Review the assignment due date

(AT)Mega Drivers Final Project

Team Number: 26

Team Name: (AT)Mega Drivers

Team Member Name	Email Address
Xiang Chen	xianc@seas.upenn.edu
Shawn Lim	shawnlzh@seas.upenn.edu
Emily Baylock	ebaylock@seas.upenn.edu

GitHub Repository URL: https://github.com/upenn-embedded/final-project-website-submission-f25-t26-f25-at-mega-drivers

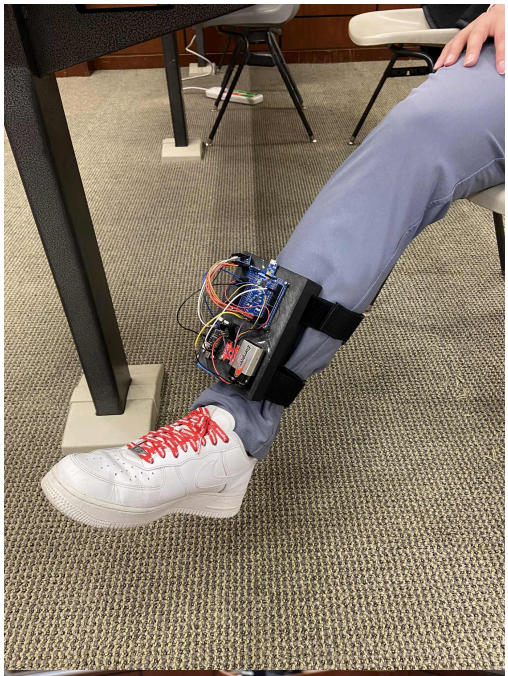
GitHub Pages Website URL: https://upenn-embedded.github.io/atmega-drivers/

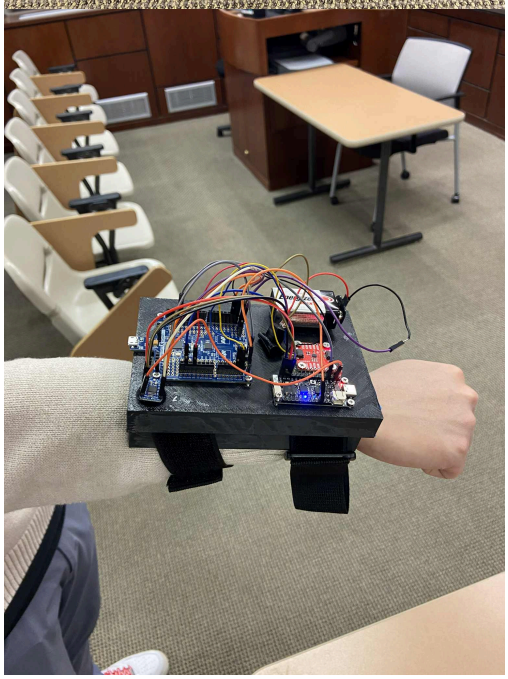
Final Project Report

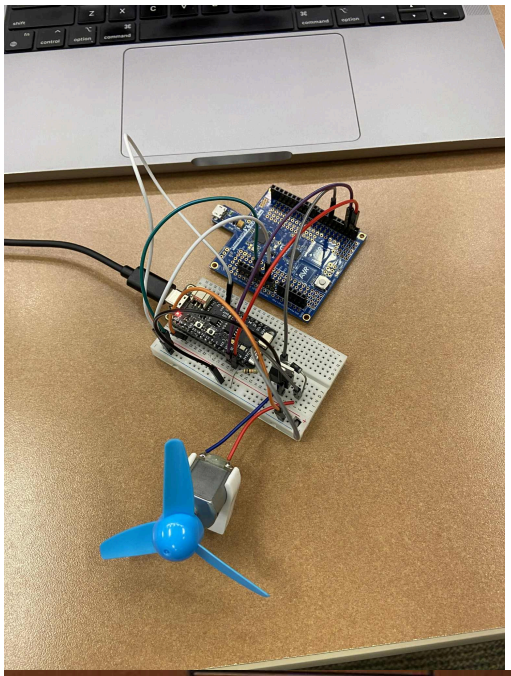
Project Video

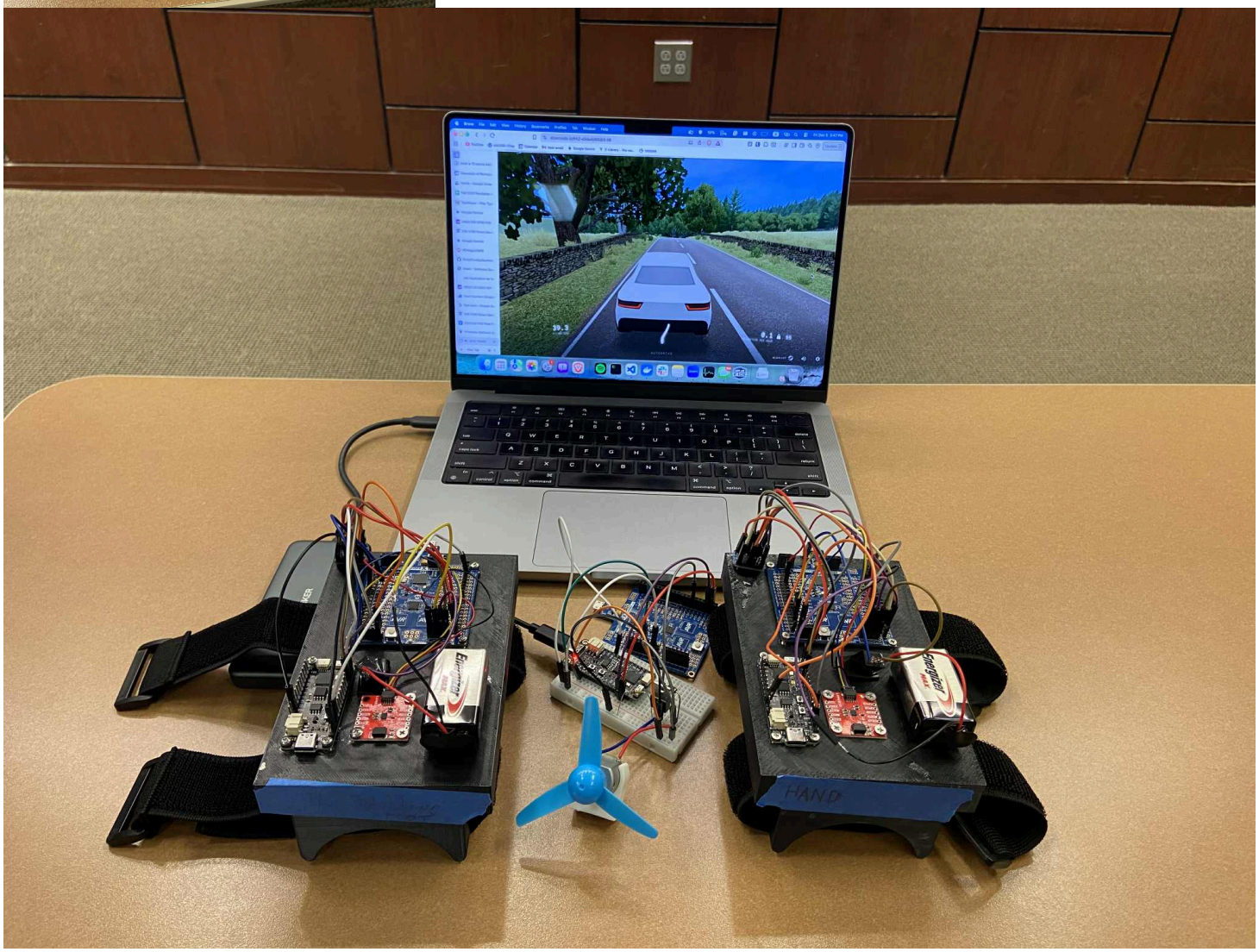
Watch our final project video here:
https://drive.google.com/file/d/1IMHc0yKnxAkiVJNU0LkIz9THIGWeNgBH/view

