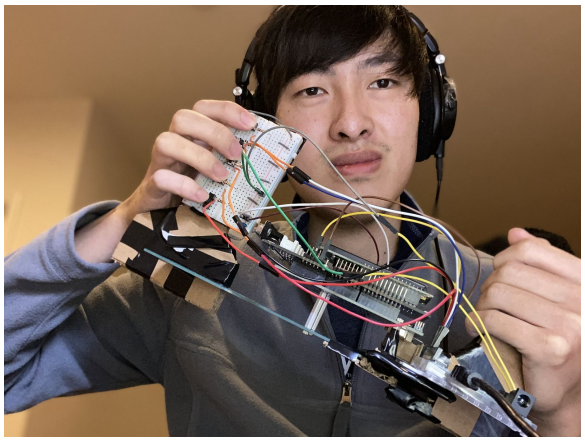
Open Flight Console
EECS 149 Final Project

Trevor Wu, Christopher Stephens, Andy Huynh



Objective

The objective of our project was to build a controller which resembled an airplane steering wheel and could be used to give a player a more immersive experience in a flight simulation. With intuitive button placement, Hypothetically, the work from our project can grow into a more flexible and portable flight simulation controller since the steering wheel does not need to be fixed to an axle and can be easily recalibrated if the player decides to sit in a different position.

Overview

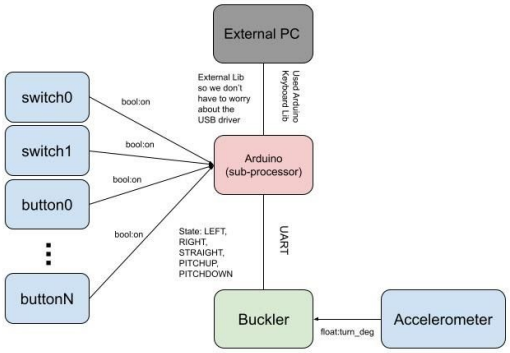
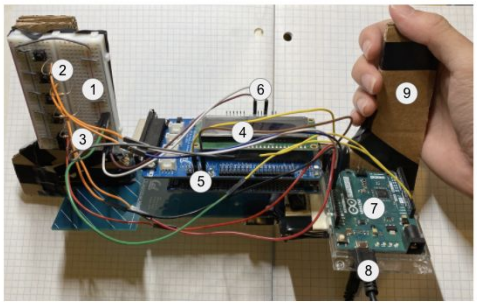


Figure 1: Hardware Architecture of the System

For this project, we used two microcontrollers -- the Berkeley Buckler to record the position of the steering wheel (with its built in accelerometer) and display (for debugging), and an Arduino Leonardo microcontroller for its easy-to-use Keyboard and Mouse integration. The Arduino is then connected to an external PC upon which we launch a video game (in the demo, this game was Minecraft). The hardware consists of the Buckler, the Arduino, Breadboards, Buttons, Switches, Jumper Wires, a MicroB to USB cable, and grips to hold the controller. The way the hardware is connected to each other is detailed in Figure 1 and a labeled photo of the controller is detailed in Figure 2.



1. **Breadboard** to connect buttons to resistors and power
2. **Buttons** that can be remapped easily for different games
3. **Power Switch**
4. **Buckler**
5. **UART connection** between Buckler and Arduino
6. **Buckler Power Connection** to get electricity from Arduino
7. **Arduino Leonardo** to interface with computer and has built-in keyboard/mouse library
8. **USB-MicroB to Computer** to communicate with and receive power from external PC
9. **Grip** to simulate the feeling of a flight steering wheel

Figure 2: Labeled Diagram of Controller

On the software side, we wrote C code on both the Arduino and the Buckler to implement respective state machines which map user inputs to outputs the computer can read. The buckler has five states: OFF, PITCHUP, PITCHDN, STRAIGHT, LEFT, RIGHT -- which are mapped according to the position of the accelerometer. It then outputs a character which represents that state. The code for the Buckler (states and its transitions) is described in Figure 1. The Buckler transmits its output via UART to the Arduino. The Arduino reads the output from the Buckler and converts that into a key press or a mouse command. A number of buttons and switches are also connected to the Arduino and those are independent of the Buckler, although we make sure that they do not map to conflicting commands. This is detailed in Figure 2.

