halting to further enhance the performance and responsiveness of the motors.

#### 5.1.3 Circuits

The supportive peripheral circuits are capable of sustaining the operation of the microcontroller steadily. The filtering capacitors can maintain the supply line voltages to within 0.2 V of specified level. The crystal oscillators operate at a steady 15.9998 MHz.

---

<sup>1</sup>Demo videos are available online at: video 1, video 2

The ground planes and thickened traces on the printed circuit board can carry the peak 4.5 A current draw without overheating. Heatsinks attached to the motor drivers keeps them cool even during sustained operation of the throttle quadrant.

## 5.2 Uncertainties

### 5.2.1 Power Supply

Due to the large amount of torque required to accurately position the levers without slipping, the motors will draw high currents ( $> 5\text{A}$ ) at onsets of lever movements. The high current demand can pull down on the 12 V line voltage and cause immediate loss of torque and subsequently slipping of the rotor. Adopting high current supplies from OEM sellers will lead to increases in the overall product cost as well as bulkiness of the product. The bigger power supply may also increase loads on the passive cooling system we have adopted.

### 5.2.2 Ergonomics

Consistent with Boeing airplane systems, our A/T override sequence starts with the pilot pressing the A/T disengage button. On real Boeing aircraft, the buttons are placed on the tips of the thrust levers, one per side. Such button arrangement allows the pilot to quickly and intuitively press them and send the A/T override signal. Due to prototyping limits, we cannot adopt physical designs that accurately reflect the thrust levers on real aircraft. Our current setup with both buttons on the side of the body of the thrust quadrant may cause ergonomic inconveniences and compromises the realism somewhat.

## 5.3 Ethical Considerations

Our throttle quadrant aims to provide an actual and smooth experience to the audience of flight simulations, we believe that our product can bring a more authentic and affordable option to the publicity, which facilitates the wide audience of flight enthusiasts to access actual flight experience, and may serve as training equipment before they step up to planes. Thus, our design aligns with the IEEE Code of Ethics Section 7.8.2: “to improve the understanding by individuals and society of the capabilities and societal implications of conventional and emerging technologies, including intelligent systems” [10].

Since our design-related extensively to mechanical and electrical parts, the safety concerns related to our physical design during design, experiment, and use need to be addressed. Such safety concerns include potential safety threats towards users, the misuse of our design may cause undesired consequences to the user, and the peripheral itself. To optimally be following the IEEE Code of Ethics Section 7.8.9, striving to reduce potential harm to the user and the property, we make efforts to optimize our design [10].

Our design involves 5 V/12 V power supplies to drive our microcontrollers and the stepper motors. Although the voltage does not exceed the safety voltage of 22 V, leakages may lead to physical irritation or fire hazards. To address the concern regarding electric shocks and leakages, the handle of our design will be made from or covered by insulating materials to isolate the user from the electrical components, and to reduce the chance of foreign objects from damaging the circuits inside.

The lever movements when A/T is engaged are controlled by the simulator program and, if not regulated, may cause physical damage to the user. There is a possibility for motor overheating and damage if the lever is obstructed or stuck. It is also possible for the user to be impacted by the automatically-moving levers. To address the safety issues concerning mechanical components, we will control and limit the travel of our thrust lever through both physical blockage and current limits. Also, the firmware mapping function ensures that the motors will never be instructed to move the levers outside of the designated range.

## 5.4 Future Work

### 5.4.1 Motorized Speedbrake

Apart from the throttle levers we successfully motorized in this project, Boeing airplanes also has motorization enabled on the speedbrake lever. When the main gears touch the ground with the speedbrake lever in the ARMED position, the speedbrakes are automatically deployed and the speedbrake lever will move to the fully extended position. The addition of a motorized speedbrake lever will enhance the realism brought by the throttle quadrant even more. We already have the motor, potentiometer, and wiring to implement this functionality. But due to time constraints imposed by the course, we have elected to leave such feature to be completed in the near future.

