Year 2 Project

Wearable Sensor, or Digital Twin

Yixiao Shen (ID 201676945)

Jiajun Guo (ID 201676348)

Qi Zhou (ID 201677596)

Charles Canning (ID 201578458)

Jack Brissenden (ID 201587773)

Group 2p23

Supervised by Dr Dave McIntosh

March 25, 2023

Abstract

Previous research on avian flight dynamics has utilized optical motion capture systems for birds to collect their indoor wingbeat motion data for simulation and analysis. However, these systems are limited by site constraints. To overcome this limitation, recent studies have focused on installing single IMUs on birds to acquire basic wingbeat data without space limitations. Despite these efforts, there is still a significant research gap in developing a multi-IMU bio-logging device capable of comprehensively capturing the complete wingbeat motion of birds. To address this deficiency, this project aims to design a simple wearable sensor system based on four IMUs for human walking pattern recognition. Additionally, a front-end displaying engine based on an external Java graphic library was also developed to create a digital representation of humans, which enables a motion-following functionality. Although the technology will be initially tested on humans, the technology could then be applicable to tracking the motion and behaviour of animals in order to learn from the elegant designs seen in the natural world.

Declaration

I confirm that I have read and understood the University's definitions of plagiarism and collusion from the Code of Practice on Assessment. I confirm that I have neither committed plagiarism in the completion of this work nor have I colluded with any other party in the preparation and production of this work. The work presented here is my own and in my own words except where I have clearly indicated and acknowledged that I have quoted or used figures from published or unpublished sources (including the web). I understand the consequences of engaging in plagiarism and collusion as described in the Code of Practice on Assessment (Appendix L).

Contents

List of Figures

List of Codes

Chapter 1

Introduction

Avian flight dynamics have been a topic of interest for many researchers over the years. To study the indoor wingbeat action of birds, previous researches have utilized optical motion capture systems to collect data. For example, in order to replicate the motions of avian flight in a physical simulation, a group of researchers from Korea employed a marker-based optical motion capture system and high-speed cameras to gather flight data from pigeons, as reported by Ju et al. in 2011 [1]. Using the observed data, they were able to develop more realistic wingbeat movements.

However, such optical bio-logging systems have practical limitations, such as site constraints and high costs. Recent studies have attempted to address these challenges by attaching a single inertial measurement unit (IMU) to birds to collect basic wingbeat data. For instance, in 2018, Taylor et al. compared the flight behavior of pigeons flying alone to those flying in pairs, by attaching a global position system (GPS) tracker and a 200 Hz tri-axial accelerometer to each bird for data collection [2]. The researchers found that paired individuals exhibited a higher frequency of wing flaps, suggesting a potential improvement in maneuverability and flight stability.

Nevertheless, the information obtained from a single IMU is limited and may not provide a comprehensive representation of a bird's complete wingbeat movement. A fundamental research gap remains in the development of a multi-IMU bio-logging device capable of comprehensively capturing the entire wingbeat action of birds.

To fill such gap, this project aims to design a simple wearable sensor system

based on four IMUs for human walking pattern recognition, capturing comprehensive motion data that can be used for analysis and simulation purposes. Furthermore, a front-end displaying engine will also be created to generate a digital twin of humans in real time. This will not only enable motion tracking capabilities, but also allow for easy monitoring through a web browser with rapid deployment.

While the initial testing of the system will be conducted on human subjects, the potential applicability in tracking the movement and behavior of birds is promising. The data collected by this device would provide a valuable opportunity to gain insights from the intricate and sophisticated designs found in nature. For instance, one potential application is to study the flight patterns of birds and how they utilize their wings to achieve optimal efficiency, especially when faced with varying wind directions.

In conclusion, this project aims to develop a cost-effective and easy-to-use multi-IMU bio-logging device that can comprehensively capture the complete motion of objects. The proposed device has the potential to contribute not only to the research on avian flight dynamics but also to the broader field of zoology, which may serve as a useful tool for researchers in various fields.

1.1 Objectives

The primary goal of this research project is to create a digital twin of a human using wearable sensors. To achieve such target, the project has been divided into the following three sub-objectives:

1. Data collection: Design a wearable device capable of collecting lower body motion data by attaching four IMUs, two on each femur and tibia. To enable unrestricted movement without the need for cables, the device will feature wireless communication capabilities. In addition, the device will integrate an independent power source to ensure uninterrupted data collection during testing.

2. Data integration: A simple C++ socket server will be developed to serve as the central hub for collecting and processing the motion data collected by the IMUs. This server will also be responsible for transmitting files to the front-end.

3. Model creation: To enable the core functionality of motion-following, a simple front-end displaying engine will be developed to create a model of the human body. This model should be input with the motion data collected by the IMUs to replicate corresponding movements.

Chapter 2

Design and Implementation

2.1 Architecture design

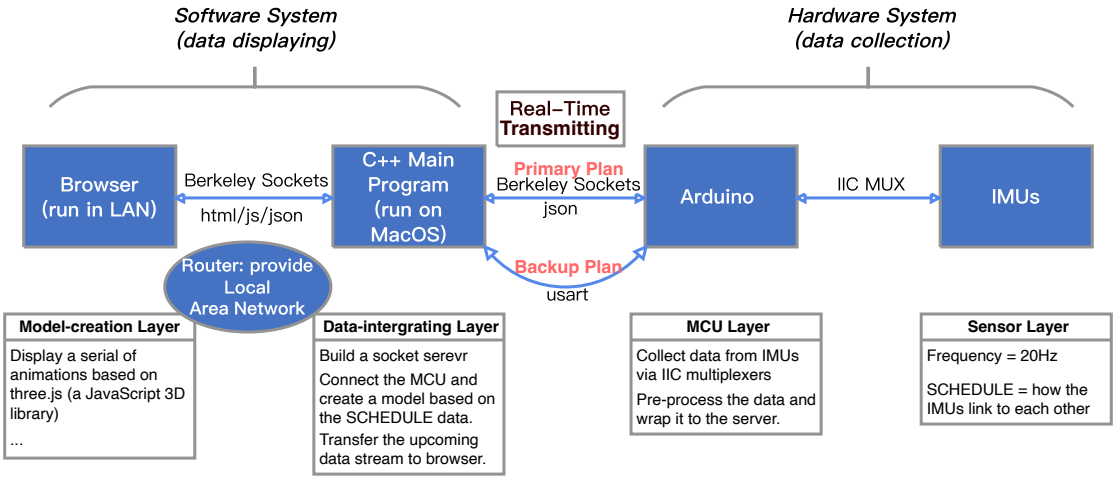


Figure 2.1: Project architecture diagram.

In order to ensure efficient task allocation, the project was partitioned into two modules as Figure 2.1 has shown: hardware and software. The hardware system was delegated to three MRS students, based on their respective areas of expertise and preferences. Their responsibilities encompassed the utilization of Arduino to collect data from multiple sensors through an inter-integrated circuit (IIC, or I2C) multiplexer, and an implementation of wireless transmission. Additionally, a backup plan utilizing wired communication through USART was selected, considering the development timeline and difficulty of using a Wi-Fi module.

On the other hand, the software module was entrusted to two CSEE students.

The responsibilities of the two CSEE students encompassed two tasks. The first one is to construct a simple C++ socket server to function as the data center. Secondly, they were tasked with developing a rudimentary human lower limb model using three.js, an external JavaScript graphic library [4].

Upon achieving these objectives, users would be able to access the visualized digital twin directly through a web browser with ease. To facilitate debugging, the entire network would operate within a local area network (LAN). However, it could be expanded to the Internet in the future.