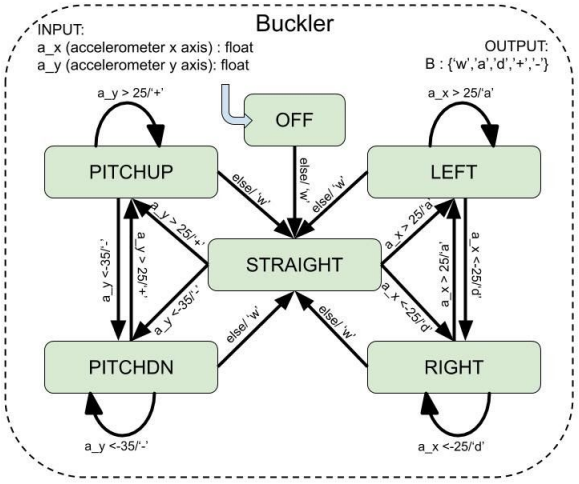


Figure 3: State Machine of the Buckler

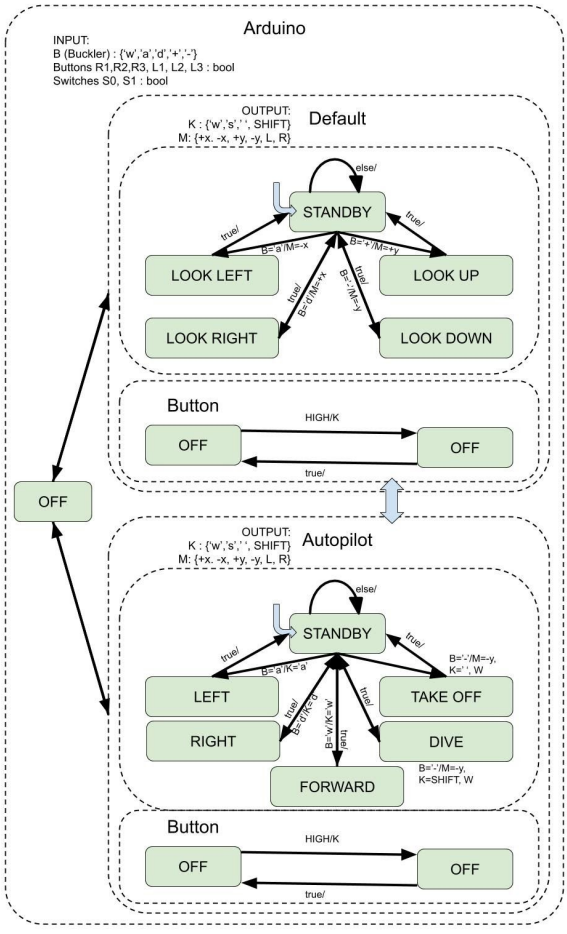


Figure 4: State Machine of the Arduino

Switches are used to turn off/on the controller and to toggle between Default and Autopilot mode. In Default mode, moving the controller around is like moving the mouse. The buttons then control other necessary movements in the game. In Autopilot mode, the controller automatically moves forward and steering it just changes the direction. The buttons are then used to bind to in game items. In both modes, the buttons R1, L1 are used to control the right and left mouse buttons respectively.

Class Topics

Our project involves four course concepts in its implementation. To detect its tilt and to calibrate its position, the steering wheel uses an accelerometer. This is an example of using a *sensor* (Chapter 7: Sensors and Actuators) because we are measuring a physical quantity -- the free fall of the buckler. The accelerometer is sampled 1000 times a second. We also calibrate the sensor so that the position it starts in will always be 0 degrees x, and 0 degrees y. This is done by taking the initial measurements when the buckler is restarted, and then subtracting those values from the subsequent readings.

We also use *state machines* extensively as a way to model the dynamics of the controller. Our project can be described as a hierarchical state machine with both asynchronous composition and cascade composition. The buttons are an example of an asynchronous side-by-side composition because the actions mapped to those buttons can happen all at once, not at all, and in any combination. The interaction between the Buckler and the Arduino (steering part) can be described as an *asynchronous cascade composition* of two discrete systems. This is because the Arduino (steering part) takes in the output of the Buckler as one of its inputs.

Input/Output concepts are used when we connected the buckler to the Arduino and when we connected the Arduino to the computer. UART is used to connect the buckler to the Arduino. We used UART because it was a universal port present on both microcontrollers, it provided enough bandwidth for a character, and it was asynchronous with the help of a buffer. The Buckler sends an output every millisecond while the Arduino scans for inputs

every 10 milliseconds (the slower refresh rate was chosen so that it doesn't overload the computer). Without the buffer, the Arduino might miss some messages sent by the Buckler.

The last concept we heavily relied on for this project is *language refinement*. Minecraft is a computer game controlled by a keyboard and mouse. In order for our controller to work, it must be l*anguage refinement* of a keyboard and mouse. This means that our controller must be able to map to all behaviors required to play the game. *Language refinement* is best shown in the Arduino code we wrote for the controller. Table 1 shows how many behaviors in game can be simulated by the controller and how the controller can replace the keyboard in those scenarios.