Images





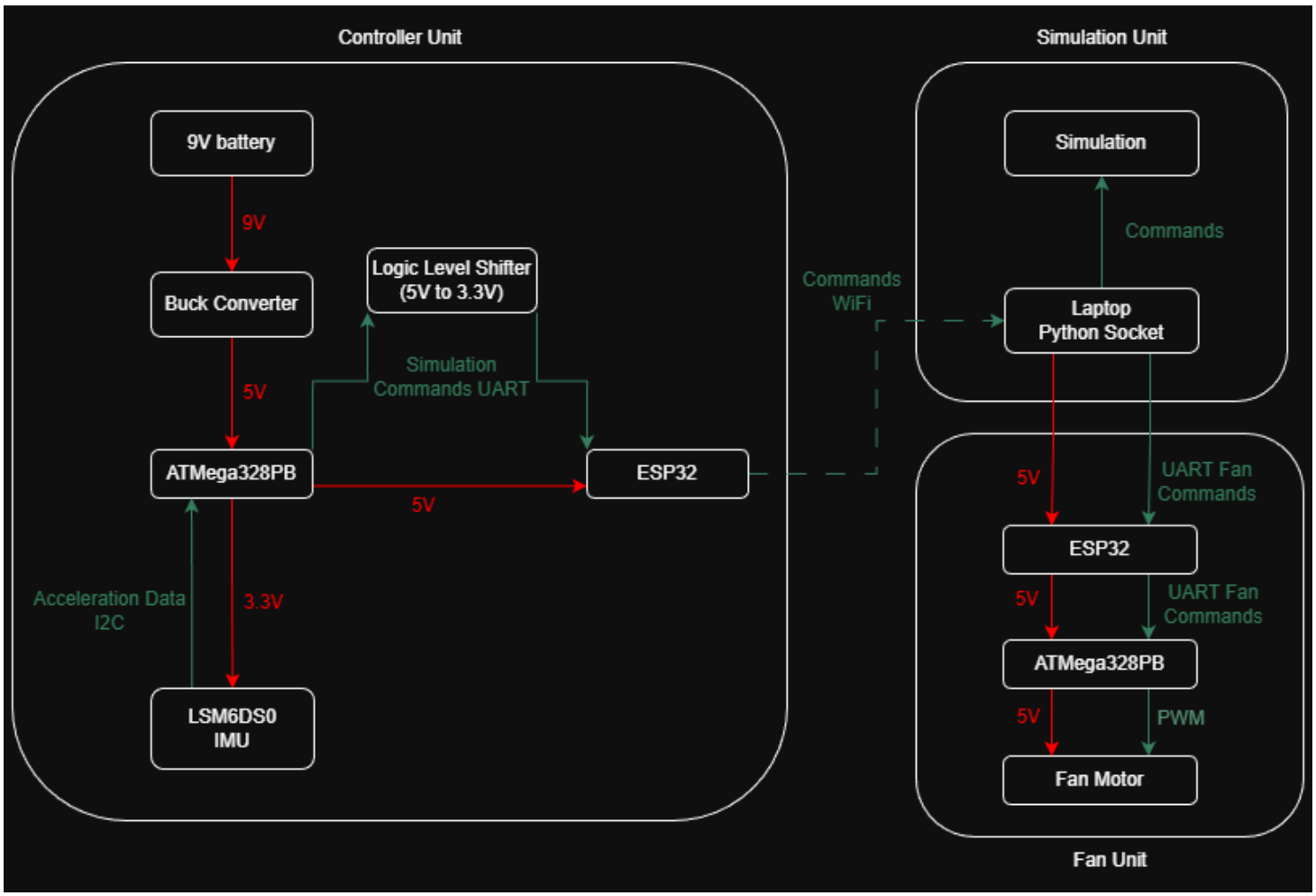



HAND

3. Results

For our final project, we were able to build a racing simulator controller as desired. We build one unit that attaches to the user's arm to mesure rotation (i.e. the steering wheel), and one unit that attaches to the user's leg to measure forward/backward acceleration (i.e. the gas pedal, with the functionality to reverse). We filtered noise from imu data to generate clear signals from which we could convert the user's movement to simulation commands. We were able to send these commands to a Python socket which used them to control the driving simulation, as desired.

With a bit of extra time, we were able to go beyond this initial goal to also control a fan that would blow air at the user if they were driving fast, to make the simulation more realistic. This unit recieved the speed from the python socket and converted that data into a PWM signal from an ATMega to control the motor. See our final system diagram below.

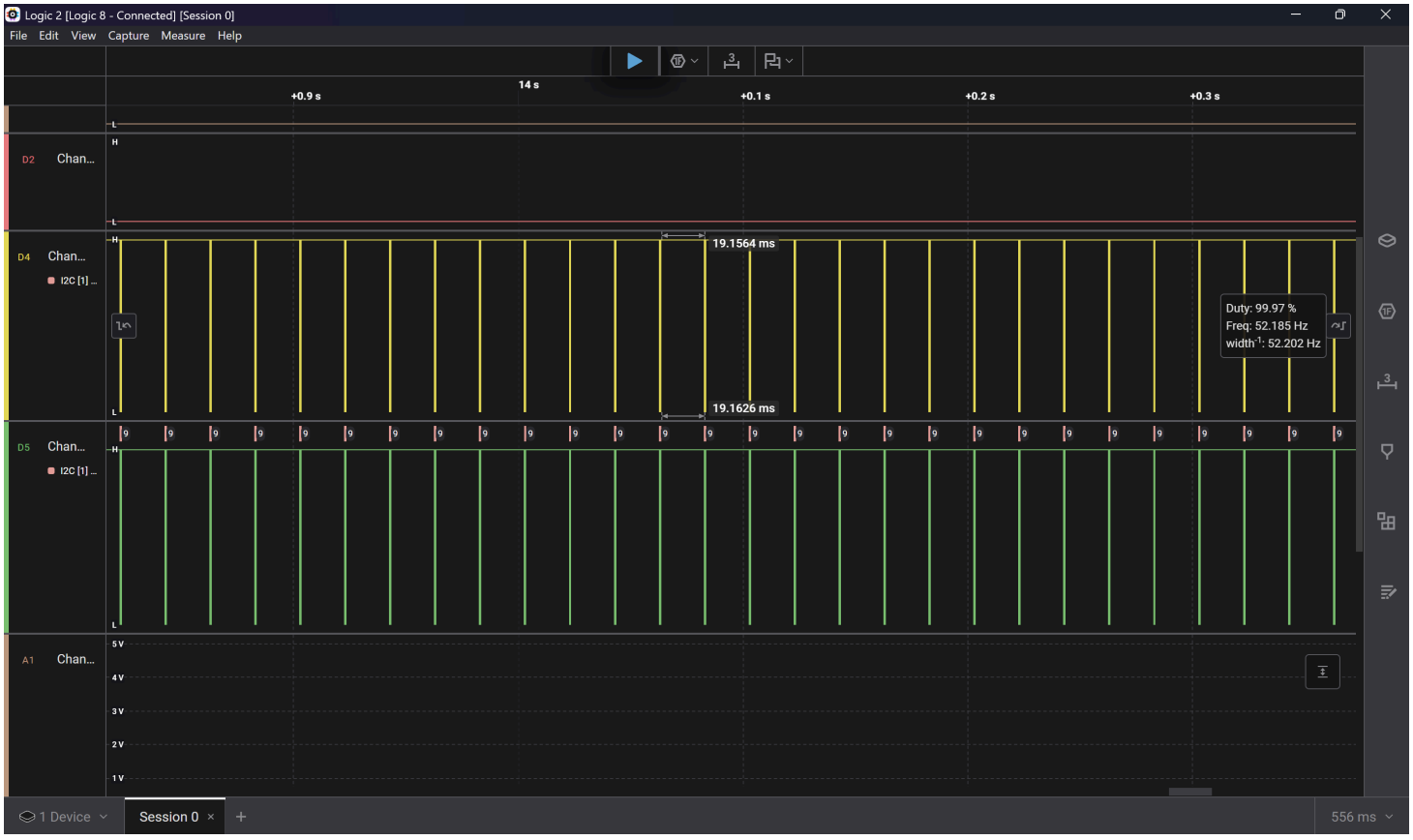


3.1 Software Requirements

ID	Description
SRS-01	The IMU 3-axis acceleration will be measured with 16-bit depth every 100 milliseconds +/-10 milliseconds.
SRS-02	ATmega328PB should be able to process IMU data and send info to ESP in 200ms.
SRS-03	ESP32 should be able to send control data through wifi/bluetooth to the computer within 500ms.
SRS-04	Python socket receives and uses user input within 200ms.

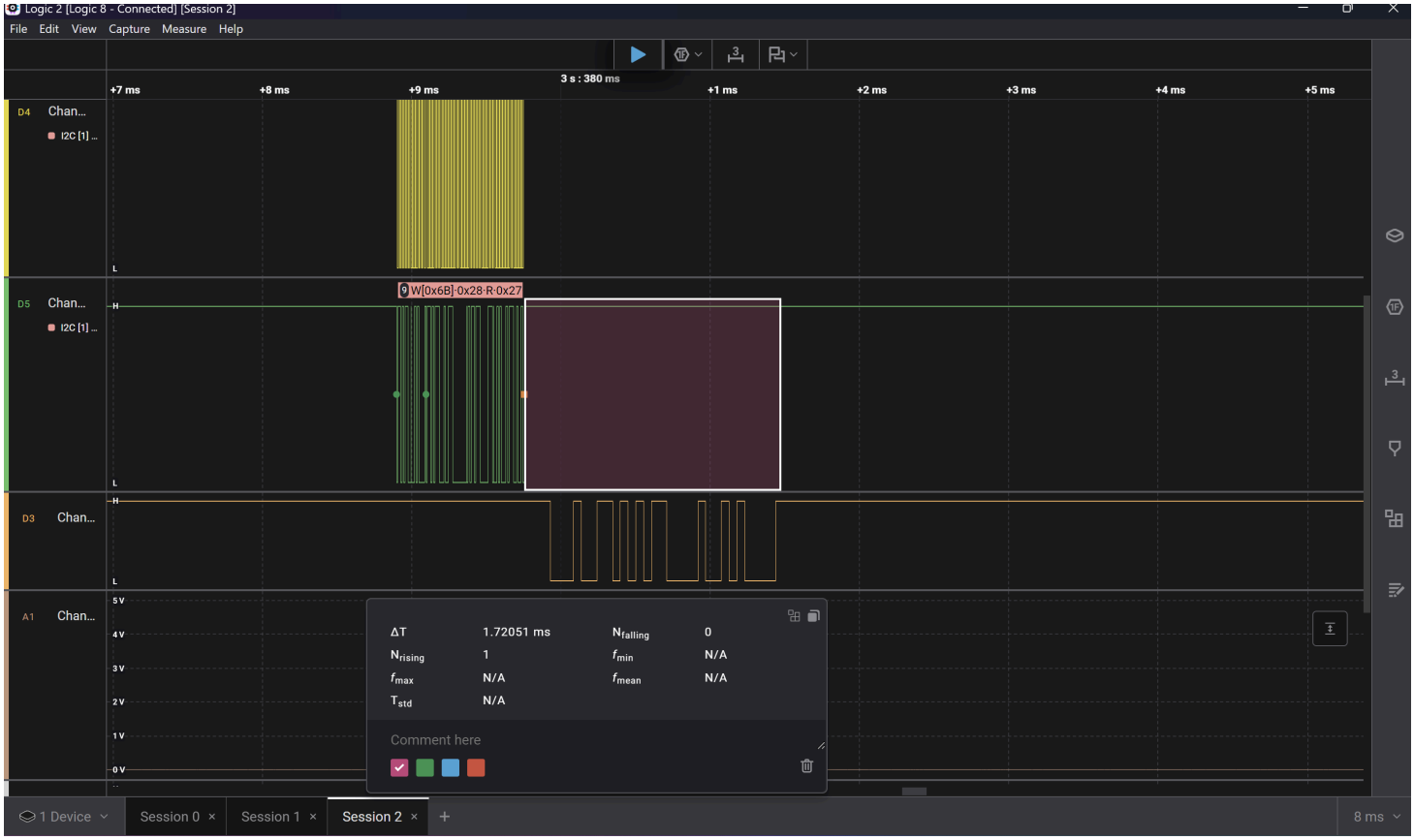
SRS-01: The IMU 3-axis acceleration will be measured with 16-bit depth every 100 milliseconds +/-10 milliseconds.