2.2 Real-time rendering design

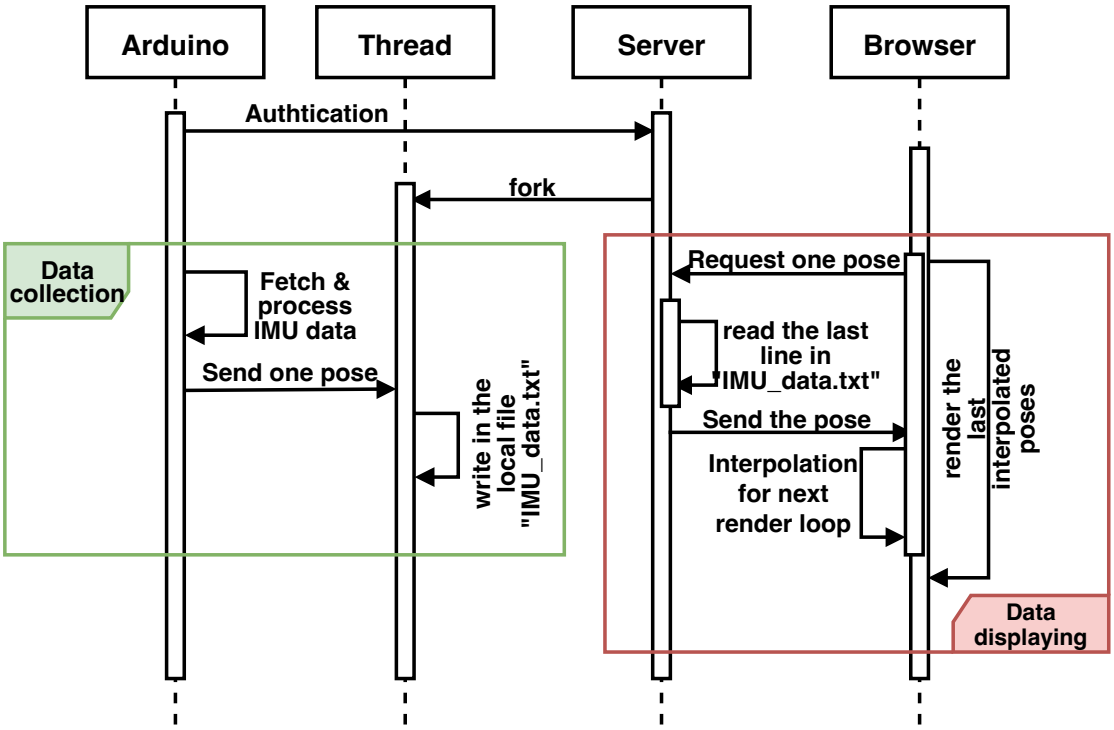


Figure 2.2: Transmitting logic of motion data collected by IMUs, where a single pose represents data from four IMUs, two each from the femur and tibia.

In this project, real-time rendering is designated as a potentially critical task metric. To achieve such functionality, two key steps are thus implemented. The first step is to ensure that the front-end could access the latest motion data. On the server, a thread is dedicated to continuously writing motion data from the IMU to the same text file. As the left hand side of Data-displaying loop in Figure 2.2 may illustrate, once the front-end browser request a pose data, another thread of

server will be launched to read the last line of the txt file and send it back to the client.

The second operation aims to reduce the latency in wireless communication . In the displaying engine, a method of multi-threading is employed. Specifically, one thread is designed to only render the animation, while another one is responsible for requesting data from the server and interpolating the returned data for use in the next rendering. This approach may not only smooth the animation but also offset some latency raised in the wireless communication.

2.3 3D model design

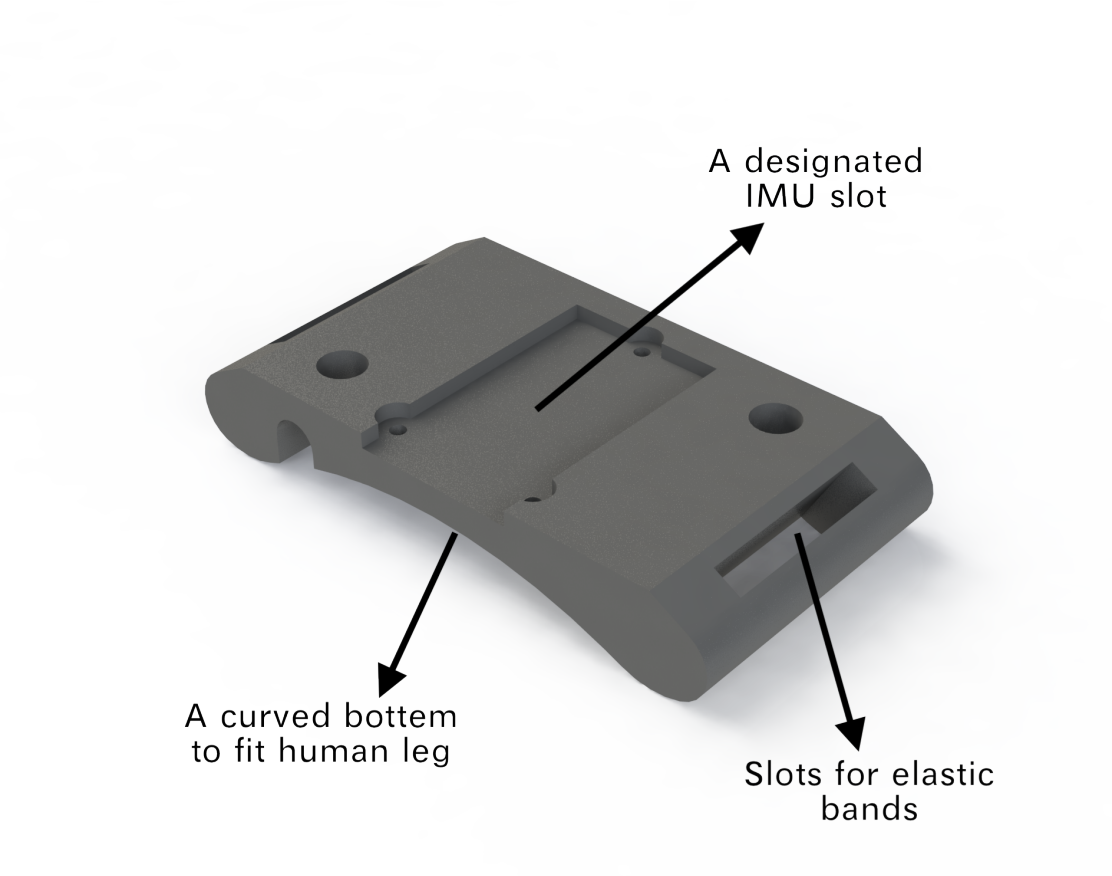


Figure 2.3: Model for holding an IMU on the femur (rendered by SolidWorks Software).

The IMU-holding component for the femur and tibia has been designed with a flat top edge and a curved bottom edge, as shown in Figure 2.3. It conforms precisely to the curve of the human leg, resulting in a more comfortable fit. To

minimize shock and vibration to IMU during testing, a form tape has been added into the designated IMU slot. Additionally, adjustable VELCRO straps have been incorporated into the component to accommodate different leg sizes, ensuring ease and speed of use.

Furthermore, two protective techniques have been applied for preventing damage to the data cable during testing. Firstly, a spiral wrap has been added to the data cable to provide additional protection. Finally, each component has been perforated with holes, through which a hemp rope will be threaded to prevent external forces from dislodging the cable.

Chapter 3

Materials and Methods

3.1 Materials

3.1.1 Main component list

1. Arduino-UNO-Wifi-Rev2: an upgraded version of the standard Arduino UNO board that incorporates several additional features to enhance its functionality. In this project, the onboard Wi-Fi module was fully used to provide a wireless communication.