### 5.4.2 Circuit Integration

Being a prototype device, our current version of the throttle quadrant features a functional but modular PCB design. There still exists debug pin-outs and it requires fly-wires to connect to the motors and the potentiometers. While the functionality of such circuit is not a concern, a fully integrated PCB with every component soldered onto it will make the overall construction more rigid and robust against unexpected impacts.

### 5.4.3 Replica Levers

Due to the limiting manufacturing capability we have in the prototyping stage, we elected to use 3D-printed levers in our prototypes. These levers do not represent the real cockpit of the aircraft we intended them to be matching hence may reduce the realism brought by the throttle quadrant. These prototype levers are also not ergonomically designed in anyway to be comfortable in the pilot's hands. To improve on this, we can use 3D-printed replicas of real Boeing thrust levers. Having matching lever designs will also allow the TO/GA and A/T disengage buttons to be fitted to the correct locations for ease of access. Not only will realism be augmented with aesthetically-correct levers, the improved ergonomics will also make pilot inputs feel more natural and smooth.

## References

- [1] “Microsoft Flight Simulator: Standard.” [Online]. Available: <https://www.microsoft.com/en-us/p/microsoft-flight-simulator-standard/9nxx8gf8n9ht>. [Accessed: 28 Feb 2021].
- [2] “Prepar3D.” [Online]. Available: <https://www.prepar3d.com/>. [Accessed: 28 Feb 2021].
- [3] “Logitech G Flight Simulator Yoke with Throttle Quadrant.” [Online]. Available: <https://www.logitechg.com/en-us/products/driving/driving-force-racing-wheel.html>. [Accessed: 28 Feb 2021].
- [4] “The Best Flight Simulator Hardware and Software for Pilots in 2021.” [Online]. Available: <https://hangar.flights/guides/flight-simulator-student-pilots/>. [Accessed: 3 Mar 2021].
- [5] “Logitech G920 & G29 Driving Force Steering Wheels & Pedals.” [Online]. Available: <https://www.logitechg.com/en-us/products/driving/driving-force-racing-wheel.html>. [Accessed: 28 Feb 2021].
- [6] “HW-FFB-CSN - Flight Illusion.” [Online]. Available: <https://www.flightillusion.com/product/hw-ffb-csn/>. [Accessed: 28 Feb 2021].
- [7] “Console Latency: Exploring Video Game Input Lag.” [Online]. Available: <https://displaylag.com/console-latency-exploring-video-game-input-lag/>. [Accessed: 4 Mar 2021].
- [8] “Prepar3D SDK.” [Online]. Available: <https://www.prepar3d.com/SDKv4/LearningCenter.php>. [Accessed: 5 Mar 2021].
- [9] “Serial Port Programming on Windows using Win32 API.” [Online]. Available: <https://www.xanthium.in/Serial-Port-Programming-using-Win32-API/>. [Accessed: 5 Mar 2021].
- [10] IEEE, “IEEE Code of Ethics,” 2020, IEEE. [Online]. Available: <https://www.ieee.org/about/corporate/governance/p7-8.html>. [Accessed: 18 Feb 2021].

## Appendix A ASDF Protocol Specification

**Table 2: Host Commands for the ASDF Protocol.**

Command	Opcode	Function
CMD_RESET	0x80	Reset the throttle quadrant device.
CMD_POLL	0x81	Get current status of the thrust levers and the buttons.
CMD_LVR_SET	0x{8-F}2	Lock and set lever positions. This command is actually a combination of 8 commands. Bit 6 to bit 4 form a bitmask indicating which thrust levers are to be set. The opcode is then followed by the position values in the order of the bitmask. If none of the bits are set, this command does nothing, and no following value bytes are required.
CMD_LVR_RELS	0x83	Release the thrust lever (and might let the flight simulator disengage A/P).
CMD_ASDF	0xFF	Reserved for debug purposes.

**Table 3: Device Responses for the ASDF Protocol.**

Response	Format	Function
ASDF_ERROR	0xFF	Response when the device encounters an error.
ASDF_ACK	0x00	A generic response when the device successfully completes a command.
ASDF_RESET	0x01	Generated when the device finishes initialization, which might be the result of a CMD_RESET command or the normal bootup.
ASDF_POLL_OK	0x02	First byte of the response to a CMD_POLL command. This byte will be followed by the data bytes asked by the command.
ASDF_LVR_RELS	0x{0,8}3	Generated when the thrust lever is released to pilot's control. If bit 7 is set, then this response is the result of a CMD_THR_RELS command. Otherwise, this indicates that the pilot wants to forcibly disengage A/P by moving the thrust lever.