Verified -- we used the logic analyzer to determine how frequently the ATmega was pinging the ESP for its acceleration data. We are currently measuring approximately every 20ms, which is much faster than required. See the screenshot from the logic software below.



SRS-02: ATmega328PB should be able to process IMU data and send info to ESP in 200ms.

Verified -- we used the logic analyzer to determine how long it took the ATmega to finish sending the UART message to the ESP32 after it finished reading the imu data over i2c. This measurement was around 1.75ms, which is much quicker than expected. See that mesurement below.



SRS-03: ESP32 should be able to send control data through wifi/bleutooth to the computer within 500ms.

Hard to be tested, but system was responsive to controls immediately after inputs.

SRS-04: Python socket receives and uses user input within 200ms.

Again, this was hard to test, but our final system very quickly responded to user movements, meaning the Python socket was operating at an acceptable timing rate.

3.2 Hardware Requirements

ID	Description
HRS-01	The ATmega328PB and ESP32 will be powered with a 9V battery and should last approximately 2 hours before the battery is depleted.

ID	Description
HRS-02	ATmega328PB should be able to process IMU data and send info to ESP in 200ms.
HRS-03	The driving controller system and the computer/simulation system will be physically separate (i.e. no cables connecting them).
HRS-04	The final system will require minimal amounts of fine motor skills to operate - the system should be operable by a user with arthritis

HRS-01: The ATmega328PB and ESP32 will be powered with a 9V battery and should last approximately 2 hours before the battery is depleted.

Verified -- We measured that the ATmega (which is powering the ESP32 and the imu) pulls a maximum of 131mA from the battery. On the datasheet, the battery has a rating of over 500mAh, which means one battery can power the controller system for well over 3 hours, exceeding our two hour requirement.

HRS-02: The IMU will capture the user's movements, which will be processed through the ATmega.

Verified -- As shown above, we used the logic analyzer to read the signals passed between the two. This can also be seen from the demo -- by moving the imu the controls of the simulation change.

HRS-03: The driving controller system and the computer/simulation system will be physically separate (i.e. no cables connecting them).

Verified -- This is shown in the pictures and the demo video. Full system was completely separate from the software running on the computer.

HRS-04: The final system will require minimal amounts of fine motor skills to operate – the system should be operable by a user with arthritis.

Verified -- This is shown in the demo video. You do not need to be able to grip anything or use your fingers. Everything is controlled by large natural movements such as the twisting of your arm or the extension of your legs.

4. Conclusion

- **What did you learn from it?**
  Through the completion of this project we learned how to take an idea for an embedded device and turn it into a fully functional device through rigorous design and prototyping. We also learned

important things to consider when building a prototype under a short timeline, such as setting milestones to ensure that we are on pace to complete the project and ordering spare parts ahead of time.

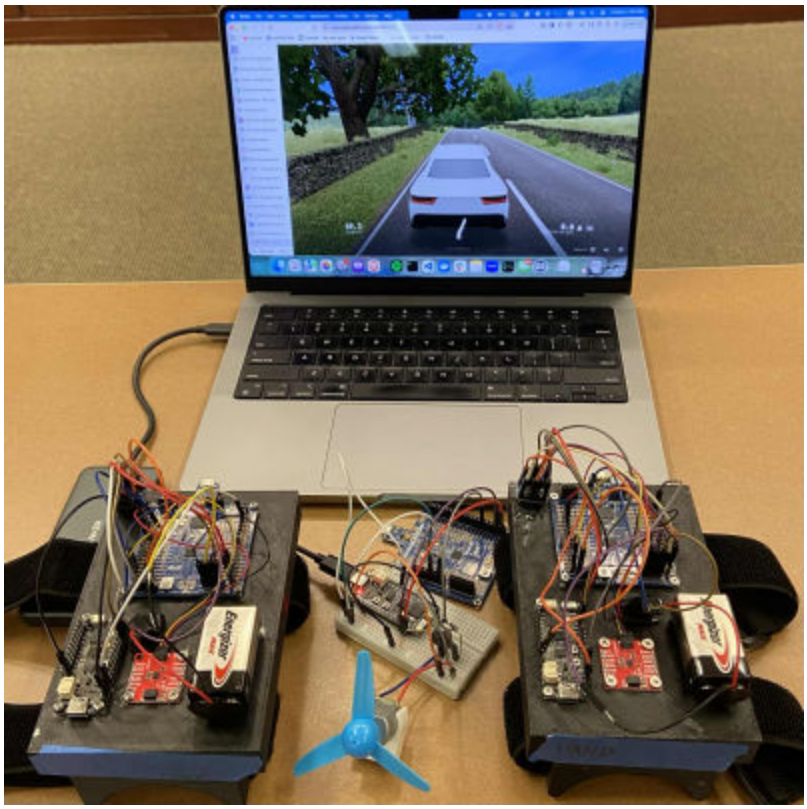
- **What went well?**
  Throughout the duration of the project, our team was able to come to a consensus on design ideas and implementation techniques efficiently, which allowed us to get a head start on the project. This proved to be crucial as the long lead times of ordering parts would have set our progress back significantly if we had not put in our orders far enough in advance. This gave us ample time to finish our prototype and test it thoroughly before demo day.
- **What accomplishments are you proud of?**
  Whenever we came across hurdles during the design phase and specifically encountering issues while building of the device, our team was able to work together to troubleshoot the problems and resolve them quickly. Given the short timeline to complete this project, we are very satisfied with the effectiveness of our final product given the complexity of the system.
- **What did you learn/gain from this experience?**
  This experience has showed us the difficulties of combining the different hardware and software components of an embedded device in order to implement it into a fully functional prototype. However, working through these difficulties under a tight schedule has prepared us for working through similar challenging situations when developing embedded devices in real world industries.
- **Did you have to change your approach?**
  The vision of our device changed slightly through our design iterations, but stayed mostly consistent as we had a clear idea of what our goals for it was from the beginning. Due to how the IMU sensors worked, we ended up having to separate the steering controller on the arm with the acceleration controller on the leg to make the device more intuitive to use.
- **What could have been done differently?**
  One component of our device that could have been improved was the hardware connections we used. Due to the tight schedule, we decided to use jumper wires for all of our power and data connections with only a few connections that were permanently soldered. This proved to be an issue as the connections would loosen and disconnect while using the device, especially since the operation of the controllers required significant physical movement. Having permanent connections between the components would further add to the robustness of our device and lead to less unexpected issues.
- **Did you encounter obstacles that you didn’t anticipate?**
  There were of course a lot of small bugs along the way that we had to sort out. But maybe our biggest challenge was figuring out how to mount all of our electronics on a controller that could be worn on the wrist. We didn't want it to be too heavy, but we did want it to be secure. And once we had designed the 3D printed part, we used a combination of threaded heat inserts to screw down the boards, velcro to attach the batteries, and tape to attach smaller components to the 3D printed

part. In software, our biggest obstacle became writing our own i2c driver for the ATmega to get data from the imu, though this was expected to be challenging.

- **What could be the next step for this project?**
  There are many different ways we could keep going with this project! One way would be to add more physical outputs, for example, a light that flashes when you drive off the road, or a speaker that plays different sounds depending on the terrain the user is driving over. Another route would be to make the system more accessible. As desired, one doesn't need fine motor skills to operate the controllers; however, it may be difficult for someone without fine motor skills to strap on the controllers, or open the simulation on their laptop. Thus, coming up with a more accessible way to attach the device to the user, or a standalone device to run the simulation, could be more accessible.