2. Grove IMU 9DOF (ICM20600+AK09918): an inertial measurement unit module that integrates three sensors, including an accelerometer, gyroscope, and magnetometer. However, this module does not have a physical circuit of temperature compensation. Additionally, the chip ICM20600 only supports configuring two kinds of IIC addresses. Relevant datasheet is shown in Figure C.2 in Appendix.

3. Gravity IIC Multiplexer: a powerful module that allows for the expansion of the IIC bus on Arduino. It solves the problem of overlapping IIC address.

4. Multi-core screened cable: to ensure the quality of wired communication.

5. TP-Link AC1200 Wireless Dual Band Router: to provide a local area network.

6. 18650 2,600mAh Lithium Rechargeable Batteries: a built-in power source.

7. Velcro Brand Adjustable Straps, 25mm x 680mm.

3.1.2 Software

1. JetBrains CLion: for socket server programming;
2. Arduino IDE & PlatformIO: for Arduino programming;
3. Autodesk Fusion 360 & SolidWorks & SketchUp: for creating 3D models;
4. Three.js: for front-end programming;

3.2 Methods

3.2.1 Arduino: data collection

1. Firstly, connect the IIC multiplexer's VCC, GND, SDA, and SCL pins to the respective pins on the Arduino. All the cables used were initially of the GROVE type, which were later replaced by screened cables.
2. Then, each IMU was connected to one of the IIC multiplexer channels. The port setting was saved into the code as portrayed in List 3.1.

Listing 3.1: Saving the port setting of four IMUs.

```
22 //port number
23 enum portConfiguration {
24       IMU_LEFT_FEMUR = 2,
25       IMU_RIGHT_FEMUR = 6,
26       IMU_LEFT_TIBIA = 5,
27       IMU_RIGHT_TIBIA = 7
28 };
```

3. After that, install relevant libraries to drive the hardware components, such as the "IMU-ICM20600.h" for IMU and "DFRobot_I2C_Multiplexer.h" for multiplexer. Due to some reading issues encountered while using the magnetic sensor, the current project only utilized a 6DOF transformation. The initialization logic of four IMUs is shown below, where the default data rate for IMU was set to be 100 Hz.

Listing 3.2: Initializing the multiplexer as well as four IMUs

```
30 I2CMulti.begin();    //init the multiplexer
```

```
31 for (int i = 0; i < 8; i++) {
32     if (i == IMU_LEFT_FEMUR
33         || i == IMU_RIGHT_FEMUR
34         || i == IMU_LEFT_TIBIA
35         || i == IMU_RIGHT_TIBIA) {
36         I2CMulti.selectPort(i);
37         IMU_external.initialize();  //init the IMU
38         IMU_external_data[i].filter.begin(100);
39         delay(100);
40     }
41 }
```

4. After initialization, a calibration process was followed to improve the accuracy and reliability of the measurements. It was achieved by averaging the output data in the first three seconds where the IMUs were motionless. Then, future output would be reduced by this average to reduce factors of potential manufacturing variations or sensor drift. Relevant code is included in List D.1 in Appendix.

5. To obtain the motion data, the attitude and heading reference system (AHRS) library named Mahony was utilized to generate Euler angles from raw data. It is a set of algorithms designed to estimate the attitude and orientation of objects in three-dimensional space using fusion algorithms such as Kalman filter, Madgwick filter, and Mahony filter. These tools enable fusing the raw data of IMUs to obtain more accurate, stable, and reliable attitude information. The Mahony library, which is a popular sensor fusion algorithm, was selected for the experiment due to its higher computational efficiency. Additionally, it contains a 6DOF transformation without using magnetic sensors.

3.2.2 Arduino: wireless communication

1. To enable the built-in Wi-Fi module on Arduino-UNO-Wifi-Rev2, an external library called WiFiNINA was chosen and implemented. This library, provided by the official Arduino website, facilitates communication between the microcontroller and the onboard wi-fi module through a serial peripheral interface (SPI). It also provides some simple examples for studying.

2. Initially, the Arduino was connected to a 2.4G network and attempted to

establish a connection with a remote server through a handshake process. These functions are all included in the library and have been implemented in the kernel.

3. Next, a simple const string message was written to one Arduino's socket and later sent by kernel. Given that the server was capable of printing any incoming message, the message was finally displayed on the server's console as intended.

4. Following this, a stress test was conducted to evaluate the system's capacity to send a high volume of data. This was accomplished by continuously sending an increasing number to the server, which was then printed by the server. Meanwhile, the Arduino would also print out the message sent through a serial communication to the computer. By such comparison, the latency caused in this process could be roughly estimated.

3.2.3 Socket server

1. Based on an example sketch from an online book [3], a simple C++ socket server was built to support responding basic HTTP-GET requests.

2. Additionally, a multi thread was added to asynchronously save the motion data sent from Arduino.

3. To enhance the transmission of motion data to front-end users, a custom JSON encoder was developed to package the raw motion data into a compact and efficient JSON format as List 3.3 shows. To increase communication rate, the content in such format has been extremely reduced by using abbreviations, where "L_F" represents the motion data of left femur. The Euler angles were also abbreviated to "y", "r", and "p". Relevant code of the JSON formatter is in List D.2 in Appendix.

Listing 3.3: JSON code sent to front-end client.

```
1 {
2         "L_F": [{
3                 "y": -0.01,
4                 "r": 1.82,
5                 "p": -0.52}],
6         "R_F": [{
```

```
7                                    ...
8  }
```

3.2.4 Displaying engine

With the tutorial given by official three.js website [4], a simple static 3D model generator was thus implemented. However, such basic generator was not flexible enough to rapidly create a specific model, such as humans or birds. Hence, a function that can generate models according to user customization was developed. It works like a text operator and user could change his text to modify a corresponding model subsequently. Relevant results are presented in the next chapter. In the meantime, the following are the main implementation of this displaying engine:

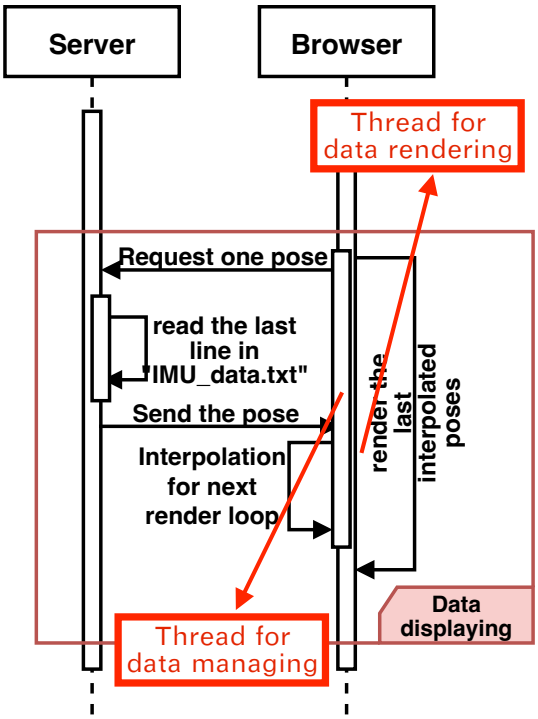


Figure 3.1: Sequence diagram of the displaying engine.

Reduce latency: multi-thread

To fully achieve the goal of real time rendering, as depicted by the sequence diagram of Figure 3.1, a multi-thread method was also developed. Particularly, one

thread is designed to render the animation, inputting motion data to the model. Simultaneously, another thread is responsible for managing motion data, requesting them from server and applying interpolation.