## Appendix B PCB Schematic and Layout

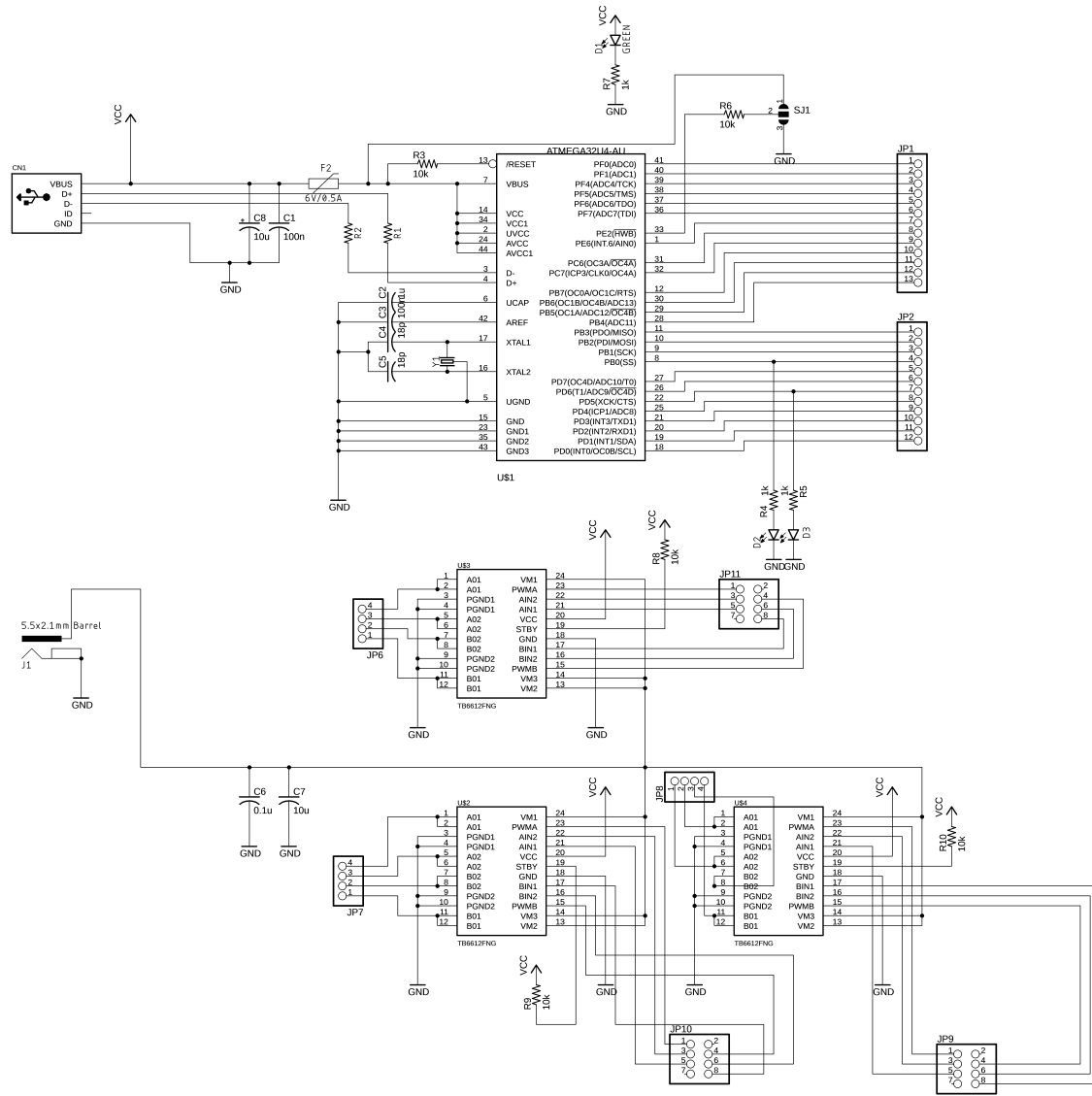


Figure 14: Circuit Schematic.