400 by 400 image

400 by 400 image:



Final Project Proposal

1. Abstract

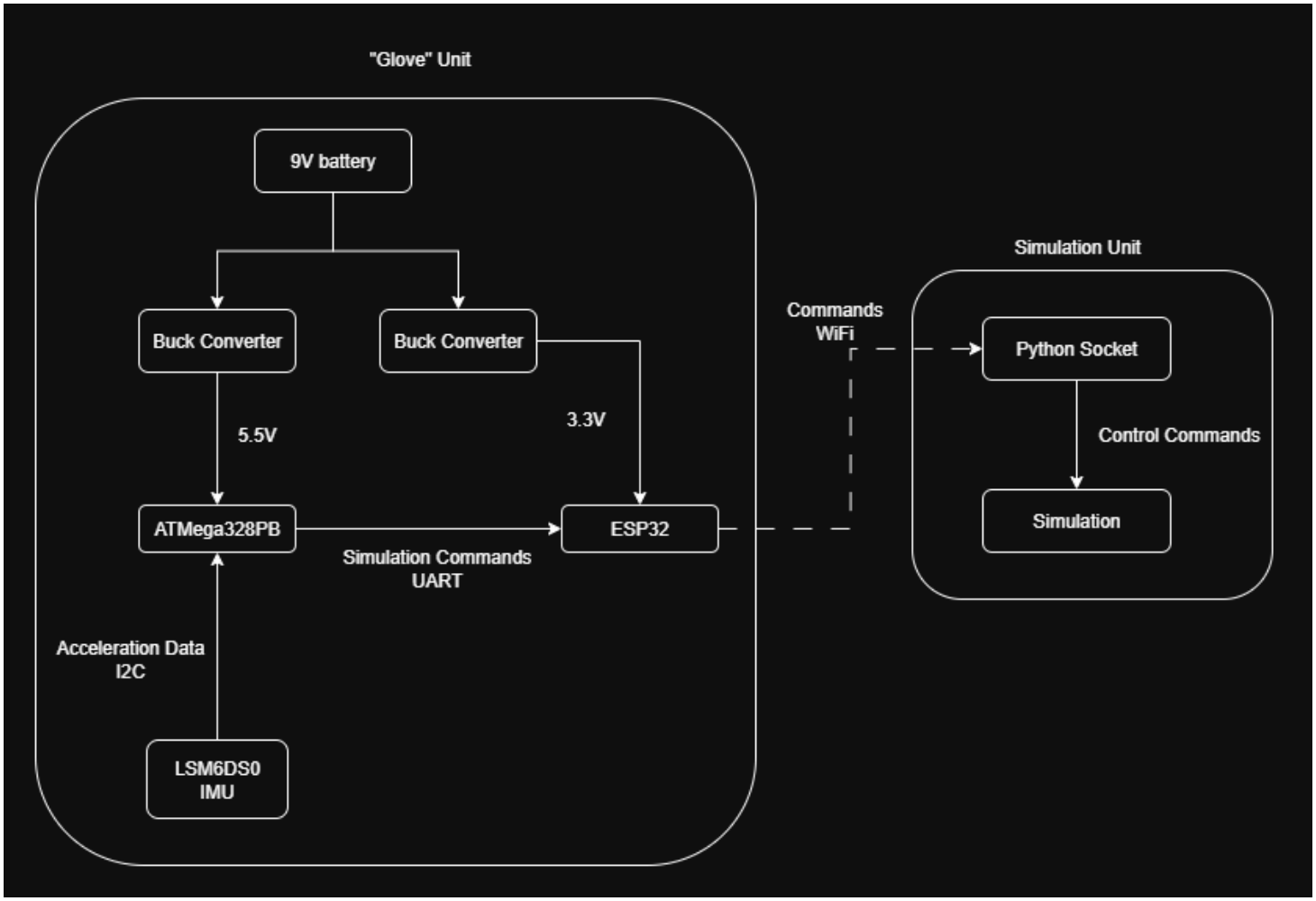
Our final project is a wireless controller based on IMU connected to the controller that acts as input for a game on your computer. Especially, the product is geared towards consumers where hand mobility

is an issue or the budget conscious consumers, as our device is low cost and designed for easy control even for the physically disabled. A lot of people with limited hand mobility or grip are not able to use a keyboard, a controller, or a racing wheel, especially where finger mobility due to conditions like arthritis is present. With our imu simulated driving controller, you are able to use it just like a wheel, without the need of grip on a physical object. There have previously been controllers made for such cases for people with limited finger mobility or amputated arms, but those are very expensive and specialized. With our low cost solution, we are able to hit a target market of both disabled personel and the budget conscious consumers together.

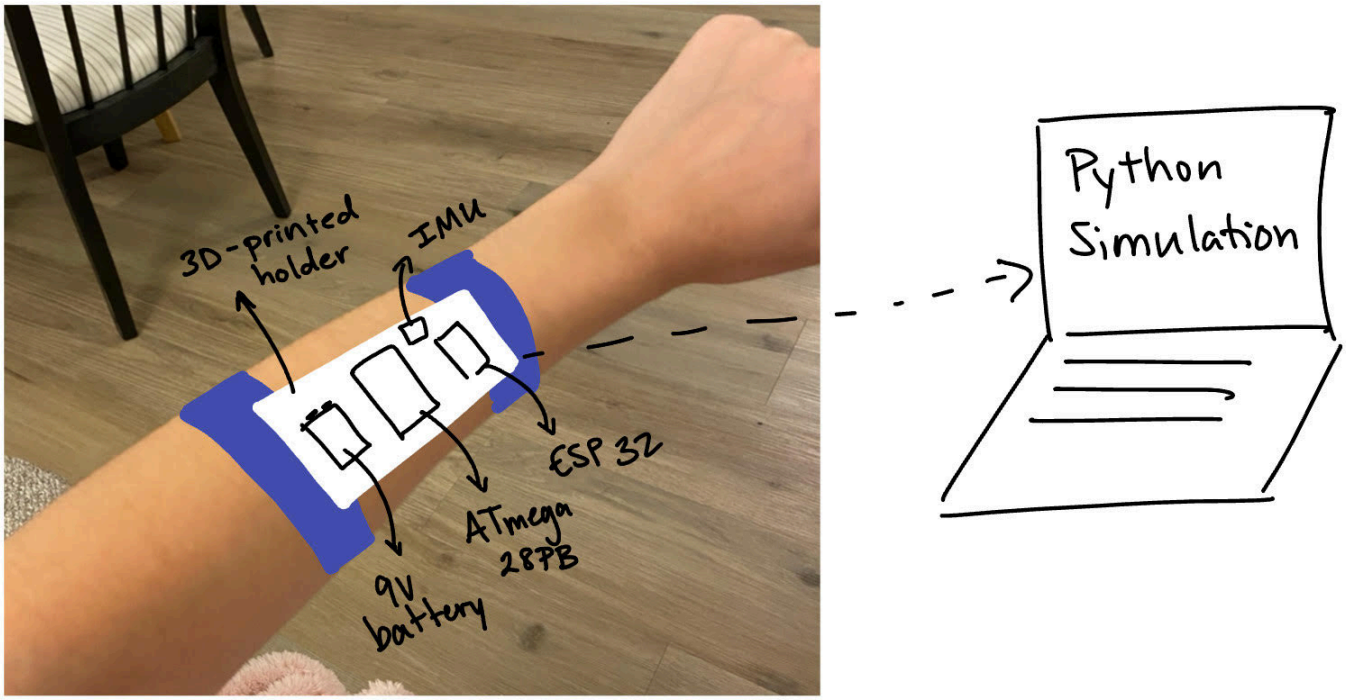
2. Motivation

We are trying to solve the issue of highly specialized, high-cost controller market for people with limited finger mobility. Being unable to enjoy playing games just because of a disability is very common due to the hefty cost of specialized controllers, and our project would act as a low cost and universal solution for these problems, whether the user has finger mobility issues or have amputated arms. We will be considering the use case of racing games, where disabilities might affect people to use controllers, keyboards, or racing wheels. Our IMU driving controller solves that issue in a very low cost manner, democractizing games for all.

3. System Block Diagram



4. Design Sketches



We will implement 3D printing to keep our hardware intact between the straps.

5. Software Requirements Specification (SRS)

5.1 Definitions, Abbreviations

Here, you will define any special terms, acronyms, or abbreviations you plan to use for hardware

5.2 Functionality

ID	Description
SRS-01	The IMU 3-axis acceleration will be measured with 16-bit depth every 100 milliseconds +/-10 milliseconds.
SRS-02	ATmega328PB should be able to process IMU data and send info to ESP in 200ms.
SRS-03	ESP32 should be able to send control data through wifi/bleutooth to the computer within 500ms.
SRS-04	Python socket receives and uses user input within 200ms.

6. Hardware Requirements Specification (HRS)

6.1 Definitions, Abbreviations

Here, you will define any special terms, acronyms, or abbreviations you plan to use for hardware

6.2 Functionality

ID	Description
HRS-01	The ATmega328PB and ESP32 will be powered with a 9V battery and should last approximately 2 hours before the battery is depleted.
HRS-02	The IMU will capture the user’s movements, which will be processed through the ATmega.
HRS-03	The driving controller system and the computer/simulation system will be physically separate (i.e. no cables connecting them).
HRS-04	The final system will require minimal amounts of fine motor skills to operate – the system should be operable by a user with arthritis.

7. Bill of Materials (BOM)

The following major components would be needed to create the device:

- ATmega328PB - This will act as the main processor to read and process data from the accelerometer sensor.
- Sparkfun LSM6DSO 6DoF Breakout IMU - This accelerometer will be the sensor used to determine the user's input movements in 3-axis, which will then be used to control the controls of the driving game or simulation.
- ESP32-S2 Feather Development Board - This co-processor will be used to wirelessly communicate the user's input control movements from the ATmega328PB to the computer to control the driving game or simulation.
- 9V Battery - This battery will be the main source of power for the processors and peripherals on the device
- Buck Switching Regulator IC - This buck converter will be used to step down the voltage of from the 9V battery to the 5V input ATmega328PB as well as the 3.3V input ESP32
  Bill of Materials

8. Final Demo Goals

To demonstrate our final project, we will strap the device onto a user's arm at two points. We will have a laptop running the driving simulation at the same time. When the user raises their arm, or completes a certain set of actions, the wheel in the simulation should move accordingly. In this way, the user can see their actions with the driving controller controlling the simulation. There are no limitations on time or place where this could be run, as long as a laptop is accessible. The driving controller should be able to last through the length of the final demo.