Table 1: Type Compatibility of Controller

Behavior	Keyboard/ Mouse	Controller Action
Look Up	Mouse Up	Pitch Up
Look Down	Mouse Down	Pitch Down
Look Left	Mouse Left	Turn left
Look Right	Mouse Right	Turn Right
Left Click	Mouse LB	Button L1
Right Click	Mouse RB	Button R1
Crouch	SHIFT	Button L2
Jump	SPACE	Button R2
Move Forward	w	Button L3
Move Backward	s	Button R3

System Evaluation

We used Minecraft as a platform to evaluate the system. Minecraft is a game with fly mode that can be a primitive flight simulator. It is also a game with relatively few controls.

Table 2: Minecraft Performance Indicators

Indicators	Evaluation	Score 1-5 (best)
Latency and Responsiveness	It can take a fraction of second for the controller to realize its tilt was changed. Some buttons are not always responsive	2
Compatibility (can I play Minecraft without the need of other peripherals?)	Can look around, walk in all directions, attack mobs and mine blocks, place blocks, and select from the hotbar. Still missing some features such as inventory and hotkeys.	3
Playability (is it easy to use and are the controls intuitive?)	It relies on just a USB connectivity and is pretty "plug and play". Holding the controller sometimes feels a bit awkward due to the breadboard. Button placement and controls are easy to learn.	3
Aesthetics	It looks like a tangle of electrical tape, chips, and wires.	1

Our system works reasonably well in Minecraft. We were able to perform many necessary actions within the game without touching the mouse or the keyboard. After calibrating the transition gates, we were also able to make the controls feel natural. Figure 5 shows that the range of motion for the controller is reasonable for *able-bodied* gamers. If a user has a disability that impedes their range of motion, they might need to change some of the transition gate values.

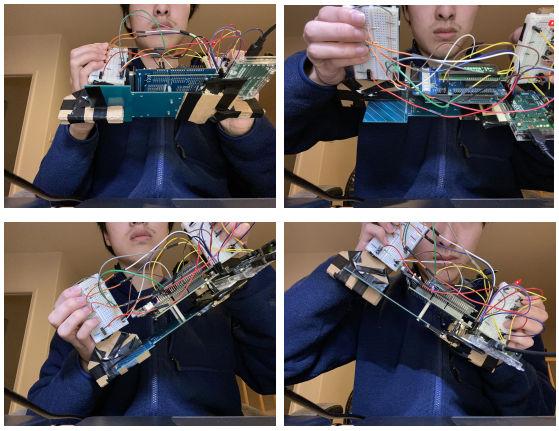


Figure 5: Controller in Action, clockwise from top left: pitch up, pitch down, left, right

There were some glitches in the system. One of the persisting problems was a slight delay between changing the direction of the controller and the computer reflecting that change. Another problem was some unresponsiveness in some of the buttons. We believe that the slight delay in direction change is due to how the communication between the microcontrollers was set up. The Arduino loops every 10 ms while the Buckler loops every 1 ms. This means that when the Arduino finally reads a message, the message was likely sent 9 ms ago. Looking at the specs of the Arduino, we found that the Arduino had a UART buffer of 64 bytes. For comparison, a char output from the Buckler is 1 byte. This means that if the buffer were full, it would take the Arduino 64 * 10 ms = 640 ms to clear out all the backed up commands. The lower refresh rate on the Arduino was chosen so that it would not overburden the computer's processor with up to 1000 keyboard strokes a second constantly. However, one way we can possibly alleviate this problem would be to set the Buckler to loop every 10 ms instead of every 1 ms.

The delay in buttons might be caused by a weak connection in the breadboard or the quality of the button. The code for the buttons was designed such that holding down a button for a few seconds still only counts as one press. This was to prevent a brief click from turning into 10 clicks due to the refresh rate. This might also cause delays if one wants to "hold down" a button.

Conclusion

Our project turned out quite well despite the challenges of working in a fragmented team remotely. We had the opportunity to really dive in depth to a few concepts in class and build a new controller based on those concepts. We managed to build a controller that can function as a type-compatible replacement for a keyboard and mouse. The controller is also quite intuitive, sensitive, and provides flexibility for many use cases. If we had more time and resources, we would have liked to make our controller more unique by integrating a feedback system with a Servo motor and an encoder so it can reflect actions in game during autopilot mode. We would also like to make a more advanced autopilot mode that can do some navigation if we had the time. Ultimately, it was a very educational experience and a project that can be expanded upon in the future.

References

"Keyboard - Arduino Reference". *Arduino.Cc*, 2020, https://www.arduino.cc/reference/en/language/functions/usb/keyboard/.

"Lab11/Buckler". *Github*, 2020, https://github.com/lab11/buckler/tree/a71a9707c03eb4b5b9b8f3a3096d44e2ee75d8ff.

Lee, Edward A, and Sanjit A Seshia. *Introduction To Embedded Systems*. 2nd ed., The MIT Press, 2017.

"Mouse - Arduino Reference". *Arduino.Cc*, 2020, https://www.arduino.cc/reference/en/language/functions/usb/mouse/.