By achieving these, the latency raised by data fetching would be reduced to some extent. Relevant results are shown in the next chapter and core code is shown in List D.3 in Appendix.

Reduce latency: frame interpolation

To smooth the animation and increase frame rate, a frame interpolation function was also integrated in the thread of data managing. For example, it would linearly insert five frames into the original single frame. Relevant core code is shown in List D.4 in Appendix.

Data mapping

To account for the mismatch between the coordinate system used in the engine and the IMU coordinate system in the real world, a mathematical mapping was also implemented to convert IMU data into angle change values for the animation.

Specifically, for the femur's up and down movements in the engine, the corresponding pitch angle is given by $\pm(90^\circ + \text{IMU pitch angle})$, where the positive or negative sign depends on whether the current angle is lifting the leg forward or backward.

During testing, it was observed that the roll output for the femur IMU is usually around 0° when lifting the leg forward, and around 180° when lifting the leg backward.

Given that the yaw data was unstable and useless, the current engine could only achieve the motion follow in vertical plane. Therefore, a conditional statement using only roll data was added to the program to simply determine whether the leg is being lifted forward or backward.

Relevant core code is shown in List D.4 in Appendix, including the data matching for tibia IMU. The logic for tibia IMU is similar but only added its corresponding femur IMU data. This is because the tibia part in the engine was built from its femur.

Avoid jittering

During actual testing, it was found that the model would exhibit some jittering even when the human object was freezing. Therefore, a threshold limitation feature was added. Only when the original data's variation exceeds a certain value, the new motion data will be regarded as valid and later input into the model. Relevant code is shown in List D.4 in Appendix.

Chapter 4

Problem Identification and Resolution

4.1 Issue in network connection

During the connecting process listed in section 3.2.2, it was found that the router had been misconfigured prior to the test. As a result, the actual LAN was a 5G network, while the built-in Wi-Fi module only supports 2.4G [5]. This was revealed by running an example sketch that scans for all available networks that the Arduino can join. After that, the unsuccessful LAN was replaced with a cellular network provided by an iPhone, where the Arduino was able to recognize and connect.

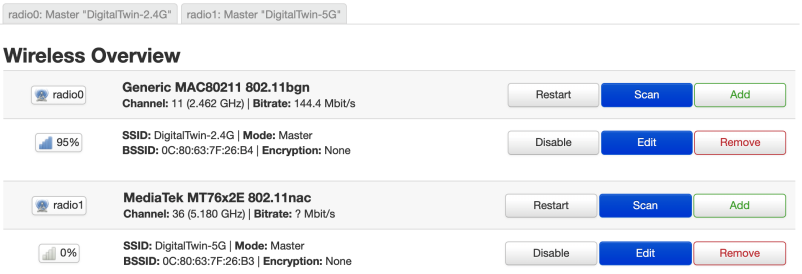


Figure 4.1: Revised configuration of router to provide a 2.4G network named "DigitalTwin-2.4G".

Such comparison highlighted that the module was working where the router configuration may be improper. By checking existed configurations and switching the 5G network to 2.4G shown in Figure 4.1, the network provided by the router was eventually recognized and connected by Arduino.

Wi-Fi	Bluetooth BR/EDR	Bluetooth Low Energy
IEEE 802.11b/g/n	Bluetooth v4.2 + EDR Maximum number of slaves: 7	Bluetooth 4.2 BLE dual-mode
Band support 2.4 GHz, channel 1-13*	Band support 2.4 GHz, 79 channels	Band support 2.4 GHz, 40 channels
Typical conducted output power 15 dBm	Typical conducted output power - 1 Mbit/s: 5 dBm - 2/3 Mbit/s: 5 dBm	Typical conducted output power 5 dBm
Typical radiated output power 18 dBm EIRP**	Typical radiated output power - 1 Mbit: 8 dBm EIRP** - 2/3 Mbit/s: 8 dBm EIRP**	Typical radiated output power 8 dBm EIRP**
Conducted sensitivity -96 dBm	Conducted sensitivity -88 dBm	Conducted sensitivity -88 dBm
Data rates: IEEE 802.11b: 1 / 2 / 5.5 / 11 Mbit/s IEEE 802.11g: 6 / 9 / 12 / 18 / 24 / 36 / 48 / 54 Mbit/s IEEE 802.11n: MCS 0-7, HT20 (6.5-72 Mbit/s)	Data rates: 1 / 2 / 3 Mbit/s	Data rates: 1 Mbit/s

* Depending on the location (country or region), channels 12-13 must be limited or disabled; the software implementation must support country determination algoritms for using channel 12-13, for example, with 802.11d. See section 6.1 for more info.

** RF power including maximum antenna gain (3 dBi).

Table 2: NINA-W10 series – Wi-Fi and Bluetooth characteristics

Figure 4.2: Datasheet of the onboard Wi-Fi module "NINA-W10", indicating that it only supports the 2.4GHz frequency band.

4.2 Issue of "no socket available"

4.2.1 Problem locating

During the code integration phase, a huge issue was encountered: the Arduino could only send motion data to the server for a very short time (from seconds to at most 3 minutes); then the built-in Wi-Fi module would throw an error called "no socket available".

Although the IMU module and wireless transmission module had passed their own stress tests, an additional stress test for the new framework was subsequently conducted, which involved sending const string messages. During the test, the Arduino was able to send a const char array continuously without any issues for half hour. Such stability indicated that the new code framework was not the cause of the problem.

With these three stress tests, this unstable output was eventually identified as "a function incompatibility between the IMU components and the onboard Wi-Fi

component".

4.2.2 Problem analyzing

Further researching revealed that even though the Wi-Fi module communicates with the microcontroller unit (MCU) using SPI protocol, it also has another channel for IIC communication as portrayed in Figure 4.3. Such "additional" IIC channel is then connected to the external IIC channel between IMUs and MCU. This may have implications for the result of "no socket available".

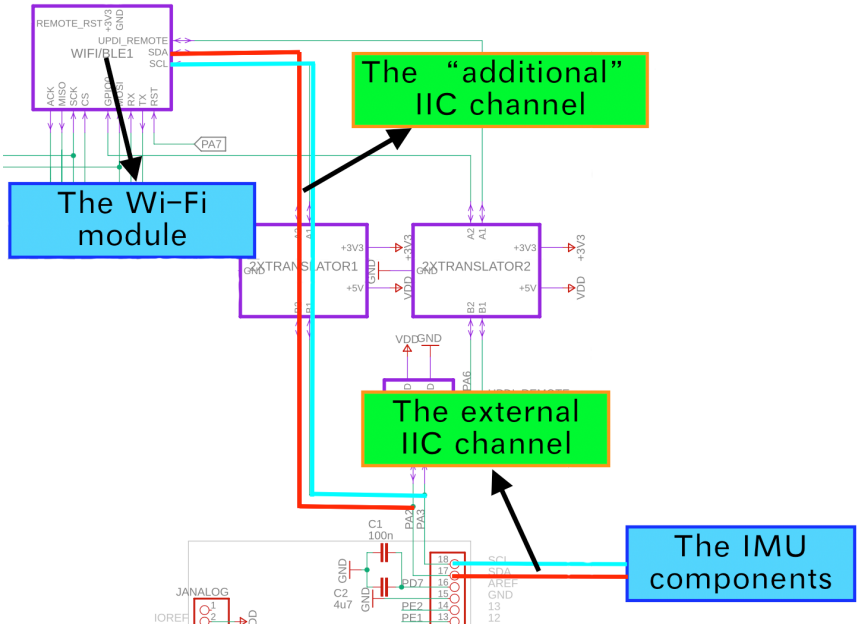


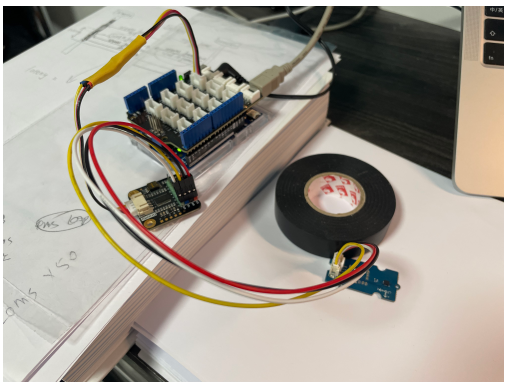
Figure 4.3: Partial schematic for the Arduino-UNO-Wifi-Rev2, where the MCU, Wi-Fi module, and IMUs are sharing a same IIC communication channel.