9. Sprint Planning

Milestone	Functionality Achieved	Distribution of Work
Sprint #1	IMU sensor reading roll/pitch/yaw data, ESP32/AtMega328PB basic communication, Python socket receiving raw IMU data	Xiang (IMU Data), Emily (Atmega and ESP communication), Shawn (Python Socket)
Sprint #2	IMU rotation mapped to steering input, controller emulation working, calibration for neutral position, basic filtering/smoothing	Xiang (rotation to steering mapping), Emily (controller emulation), Shawn (hardware implementation and wireless connection)
MVP Demo	Smooth steering control in at least one racing game, reliable connection, user can calibrate center position	Xiang (game integration testing), Emily (firmware optimization), Shawn (user calibration)
Final Demo	Additional feedback features like variable speed fan based on speed or another ankle controller for acceleration and breaking	Xiang (connection of new features to python socket), Emily (embedded software for feedback systems), Shawn (physical connection and embedded software for integration)

This is the end of the Project Proposal section. The remaining sections will be filled out based on the milestone schedule.

Sprint Review #1

Last week's progress

These are the tasks we worked on last week

(1) Setting up WiFi communication between the ESP32 and a Python socket. We wrote python code on the laptop and arduino code on the ESP32 and successfully recieved messages in a Python Socket from the ESP32.

(2) Setting up Python to control a driving simulation. Using a Python library we wrote a script that will control the keyboard (which will control the driving simulation) based on hard coded values.

(3) Modeling our wrist strap device holder module in CAD, and getting feedabck from Guanlin on the design, how each device will be mounted to the holder, and where we can source parts from protoyping labs.

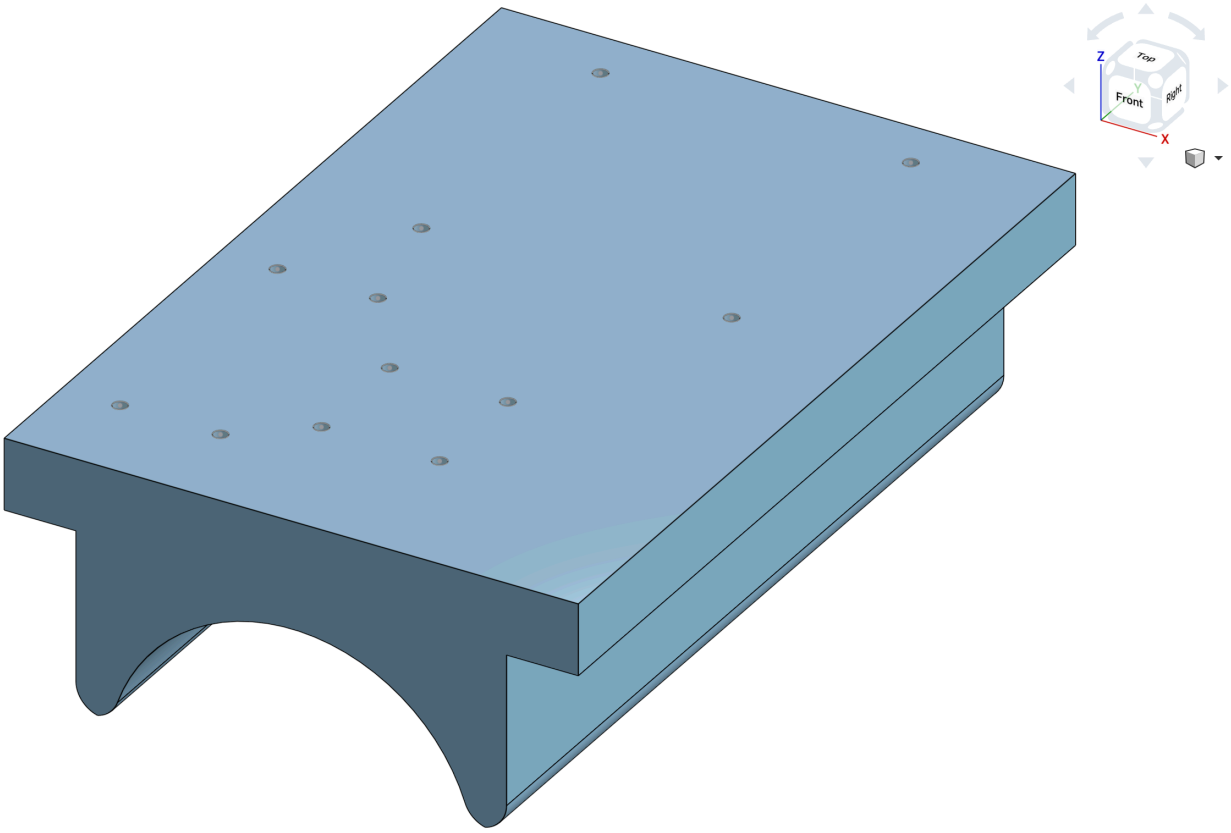
(4) Begin looking for a simulation. We've tried a few but want to continue looking for one that is "professional" enough for this context, but also has simple controls.

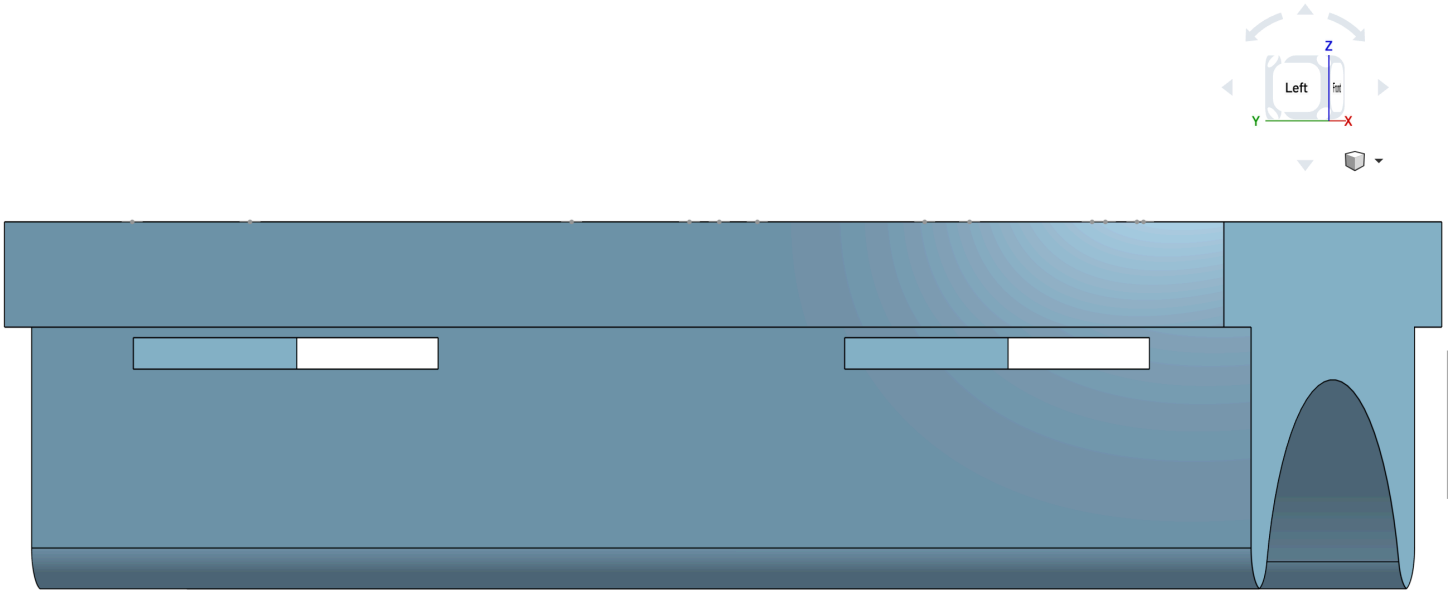
(5) Discussed how we want to communicate gestures. We decided on having the user pick a "zero" position, and calibrate that at the start of the game. Then gestures can be of the form "move right" and "return to zero." This will help us differentiate between a user moving from right to left, as opposed to moving from right to center.

(6) Finding buck converters and ordering parts.

Current state of project

We can send commands from the ESP32 to Python, and Python can send commands to a simulation, but those two haven't yet been integrated together. We have a CAD model of the wrist strap device holder module, and the rest of the parts we need have been ordered. We are able to read imu commands from the previous worksheet, but we haven't yet written our own i2c library.





Keyboard Control

Keyboard Control

ESP32 Socket

ESP32 Socket

Wifi Test

Wifi Test

Next week's plan

The following are our goals for next week:

(1) We just heard from Guanlin that we will have to rewrite the i2c library. So our focus for next week will be working on reading the i2c messages from the imu using our own library.

(2) While we attempted to start "gesture recgognition" from the imu this week, we didn't make a lot of progress, so we would like to continue working on this.

(3) We would like to 3D print our prototype of the wrist strap device holder module. This includes updating the design to use standoff screws as well as acquiring 3D printing lab access at the Venture Lab Prototyping Lab.

(4) We would like to integrate recieving controls from the ESP to controlling a simulation. For testing purposes, this can be hard-coded commands.

(5) Pick a simulation to use.

(6) Test sending messages from the ATmega to the ESP32. We origianlly decided to do that communciation over UART.