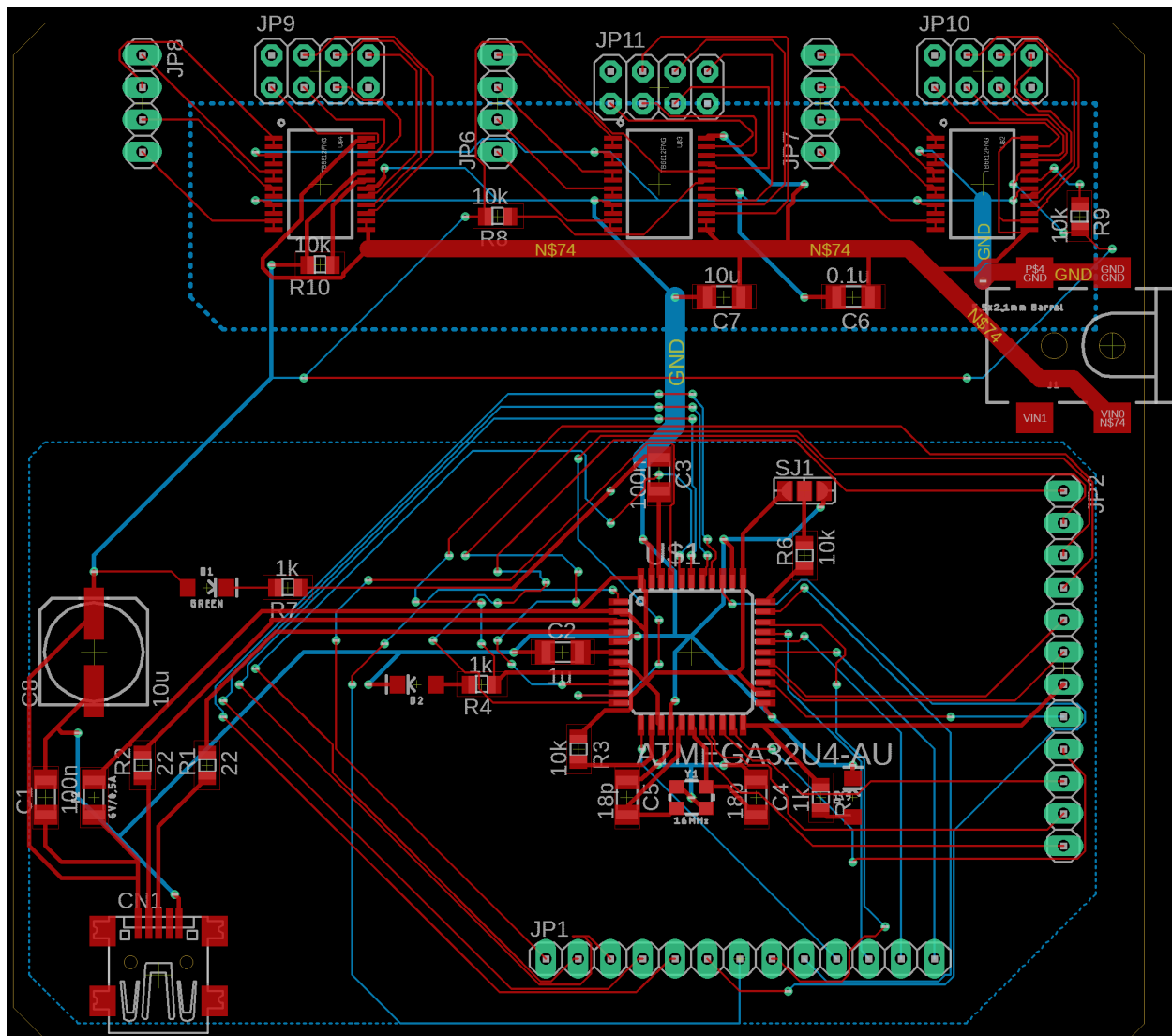


Figure 15: PCB Layout.