Based on such analysis, a hypothesis was formed that the external IIC might have an impact on the internal IIC, which could then result in a "no socket available" error in the Wi-Fi module. Additionally, during testing, it was noticed that shaking the IMU could subsequently trigger the error. It was speculated that the internal IIC requires a high level of communication quality. Hence, one possible solution is improving the poor cables captured by the image of Figure 4.4 (a).

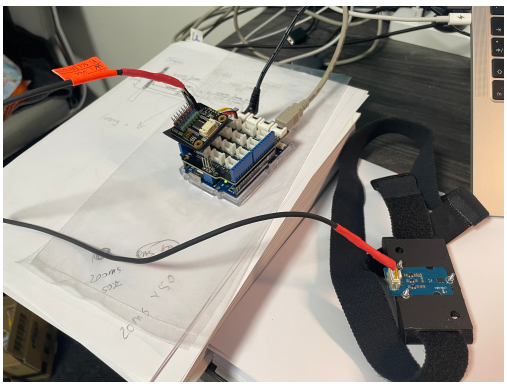
4.2.3 Problem solving

Improve IIC communication quality

To prevent potential problems raised by poor IIC communication quality, a multi-core screened cable was therefore applied, as Figure 4.4 (b) may show. Successfully, the Arduino could thus continuously send motion data to remote server.



(a) Original connection using dupont line.



(b) Improved connection using screened cable.

Figure 4.4: Two types of connections to build the external IIC channel.

Remove the "additional" IIC channel

During the following week of testing, the issue did not reoccur. However, after several accidental drops of the device, the problem resurfaced and the Arduino could not stably send motion data to server.

To solve the issue from root, the "additional" IIC channel was physically removed by desoldering two resistors on that channel as Figure 4.5 may show. As expected, the Wi-Fi module was indeed functioning well without any interference from external IIC channel. This success also proved the correctness of previous analysis.

In summary, the Wi-Fi module appears to be very vulnerable to any external interference in its IIC channel. If affected, it may stop working and throw a "no socket available" error. As the MCU and module use another SPI channel to communicate, we decided to physically remove that "additional" IIC channel since this project did not require an IIC functionality of the built-in Wi-Fi component. After the removal, the whole system could work perfectly. As for the root of such issue,

it may be the design flaw of Arduino-UNO-Wifi-Rev2, where the MCU, Wi-Fi module, and external IIC devices directly share one IIC channel. Simultaneously, the Wi-Fi module tend to require a high level of communication quality.

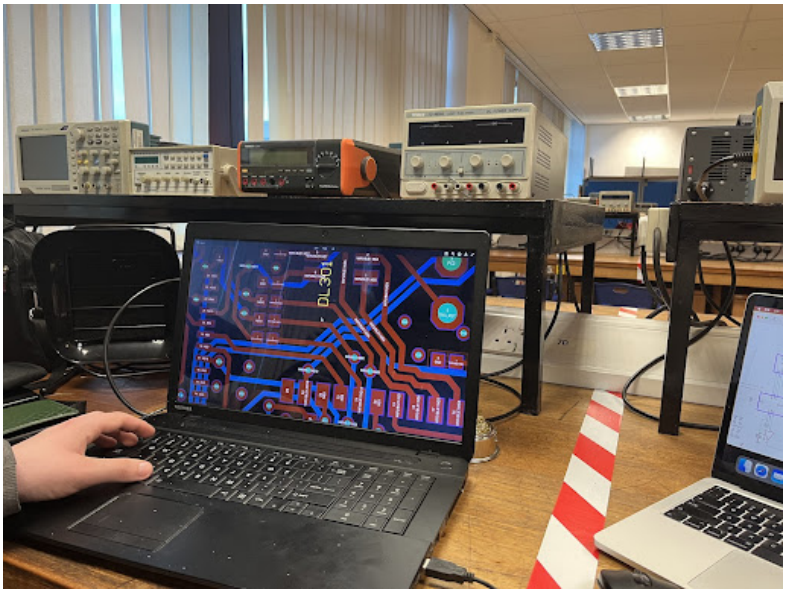


Figure 4.5: Desoldering two resistors on the "additional" IIC channel to completely solve the "no socket available" issue.

Chapter 5

Results

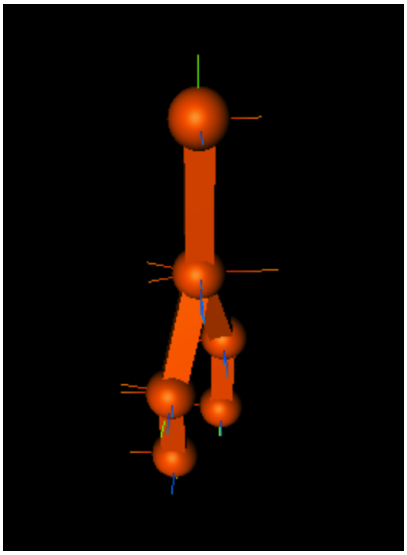
5.1 Arduino: Euler angle output

Based on the information presented in List D.5 (see Appendix) in the case of using four IMUs, the data collection interval of each round was about 55ms, nearly 20hz.

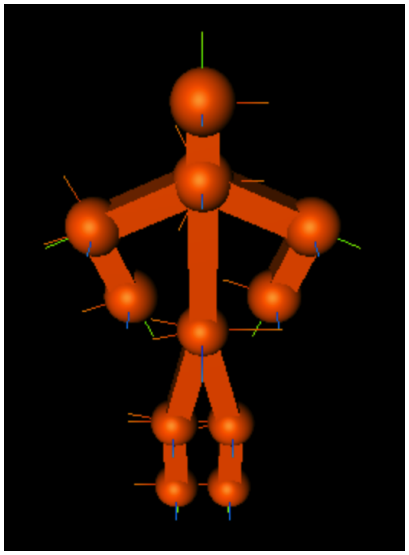
Meanwhile, it also turned out that the output of yaw axis data was unstable and inaccurate. Due to the time limitation, such data was discarded, leading to a loss of some functions for this project. As a replacement, the roll axis data was used to identify the pitch angle, which has been illustrated in the data mapping part of 3.2.4.

5.2 Displaying engine: flexible configuration

As depicted in Figure 5.1, with different text inputs, the displaying engine could generate corresponding models. The Figure (b) is based on the input of Figure (a) but adds other parts such as arms, forearms. Relevant configs are in List D.6 and List D.7.



(a) A human lower body model generated by the input text in List D.6.



(b) A full human body model generated by the input text in List D.7.

Figure 5.1: Corresponding models generated by various text inputs in the displaying engine.

5.3 Displaying engine: asynchronously fetching data

As highlighted by the representation of Figure 5.2, the thread for data fetching needed about 2 frame intervals to obtain the whole 0.73 MB data from the server.