Sprint Review #2

Last week's progress

These are the tasks we worked on last week

(1) We wrote an i2c library for communication between the imu and the ATmega.

i2c.c

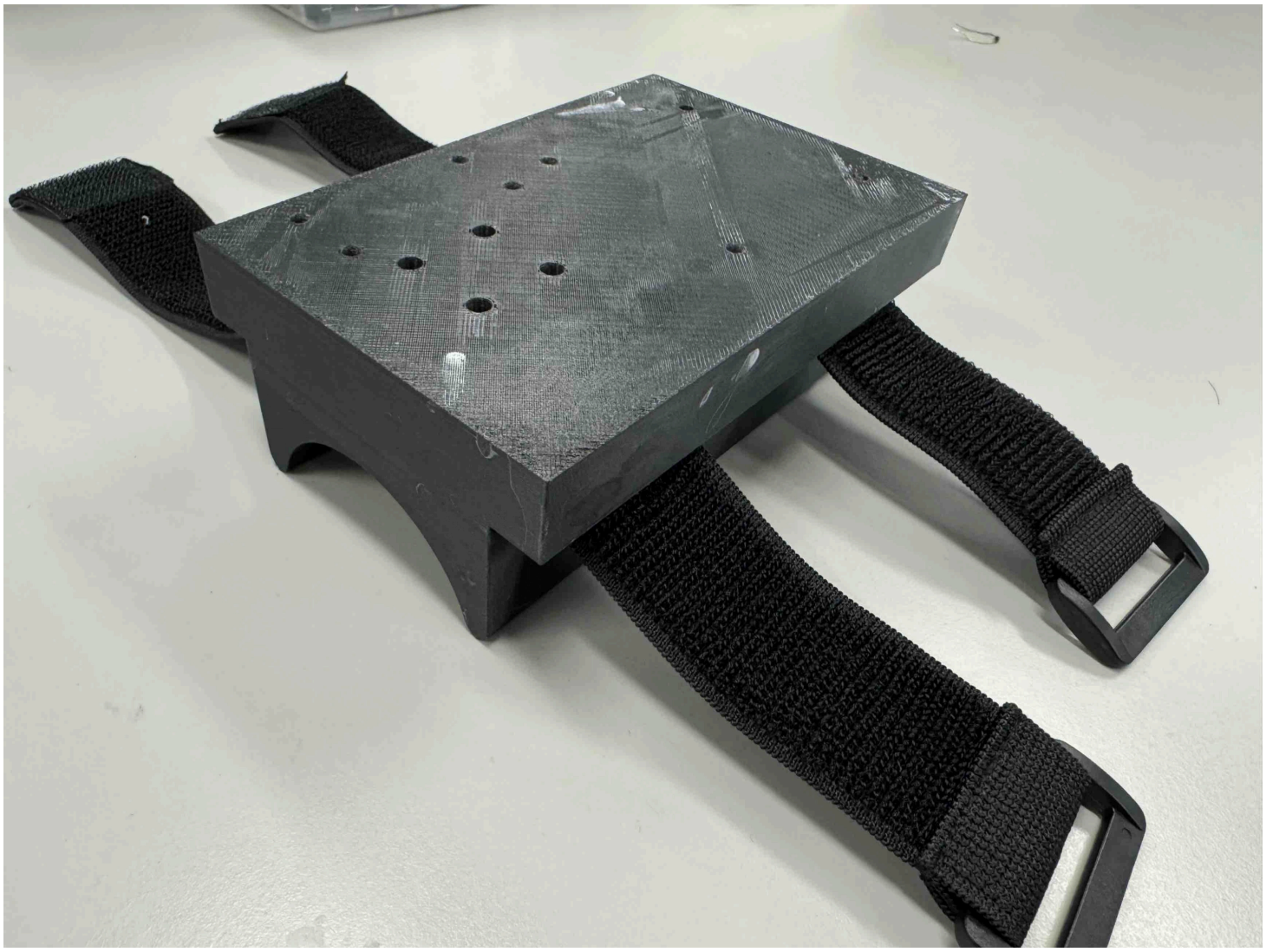
i2c.h

(2) We implemented a basic "gesture recognition." In this implementation we are using a temporal filter on the z-axis linear acceleration to recognize the user's hand as in one of three states. We implemented hysteresis to improve the state recognition.

main.c

(3) We updated our CAD model of the wrist strap device holder to use standoffs and heat set inserts with our parts. We also found the appropriate standoffs, heat set inserts, and screws in one of the campus maker spaces.

(4) We successfully 3D printed the wrist attachment module and have started assembling the components onto the 3D printed module.



(5) We integrated two python programs into one python script (wifi_to_control.py) that can now recieve controls over WiFi from the ESP32 and usse those controls to control a driving simulation.

wifi_to_control.py

(6) We decided on the simulation slowroads.io as it doesn't have obstacles or distractions in the way of the driving, and it is easy to control.

(7) We successfully powered the ATmega using a 9V battery and buck converter, and have begun testing powering the ESP32 with the 9V battery, though that needs more work.

(8) We attempted to communicate between the ATmega and the ESP32 over UART, though haven't been able to get that working yet.

(9) After consideration, we came to the conclusion that ADC will not be needed to process the data from the imu to the ATmega since the imu already outputs a digital signal. Therefore, with approval from our account manager, we have decided to remove all instances of ADC from our software and hardware requirement specifications.

Current state of project

We currently have all of the hardware ready to assemble for our MVP demo. The ATmega can be powered by the on-board 9V battery, while the ESP32 and the imu would be powered using a 5V output from the ATmega. In terms of software, the only missing link for the MVP demo is the UART communication between the ATMega and the ESP32. Other than that, we have the imu sending acceleration data to the ATmega, the ATmega turning that data into commands, and then the ESP32 able to send commands over WiFi to a python socket, and finally, a python socket able to control the chosen simulation.

Next week's plan

(1) Get the ESP32 and ATmega communicating over UART. This should close the software chain, meaning this software should be sufficient for the MVP demo.

(2) Connect and secure the power lines between the 9V battery, buck converter, ATmega, and ESP32. The current plan is to solder the connections together once the design has been fully finalized and tested.

(3) Complete the assembly of the device on the 3D printed holder. This means screwing in all of the components using standoffs, attaching the standoffs to the 3D printed holder, and ensuring the battery is securely mounted to the holder.

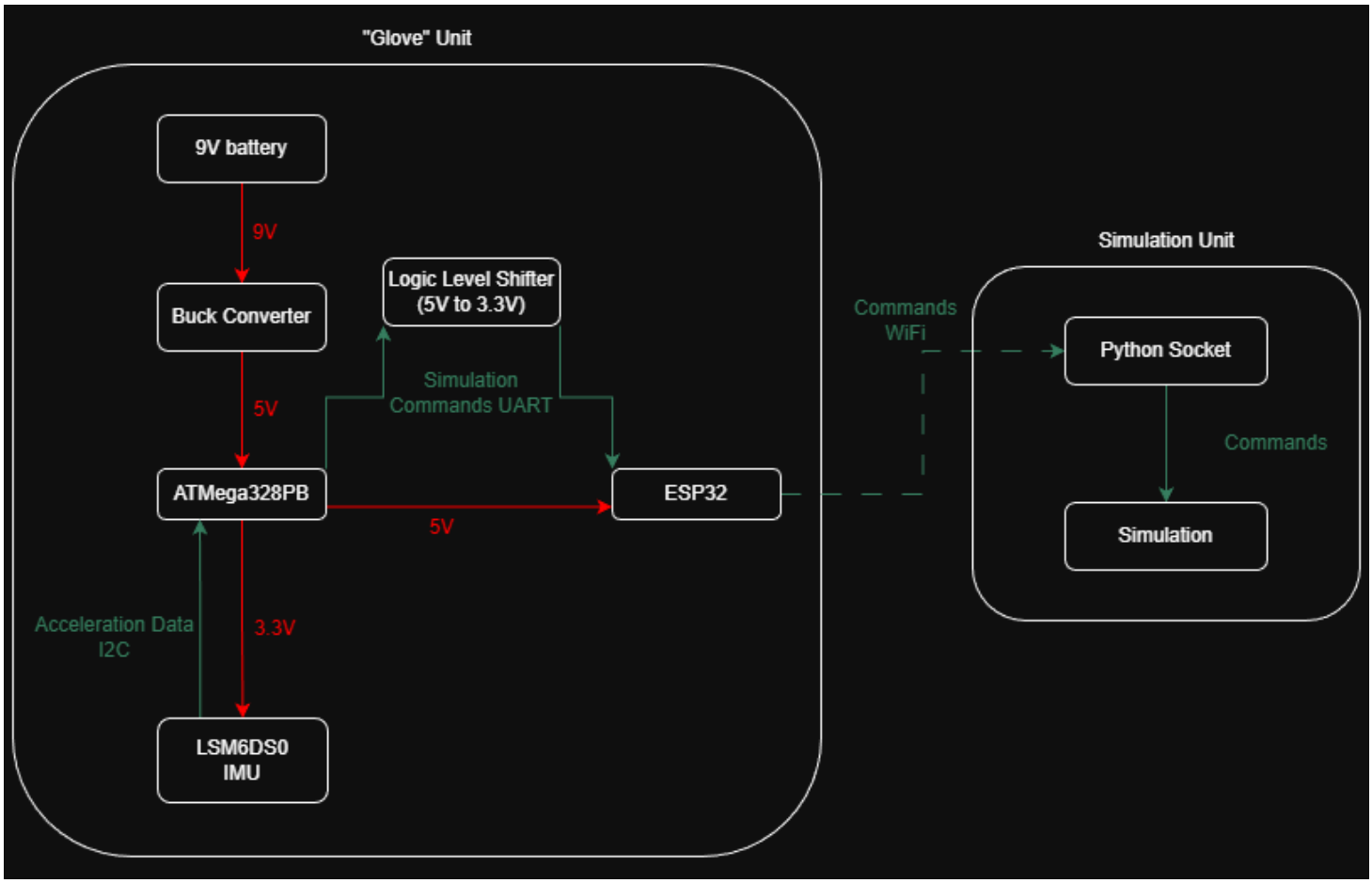
(4) Test the full system! In theory, we should be able to run a full game with the device at this point. Testing the system integration will reveal any mission critical faults, but also the response time of our controller, which we would like to measure and bound.