## Appendix C Exterior Appearance of Final Prototype

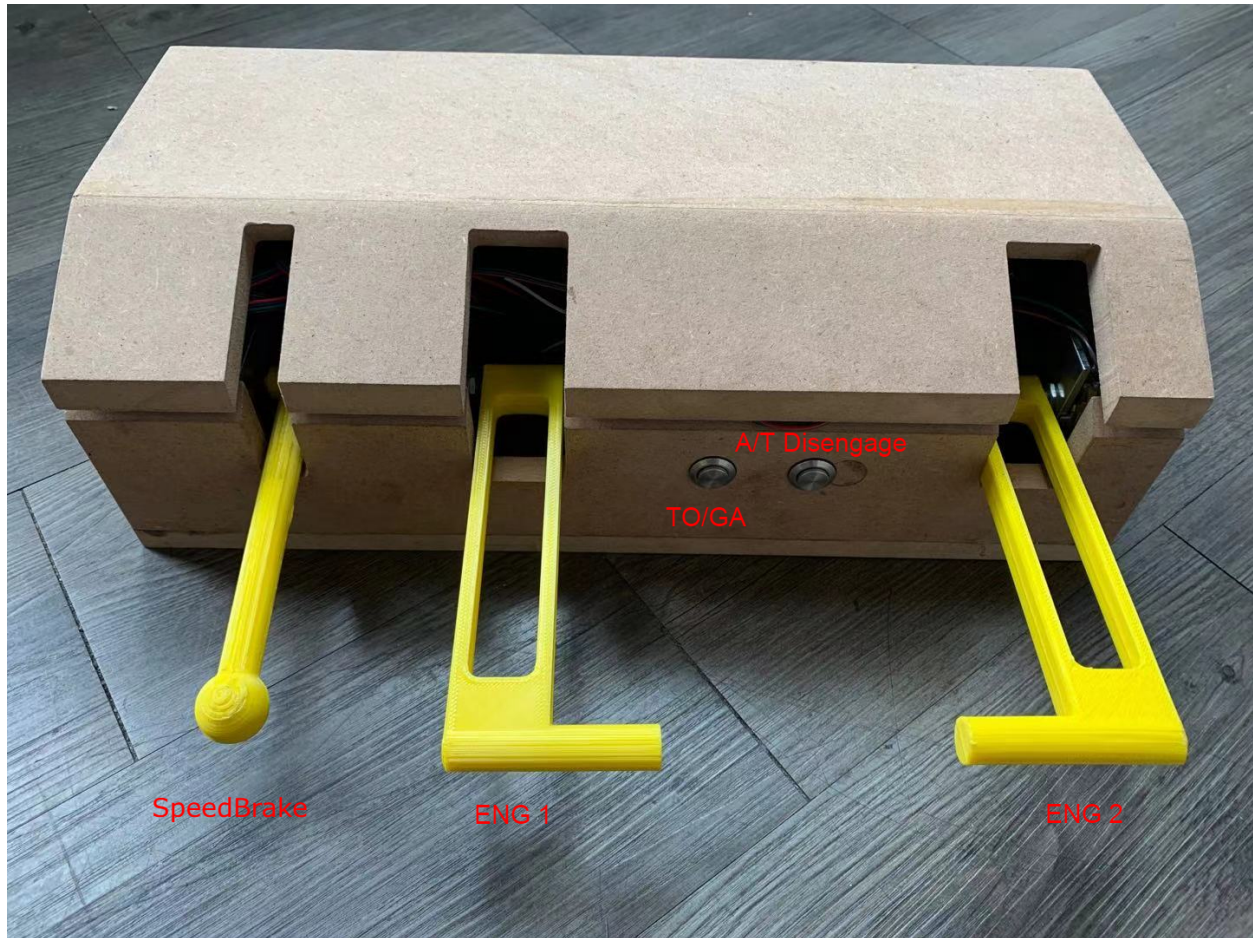


Figure 16: Exterior Appearance of Final Prototype.