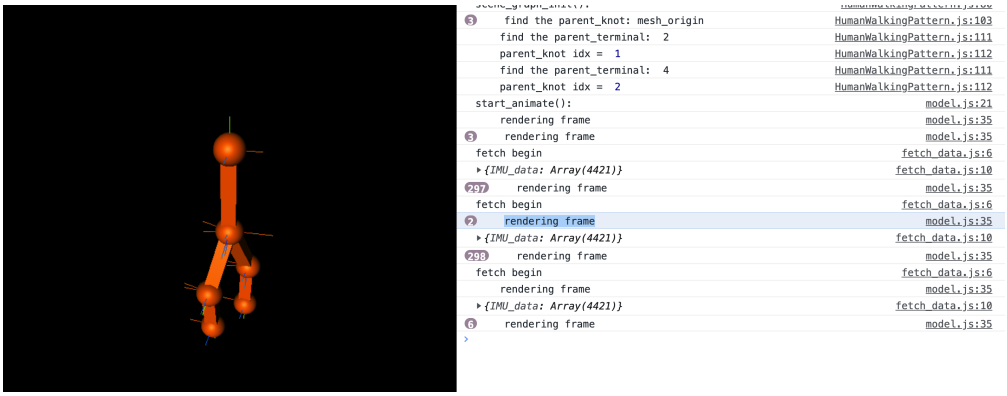


Figure 5.2: Time consuming while fetching a single string message with a size of 0.73 MB. The quantity of poses in such message is 4421.

By comparison, the thread needed 42 frame intervals to completely hold a 14.4 MB message.

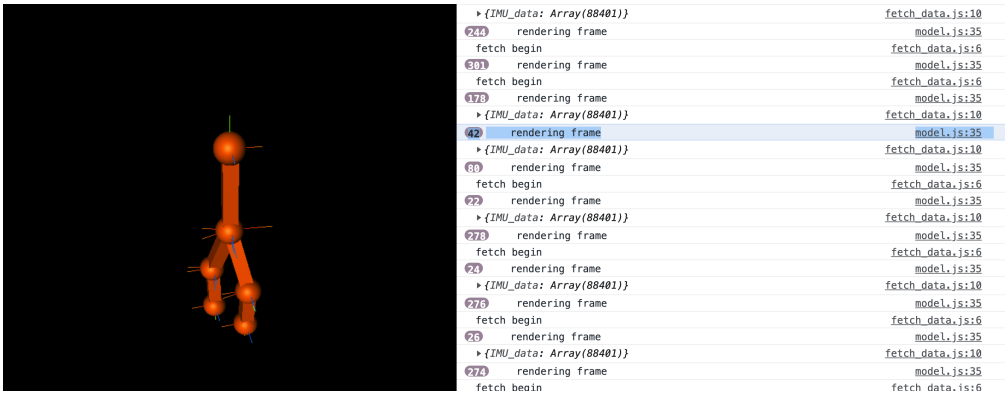


Figure 5.3: Time consuming while fetching a single string message with a size of 14.4 MB. The quantity of poses in such message is 88401.

5.4 Final Performance

Figure 5.4: A full testing with four IMUs and wireless communication.

As presented in the videos at [6], available via the QR codes in the footnote[1], the model generated by the displaying engine in the browser was capable of mimicking human leg movements such as lifting the leg forward or stretching it backward. As a result, the rotation of the leg could also be recognized and displayed, although it was not well demonstrated in the video. However, movements of extending the

[1] [6].

leg to the left or right could not be replicated to the digital twin due to the lack of yaw axis data.

After one-fold interpolation, the animation's rendering rate could reach up to 40 frames per second (FPS). However, due to potential factors such as communication delays in wireless transmission, the latency between the digital twin and the actual object is still within the range of human perception, approximately 100 ms by estimation.

In addition to the aforementioned results, the inclusion of the threshold operation has also partially reduced the jittering phenomenon in the animation. However, it was also found that as soon as the distance between the Arduino and the router exceeds two or three meters, there would be a significant increase in delay, which eventually leads to the displaying engine getting stuck.

Chapter 6

Discussion and Conclusions

6.1 Discussion

6.1.1 Arduino: Euler angle output

Data rate

The data collection frequency for a single round is only 20 Hz, which seems significantly different from the 100 Hz data rate set for the sensors. However, it should be noted that we have four sensors, and each sensor works at 100Hz to output data. In a single round, it would take the MCU longer to collect data from these four sensors, and there is also additional time required for the multiplexer to switch between the circuits. Therefore, the overall output data frequency for the entire pose, which includes data from all four IMUs, should be normal around 20 FPS.

In order to increase the sampling frequency, a simple approach is to attempt upgrading the default data rate of the sensors to 1 kHz. However, this places higher demands on the quality of external IIC communication. Although shielded wires have been employed to transmit data, the length of the wires has exceeded 50cm, which may already be at the limit for IIC communication. As the IMU sampling frequency increases, the IIC communication lines may become busier and more susceptible to minor external interference.

Loss of yaw axis data

Although there was some instability in the yaw data for some unknown reasons, we were still able to create a digital twin that is capable of replicating the motion of kicking forward and backward using the roll axis data to judge.

Future work should locate and solve this problem, where then the displaying engine could have a full functionality. If feasible, the sensor should also be upgraded to a more powerful one, which may include a temperature compensation circuit to avoid the effect from changing temperature.

6.1.2 Displaying engine: flexible configuration

The result in section 5.2 indicates that the current displaying engine is sufficient to meet various requirements of modelling, merely generating corresponding skeletons composed of various joints.

However, it is now still difficult to handle an incorrect input. Once the setting is wrong, the program will fail immediately, without giving any guidelines for the user to diagnose their wrong inputs. Hence, this could also be a future improvement.

6.1.3 Displaying engine: asynchronously fetching data

With the results in section 5.3, the operation of multi-thread should be particularly useful while sending a large message, at least containing 10,000 poses in single message. However, in current project, given that the pose data rate was only 20Hz, the quantity in a one message would not exceed 20 poses, or the latency will be at least 1 second. As a result, the current design of multi-thread has not fully realized its intended function as the size of message is small.

6.1.4 Final Performance

Latency

Despite efforts to reduce latency in the current real-time rendering process, such as using message abbreviations and implementing multi-threading for data sending

and receiving, further improvements can be made. One potential solution for the future is to conduct a thorough analysis of the latency from the Arduino to the display engine. This analysis can help identify the primary sources of delay, allowing for targeted improvements.

Adaptation

The current device is lightweight and suitable for use by humans, but it is not compatible with birds. Therefore, to collect motion data of birds in outdoor settings in the future, a complete redesign of the data collection layer is necessary. For instance, the smaller Arduino Nano Every board can be utilized, and new wearable solutions can be explored, taking inspiration from the structure of birds themselves. However, some other parts, such as the displaying engine, only require partial improvements as they have already been designed to generate corresponding models flexibly with input text.

Accuracy

Although accuracy was not listed as one of the main objectives for current project, it may be a key focus for future work, especially when it comes to precisely mirroring complex motions. This could become particularly important when the device is applied to study the flight dynamics of birds, where accurate data is essential.

For instance, in the case of studying bird flight, the device will need to capture and analyze a vast amount of data with high accuracy in order to gain insights into the complex mechanics of avian flight. By achieving a higher level of accuracy in motion tracking, researchers can better understand the nuances of bird flight, including the subtle changes in wing angle, speed, and other factors that can affect flight performance. This can ultimately lead to a better understanding of the biomechanics of avian flight and help researchers design better drones and other aircraft.