MVP Demo

Link to slide deck: https://docs.google.com/presentation/d/1Pktbb3_ehqotvovm03wZOGoHVvAa9-r1PvvPOWoFTHM/edit?usp=sharing

1. Show a system block diagram & explain the hardware implementation.

Updated system block diagram:



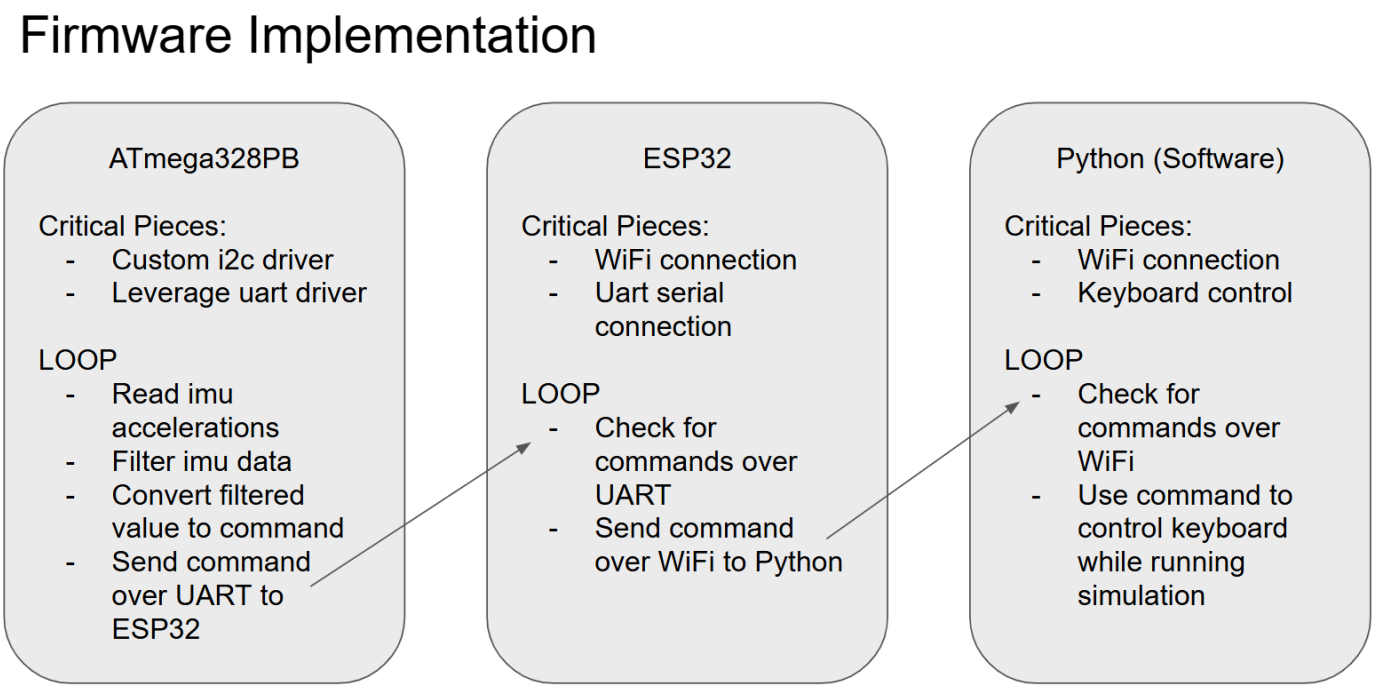
Hardware Description: See slides for images of the circuits and our 3D printed arm strap.

Data begins at the LSM6DS0 imu, which has 4 pinouts, GND, 3.3V, SCL, and SDA. These pare all connected to the ATmega, with SCL being connected to PC5 and SDA connected to PC4. The ATmega is powered by a buck converter from a 9V battery, so we first connect the battery to the buck, and then connect the output and ground to the ATmega's Vin and GND pins. To send data from the ATmega to the ESP32, we connect to a logic level shifter. We are sending data over UART, so we connect pins PD0 (Rx) and PD1 (Tx) to the two level shifter inputs. On the same side we connect to 5V from the ATmega, and pull the reset pin high (connect to 5V from the ATmega) to keep the level shifter working. The other side of the level shifter we connect to Rx and Tx on the ESP32, as well as

connecting to a 3.3V source and GND. We power the ESP32 with 5V from the ATmega. Finally, we connect all of the ground signals between the devices.

2. Explain your firmware implementation, including application logic and critical drivers you've written.

From slide 3:



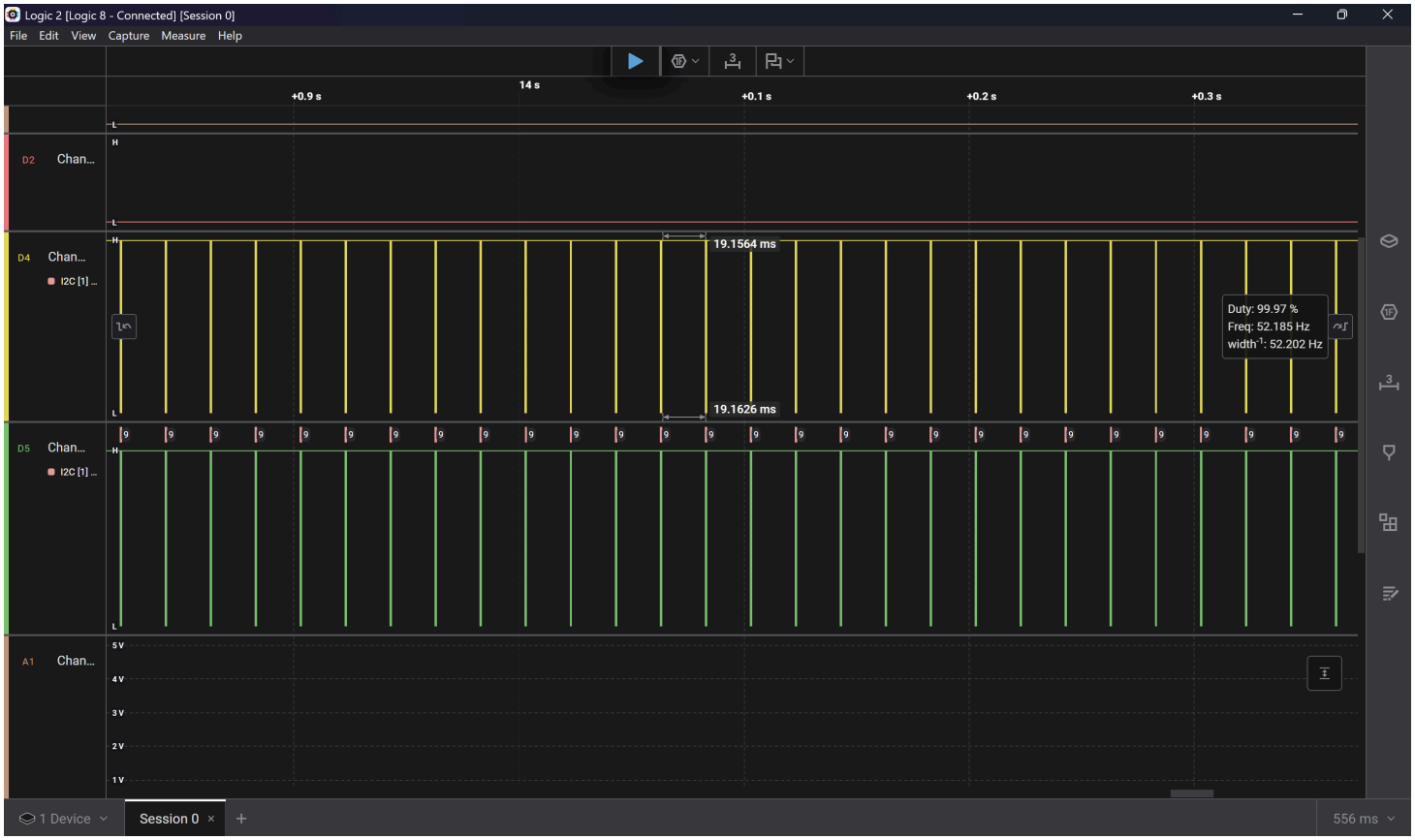
3. Demo your device.