## Appendix D Requirement and Verification Table

Table 4: R&Vs for Host Software.

Requirement	Verification	Verified
The Flight Simulator (FS) Add-on can read flight data from the simulator software via its API calls.	<ol style="list-style-type: none"> <li>1. Start the simulator program with the add-on.</li> <li>2. The debug terminal of the add-on should appear automatically or be opened manually.</li> <li>3. Load a scene in the simulator program, bring the aircraft into sky, manually adjust thrust demand for both engines and extend the speedbrake.</li> <li>4. Verify that the debug terminal of the add-on prints correct data that agree with the flight instruments in the virtual cockpit.</li> </ol>	Y
The FS Add-on can write data via the simulator API to control the airplane with < 10ms delay.	<ol style="list-style-type: none"> <li>1. Start the simulator program with the add-on and the debug terminal.</li> <li>2. Load a scene in the simulator program.</li> <li>3. Send thrust demand change commands via the debug terminal and start the timer.</li> <li>4. Confirm requested flight control changes correctly takes effect and end the timer.</li> <li>5. The time between command sent and flight control response should be less than 10 ms.</li> </ol>	Y
The FS Add-on correctly maps data ranges between the flight simulator data and the raw user inputs.	<ol style="list-style-type: none"> <li>1. Connect the throttle quadrant to the host computer.</li> <li>2. Load the test firmware onto the throttle quadrant microcontroller. The test firmware emulates the user inputs by continuously changing the throttle lever position variables and intermittently changing the button status. When the host add-on sends a poll command, the emulated data will be sent rather than the real user inputs.</li> <li>3. Start the simulator program with the add-on.</li> <li>4. Load a scene in the simulator program.</li> <li>5. Observe that the levers in the virtual cockpit move continuously and can reach both ends.</li> <li>6. Observe that the buttons in the virtual cockpit are also automatically pushed.</li> </ol>	Y

<p>The FS Add-on can read (write) data from (to) the throttle quadrant microcontroller via USB Serial Protocol within <math>&lt; 100</math> ms delay in each direction.</p>	<p>Note: This test procedure assumes that the throttle quadrant microcontroller correctly handles USB Serial Communication with the host and correctly controls at least one motorized lever.</p> <ol style="list-style-type: none"> <li>1. Start the flight simulator with the add-on.</li> <li>2. Connect the throttle quadrant to an operating USB 3.0 port.</li> <li>3. Load a scene in the simulator program.</li> <li>4. The data read from the throttle quadrant will be automatically printed onto the debug terminal every 250 ms. By default, A/T is not active, and the add-on will continuously send poll commands to the device. A running average latency is also measured for each poll command.</li> <li>5. Move any levers on the throttle quadrant. Verify that correct data is printed in the ATmega debug terminal, and that the printed latency for poll commands is less than 100 ms.</li> <li>6. Push the TOGA button on the throttle quadrant. This will put the add-on into A/T engaged state so that it starts sending lever set commands.</li> <li>7. Observe that the throttle levers move accordingly with the throttle levers in the virtual cockpit.</li> <li>8. Verify that the lever set latency printed in the debug terminal is less than 100 ms.</li> </ol>	<p>Y</p>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------