6.2 Conclusions

In summary, this research project managed to create a non-optical bio-logging system that utilizes four IMUs to capture motion data for human objects. Moreover,

the system includes wireless communication and a displaying engine that creates a digital twin of the human lower body in the browser. This digital representation can mimic real leg movements such as lifting the leg forward or stretching it backward, with an acceptable latency. Given the flexibility of the system, it also has the potential to be further enhanced to investigate avian flight dynamics or other zoology-related topics.

Acknowledgments

Special thanks to Aurora for the 3D printing and to Dave for his guidance throughout the entire project.

References

[1] E. Ju, B. Choi, J. Noh, and J. Lee, "Data-driven bird simulation," in *ACM SIGGRAPH 2011 Talks on - SIGGRAPH '11*. Vancouver, British Columbia, Canada: ACM Press, 2011, p. 1. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2037826.2037910

[2] L. A. Taylor, G. K. Taylor, B. Lambert, J. A. Walker, D. Biro, and S. J. Portugal, "Birds invest wingbeats to keep a steady head and reap the ultimate benefits of flying together," *PLoS Biol*, vol. 17, no. 6, p. e3000299, Jun. 2019. [Online]. Available: https://dx.plos.org/10.1371/journal.pbio.3000299

[3] B. Jorgensen, *Beej's Guide to Network Programming*, n.d. [Online]. Available: https://beej.us/guide/bgnet/html/split/index.html

[4] "Creating a scene - three.js," https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene, accessed March 17, 2023.

[5] Arduino, "Arduino nina-w10 data sheet (ubx-17065507)," https://content.arduino.cc/assets/Arduino_NINA-W10_DataSheet_%28UBX-17065507%29.pdf, n.d.

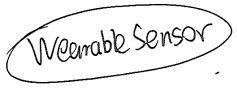
[6] Y. Shen, "Demo of uol-y2p-wearablesensor, or digitaltwin," Video, YouTube, March 1 2023. [Online]. Available: https://youtu.be/BtlxjvaMq8Y

[7] TDK InvenSense, "ICM-20600 Datasheet," https://invensense.tdk.com/wp-content/uploads/2021/05/DS-000184-ICM-20600-v1.1.pdf, 2021, [Online; accessed 17-March-2023].

Appendices

Appendix A

Project Management Forms



Y2 project (ELEC222/ELEC273) - *Attendance record*

Notes:

- ***This sheet should be updated by group members weekly when they meet to discuss the project.***
- ***Assessors will request to see this sheet in the bench inspection day.***

	Member name	Attended the weekly meeting? (Yes/No)										Comments
		Week 1		Week 2		Week 3		Week 4		Week 5		
1	Shen, Yixiao	Yes	Yes	✓	✓	✓	✓	✓	✓	✓		Good.
2	Guo, Jiajun	Yes	Yes	✓	✓	✓	✓	✓	✓	✓		Good.
3	Zhou, Qi	Yes	No	✓	✓	✓	✓	✓	✓	✓		Good.
4	Canning, Charles Thomas	Yes	Yes	✓	✓	✓	✓	no	✓	✓		Good.
5	Brissenden, Jack	Yes	Yes	✓	✓	✓	✓	✓	✓	✓		Good.

Outing.

working.

Figure A.1: Attendence record.

Weerable Sensor

Y2 project (ELEC222/ELEC273) - *Contribution to project deliverables*

Notes:

- ***Assessors will request to see this sheet in the bench inspection day.***
- ***Typical deliverables include: Bench, Poster, Blog, Report, Code, Circuit, etc.***

	Member name	Deliverable(s)	Comments
1	Shen, Yixiao	① Code of socket server & displaying engine. ② idea creator.	technical & creative, but need to improve communication skills.
2	Guo, Jialun	① Hardware code of IMU. ② Poster & Blog.	proactive & empathy.
3	Zhou, Qi	① Software code of IMU-data-matching in the displaying engine. ② Blog ③ Circuit constructing.	positive - attitude.
4	Canning, Charles Thomas	① Arduino code of socket. ② Report & Blog.	responsible & collaborative.
5	Brissenden, Jack	① Arduino code of IMU. ② 3D modelling & printing. ③ soldering & circuit constructing.	technical && proactive.

Figure A.2: Contribution to project deliverables.

Wearable Sensor

Y2 project (ELEC222/ELEC273) – *Role allocation (responsibility matrix)*

Notes:

- ***Assessors will request to see this sheet in the bench inspection day.***
- ***See overleaf for details on titles and associated roles and responsibilities.***

	Member Name	Title(s)
1	Shen, Yixiao	Project Manager Designer Software Developer
2	Guo, Jiajun	Hardware Developer Technical Writer
3	Zhou, Qi	Software Developer Technical Writer
4	Canning, Charles Thomas	Hardware Developer Technical Writer
5	Brissenden, Jack	Designer Hardware Developer

Figure A.3: Role allocation (responsibility matrix).

Year 2 Project (ELEC222/273) – *Supervisor meeting – Week 1*

Date: 30/1/2023 Supervisor: Dr. D. Wrinch

Project Title: Wearable Sensor

Student Names /Attendees:	1. Yixiao Shen	2. Jialun Guo
3. Qi Zhou	4. Jack Brissenden	5.

Summary of week's activities:

1. confirm the lab days.
2. complete part of project-management-forms.
3. assign poster & blog & weekly log-book to members.

Problem, issues and concerns:

1. Hardware: the using of IIC multiplexer.
2. Software: how to bind the joints with their corresponding IMUs.

Tasks for next week/Actions for next meeting:

(using multiplexer)

1. Hardware: read data from multiple IMUs at once, if successful, try to build communications with PC via ~~UART~~/Socket.
2. Software: Try to work out relavent coordinate transformations, generally, design a simple displaying engineer for all kinds of combinations of joints, binding joints with their corresponding IMUs.

Supervisor use only

Progress Assessment: □ Unsatisfactory □ Satisfactory ☒ Good

Comments/Recommendations:

Excellent plans. Keep working on those but bear in mind you may not achieve every objective in the end - if so, may need to prioritise.

Figure A.4: Supervisor weekly meeting log 1.

Year 2 Project (ELEC222/273) – *Supervisor meeting – Week 2*

Date: 6/2/2023 Supervisor: Dr. D. McIntosh

Project Title: Wearable Sensor

Student Names /Attendees:	1. Yixiao Shen	2. JiaJun Guo
3. Qi Zhou	4. Jack Brissenden	5. Canning Charles Thomas

Summary of week's activities:

~~Last week~~,
1. able to fetch data from single IMU.
2. support 2.4G Wi-Fi connection.
3. a simple displaying engineer using THREE.JS was built for all kinds of joint-connections.
4. a simple HTTP sever on MacOs was built for transmitting data in a local area network.

Problem, issues and concerns:

① Due to MRS's own assignments, the ~~process~~ progress of hardware system was affected. ~~is not~~ went as (new deadline has been postponed to this Friday).

② One solved problem is that the Arduino Uno Wifi REV2 only supports 2.4G, where we have downgraded the 5G network to 2.4G ~~one~~ in the router.

③ One problem ~~dect~~ detected is that improper Arduino IDE version may lead to ~~burning~~ uploading failure: the Arduino Nano Every ~~only~~ is incompatible with the 2.0.3 version.

Tasks for next week/Actions for next meeting:

Hardware: send multiple IMUs' data to server.

Software: receive the data and forward them to ~~udp~~ requestors (no modelling right now)
The main aim is to build a whole clean channel.

Better installtion:) model & build the 3D-printing part.
) check the 3rd component order list.

Supervisor use only

Progress Assessment: □ Unsatisfactory □ Satisfactory ☑ Good

Comments/Recommendations:

Well done! Main area for progress is now to keep developing the software, and keep getting the blog updated each week. Corrected minor typo's in blog.