Done in person

4. Have you achieved some or all of your Software Requirements Specification (SRS)?

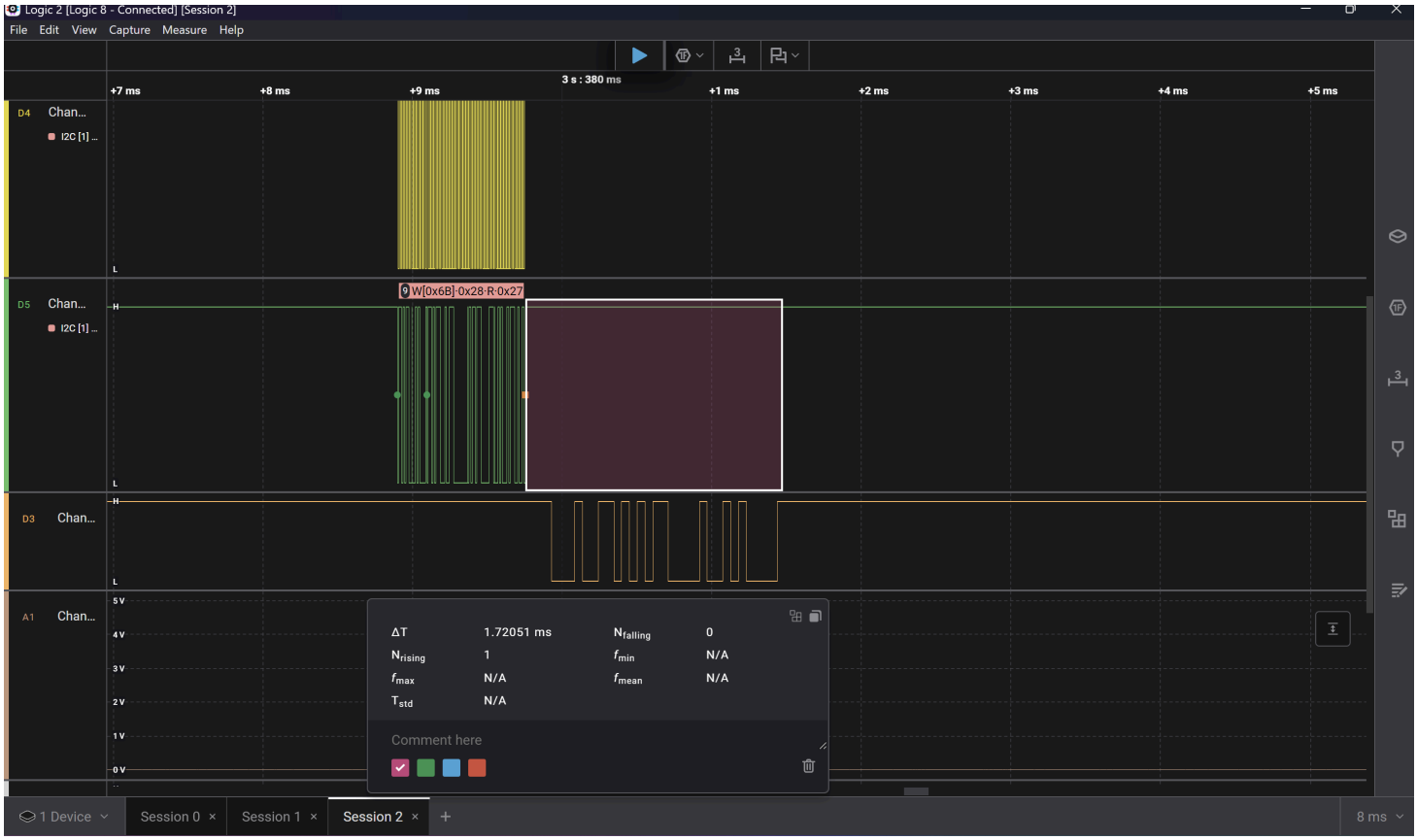
SRS-01: The IMU 3-axis acceleration will be measured with 16-bit depth every 100 milliseconds +/-10 milliseconds.

Verified -- we used the logic analyzer to determine how frequently the ATmega was pinging the ESP for its acceleration data. We are currently measuring approximately every 20ms, which is much faster than required. See the screenshot from the logic software below.



SRS-02: ATmega328PB should be able to process IMU data and send info to ESP in 200ms.

Verified -- we used the logic analyzer to determine how long it took the ATmega to finish sending the UART message to the ESP32 after it finished reading the imu data over i2c. This measurement was around 1.75ms, which is much quicker than expected. See that mesurement below.



SRS-03: ESP32 should be able to send control data through wifi/bleutooth to the computer within 500ms.

To be tested

SRS-04: Python socket receives and uses user input within 200ms.

To be tested

5. Have you achieved some or all of your Hardware Requirements Specification (HRS)?

HRS-01: The ATmega328PB and ESP32 will be powered with a 9V battery and should last approximately 2 hours before the battery is depleted.

While both boards can be powered by the battery, we have yet to test for how long. We are planning to measure the current of the battery once the system is fully implemented in order to calculate the theoretical amount of time the 9V battery will be able to supply that current for before being fully depleted.

HRS-02: The IMU will capture the user's movements, which will be processed through the ATmega.

Verified by using the logic analyzer to read the signals passed between the two, as shown above. This can also be seen from the demo -- by moving the imu the controls of the simulation change.

HRS-03: The driving controller system and the computer/simulation system will be physically separate (i.e. no cables connecting them).

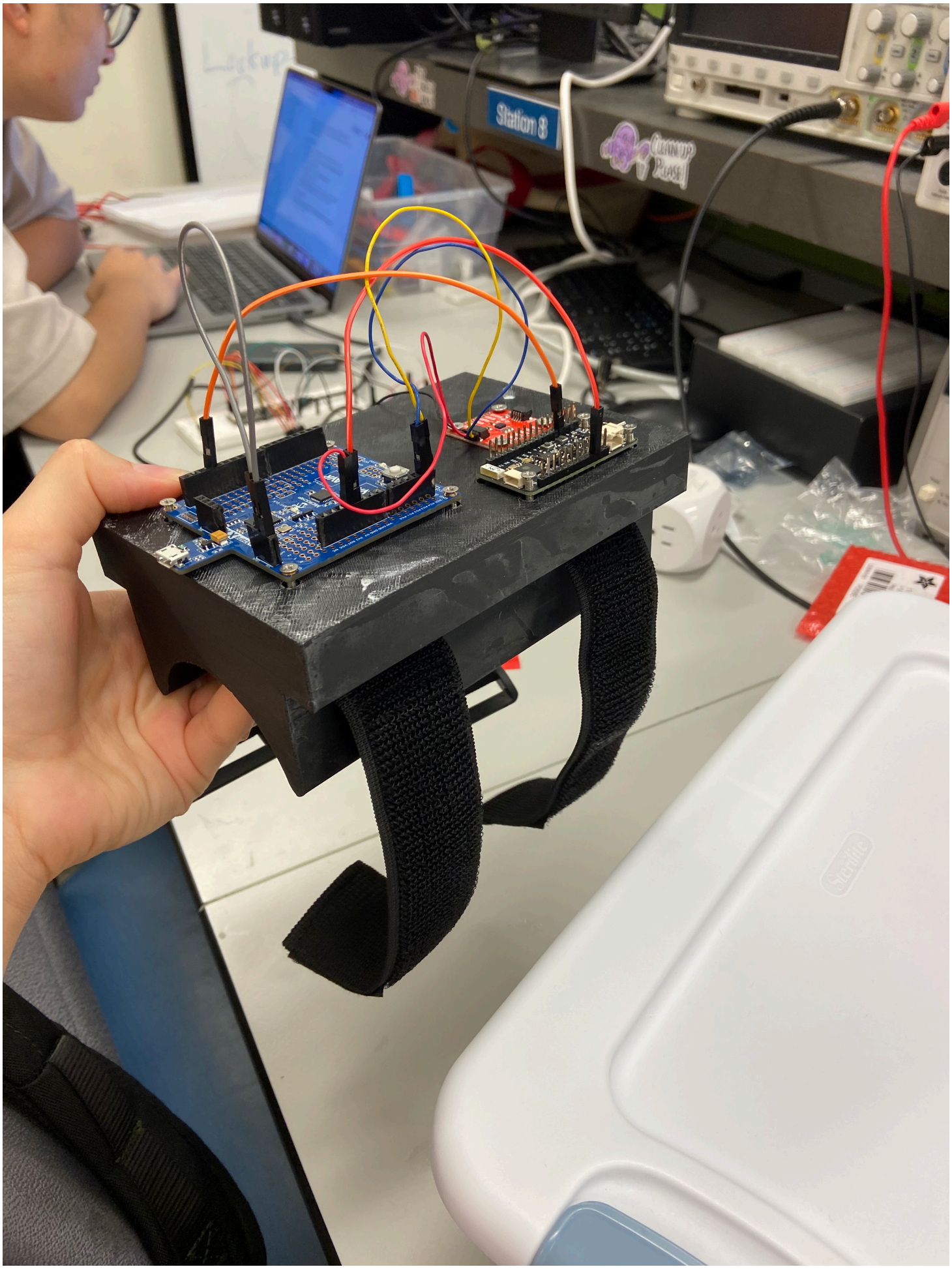
To be implemented

HRS-04: The final system will require minimal amounts of fine motor skills to operate – the system should be operable by a user with arthritis.

To be verified once the final system is complete

6. Show off the remaining elements that will make your project whole: mechanical casework, supporting graphical user interface (GUI), web portal, etc.

See our 3D printed device holder below


Station 3
CLEAN UP PLEASE!

7. What is the riskiest part remaining of your project?

The riskiest goal we have for the final product is to add a controller that attaches to the leg/foot to control acceleeration. We want to do this to make the simulation more realistic and include more functionality, but we need to ensure that adding the leg controller doesn't interfere with our arm controller. Notably, both we will need two ESPs communicating over WiFi with the Python socket, and we will need to ensure no messages get dropped and the messages are unique from each device.

We plan to de-risk this by having each ESP send tagged messages, meaning each message will come with an indicator of which device it came from. To ensure no message gets dropped, we are considering paralellizing our code, or adding messages into a queue so they get processed in the order they arrived.

8. What questions or help do you need from the teaching team?

We may need to discuss how to verify our software requirements and measure the speed of each step. But other than that, we are ready to move forward.

Final Project Report

See top of page 😀

References

We took advantage of many great Python packages in this project, as follows:

- socket -- We use socket to connect to our ESP32's over wifi.
- threading -- We use threading to multithread our code! Each ESP has its own Python thread that listens for data. And we have a control thread that acutally executes the simulation controls.
- pyautogui -- We use this to press and hold the keys on the keyboard to control the simulation.
- time -- We are using time to pulse keys on the kayboard. For example, if the user doesn't want to drive forward full throttle, we choose to pulse the forward key, and use time to keep track of those pulse times.
- serial -- We use serial to communicate with the ESP over the computer USB port to control the fan.