**Table 5: R&Vs for Electronic Hardware.**

Requirement	Verification	Verified
The microcontroller firmware can decode and encode data in the correct format.	<ol style="list-style-type: none"> <li>1. Start the simulator program with the add-in and the debug terminal.</li> <li>2. Connect and set up the throttle quadrant with ATmega serial debug terminal monitoring.</li> <li>3. Send thrust demand change and speedbrake deployment commands to the throttle quadrant.</li> <li>4. Verify the successful decoding of the test pattern in the serial debug terminal.</li> <li>5. Manually apply test signals to the microcontroller via debug ports. (bypassing hardware encoder/motor feedback)</li> <li>6. Verify the reception of test signals in the Add-in debug terminal.</li> </ol>	Y
The motor driver can advance the stepper motor without slipping and stop within 1.8° of assigned position.	<ol style="list-style-type: none"> <li>1. Power up the throttle quadrant with 12 V DC.</li> <li>2. Manually send thrust lever position change commands to the motor driver circuit (bypassing the microcontroller).</li> <li>3. Verify all 3 steppers can start without slipping and stop within 1 step (1.8°) of assigned location after continuous (more than 2 steps) motion.</li> </ol>	Y
The stepper motors can move all levers smoothly. Individual levers can move without affecting its neighbors. The motors sustain long operations without reach above 60°C.	<ol style="list-style-type: none"> <li>1. Fully power the throttle quadrant (5 V rail and 12 V rail).</li> <li>2. Open ATmega debug terminal and send stress test command (cycle all motors at full power continuously).</li> <li>3. Verify the movements of the levers are steady at 12.6° /s turn rate (7 seconds to complete idle-TO/GA travel).</li> <li>4. Leave test running for at least one hour.</li> <li>5. Confirm with infrared camera that the stepper motor packages are below 60°C.</li> </ol>	Y

<p>The microcontroller can accurately respond to virtual thrust adjustments and adjust the thrust levers accordingly with delay of &lt; 100ms.</p>	<p>Note: This test procedure assumes that the software host is functional.</p> <ol style="list-style-type: none"> <li>1. Start the simulator program.</li> <li>2. Connect the throttle quadrant to an operating USB 3.0 port.</li> <li>3. Load compatible scenario.</li> <li>4. Verify the levers are moved to standby positions when the aircraft system is initialized.</li> <li>5. Engage A/T inside the simulation software and command a thrust change via the MCP and start a timer.</li> <li>6. Verify the movement of the thrust levers follows that of the thrust levers in the virtual cockpit and stop the timer</li> <li>7. The delay should be no more than 100 ms.</li> <li>8. Extend the speedbrake inside the virtual cockpit and start a new timer.</li> <li>9. Verify the speedbrake lever moves to match its position in the virtual cockpit and stop the timer.</li> <li>10. The delay should be no more than 100 ms.</li> </ol>	<p>Y</p>
<p>The power supply is capable of motorizing all step-pers at 12V 0.5A each constantly without reaching &gt; 60°C on the unit package.</p>	<ol style="list-style-type: none"> <li>1. Power up the throttle quadrant 12 V rail with the power supply.</li> <li>2. Send power stress test command via the ATmega debug terminal (run all motors at 12 V 0.5 A continuously).</li> <li>3. Confirm with an oscilloscope that the output of the power supply is at <math>12 \pm 0.5</math> V DC.</li> <li>4. Keep the test sequence running for 1 hour</li> <li>5. Confirm with infrared camera the temperature of the exterior of the power supply unit is &lt; 60°C</li> </ol>	<p>Y</p>

**Table 6: R&Vs for Mechanical Parts.**

Requirement	Verification	Verified
The levers can be mapped to corresponding functionalities in the simulation software.	<p>Note: This test procedure assumes that the software host and the EE hardware are functional.</p> <ol style="list-style-type: none"> <li>1. Open the simulator settings and select throttle/speedbrake lever mappings</li> <li>2. Map the physical levers to corresponding controls and initiate calibration.</li> <li>3. Confirm the levers can be mapped to full control range (determined automatically by the simulation program) with at least 1:1 resolution.</li> </ol>	Y
The TO/GA button and both A/T disengage buttons are functional with < 100ms input latency.	<p>Note: This test procedure assumes that the software host and the EE hardware are functional.</p> <ol style="list-style-type: none"> <li>1. Start the simulator program.</li> <li>2. Load a compatible scenario.</li> <li>3. Setup FMC/CDU for takeoff.</li> <li>4. Press the TO/GA button and start a timer.</li> <li>5. Confirm on the PFD that A/T mode has changed to TO/GA and stop the timer.</li> <li>6. The delay should be no more than 100 ms.</li> <li>7. Press A/T disengage button and start a new timer.</li> <li>8. The A/T ARM switch on the MCP flicks to off position. The Master Caution is lit up. A/T DISENGAGE is shown on the MFD recall section and stop the timer.</li> <li>9. Verify that the delay is no more than 100 ms.</li> </ol>	Y