Figure A.5: Supervisor weekly meeting log 2.

Year 2 Project (ELEC222/273) – *Supervisor meeting – Week 3*

Date: 13/2/2023 Supervisor: Dr. D. McIntosh

Project Title: Wearable Sensor

Student Names /Attendees:	1. Yixiao Shen	2. Jiulun Guo
3. Qi Zhou	4. Jack Brissenden	5. Canning Charles Thomas.

Summary of week's activities:

① able to fetch data from multiple IMUs finally.

② able to send messages to server via socket.

③ ordered the 3rd component list, which mainly contains elastic bands & screws.

Problem, issues and concerns:

① The integration of module "fetch ~~dat~~data from IMU" & module "send messages through socket" may have incompatibilities.

↳ or use wired communication (UART/USART).

Tasks for next week/Actions for next meeting:

① construct the whole system & test in practice, if things ~~are~~ arrive.

② extended IMU configuration & filtering.

③ motion-following for the "Digital Twin". ⇒ rotation & heigh-shifting.

Supervisor use only

Progress Assessment: ☐ Unsatisfactory ☐ Satisfactory ☑ Good

Comments/Recommendations:

Well done to all. If the work on the Arduino PCB works (to break internal SDA/SCL connections) works, then great, but if not, it is good to revert to a wired connection (after documenting your progress & problems with I2C/WiFi) with videos.

Figure A.6: Supervisor weekly meeting log 3.

Year 2 Project (ELEC222/273) – *Supervisor meeting – Week 4*

Date: 23/02/2023 Supervisor: Dr Dave McIntosh.

Project Title: Wearable Sensor

Student Names /Attendees:	1. Yixiao Shen	2. Jiajun Guo
3. Qi Zhou	4. Jack Brissenden	5. Canning Charles Thomas

Summary of week's activities:

1. solved the issue of "no socket available!" finally.
2. finished the sustainable ethics report.
3. printed all of 3D models.
4. able to have the Euler angle.
5. Finished previous five blogs, with refinement.

Problem, issues and concerns:

① concern: Qi Zhou ~~dos~~ may have not enough time to do those data stuff to make the model look right, ~~since solving~~ because solving the "no socket ~~is~~ available" issue has ~~consumed a lot of time. But we will try to get the~~ progress on ~~da~~ track.

② concern: ~~we have not received the battery holder yet.~~

Tasks for next week/Actions for next meeting:

1. print the poster.
2. a full system test.
3. ~~upload~~ upload the sustainable ethics report.
4. ~~determine the presentation specifics.~~

Supervisor use only

Progress Assessment: ☐ Unsatisfactory ☐ Satisfactory ☑ Good

Comments/Recommendations:

Excellent - you also solved a problem with I2C signal unexpectedly affecting the WiFi module functionality - well done!

Figure A.7: Supervisor weekly meeting log 4.

Appendix B

A Breakdown of Individual Contributions

Member Name	Breakdown	Overall Percentage
Yixiao Shen	3D model design (20%)	25%
	Blog (20%)	
	Displaying engine development (60%)	
	Socket server development (60%)	
	Project report (25%)	
	Sustainable development report (20%)	
	Poster (20%)	
Jiajun Guo	Blog (20%)	20%
	Project report (25%)	
	Sustainable development report (20%)	
	Arduino development (33.3%)	
	Poster (60%)	
	Circuit construction (20%)	
Qi Zhou	3D model design (20%)	15%
	Blog (20%)	
	Displaying engine development (40%)	
	Socket server development (40%)	
	Project report (25%)	
	Sustainable development report (20%)	
Charles Canning	Circuit construction (20%)	20%
	Blog (20%)	
	Project report (25%)	
	Sustainable development report (30%)	
	Arduino development (33.3%)	
Jack Brissenden	3D model design (60%)	20%
	3D model printing (100%)	
	Blog (20%)	
	Sustainable development report (10%)	
	Arduino development (33.3%)	
	Poster (20%)	
	Circuit construction (60%)	

Figure B.1: Breakdown of individual contributions.

Appendix C

Datasheets

ACCEL_FCHOICE_B	A_DLPF_CFG	Accelerometer		
		3-dB BW (Hz)	Noise BW (Hz)	Rate (kHz)
1	X	1046.0	1100.0	4
0	0	218.1	235.0	1
0	1	218.1	235.0	1
0	2	99.0	121.3	1
0	3	44.8	61.5	1
0	4	21.2	31.0	1
0	5	10.2	15.5	1
0	6	5.1	7.8	1
0	7	420.0	441.6	1

Table 19. Accelerometer Data Rates and Bandwidths (Low Noise Mode)

The data output rate of the DLPF filter block can be further reduced by a factor of 1/(1+SMPLRT_DIV), where SMPLRT_DIV is an 8-bit integer. Following is a small subset of ODRs that are configurable for the accelerometer in the low-noise mode in this manner (Hz):

3.91, 7.81, 15.63, 31.25, 62.50, 125, 250, 500, 1K

The following table lists the approximate accelerometer filter bandwidths available in the low-power mode of operation for some example ODRs.

In the low-power mode of operation, the accelerometer is duty-cycled. Table 20 shows some example configurations for accelerometer low power mode.

	Averages	1x	4x	8x	16x	32x
	ACCEL_FCHOICE_B	1	0	0	0	0
	DEC2_CFG	X	0	1	2	3
	A_DLPF_CFG	X	7	7	7	7
	Ton (ms)	1.084	1.84	2.84	4.84	8.84
	NBW (Hz)	1100	442	236	122	62
	3-dB BW (Hz)	1046	420	219	111	56
	Noise TYP (mg-rms)	3.3	2.1	1.5	1.1	0.8
SMPLRT_DIV	ODR (Hz)	Low-Power Accelerometer Mode Current Consumption (µA)				
255	3.91	9.4	10.2	11.5	13.8	18.5
127	7.81	10.7	12.4	14.7	19.6	28.9
99	10	11.4	13.7	16.6	22.6	34.7
63	15.63	13.3	16.7	21.5	30.8	49.7
31	31.25	18.3	25.4	34.8	53.6	91.2
19	50	24.4	35.8	50.8	80.8	141.1
15	62.5	28.4	42.7	61.5	99.0	174.3
9	100	40.7	63.5	93.6	153.7	303.3
7	125	48.8	77.4	114.8	190.1	N/A
4	200	73.4	118.8	178.9	299.3	
3	250	89.6	146.5	221.6	N/A	
1	500	171.1	284.9	N/A		

Table 20. Example Configurations for Accelerometer Low Power Mode

Figure C.1: Example output data rateconfigurations for accelerometer in low power mode, from datasheet [7].

11.41 REGISTER 116 – FIFO READ WRITE

Register Name: FIFO_R_W

Register Type: READ/WRITE

Register Address: 116 (Decimal); 74 (Hex)

BIT	NAME	FUNCTION
[7:0]	FIFO_DATA[7:0]	Read/Write command provides Read or Write operation for the FIFO.

Description:

This register is used to read and write data from the FIFO buffer.

Data is written to the FIFO in order of register number (from lowest to highest). If all the FIFO enable flags (see below) are enabled, the contents of registers 59 through 72 will be written in order at the Sample Rate.

The contents of the sensor data registers (Registers 59 to 72) are written into the FIFO buffer when their corresponding FIFO enable flags are set to 1 in FIFO_EN (Register 35).

If the FIFO buffer has overflowed, the status bit *FIFO_OFLOW_INT* is automatically set to 1. This bit is located in INT_STATUS (Register 58). When the FIFO buffer has overflowed, the oldest data will be lost and new data will be written to the FIFO unless register 26 CONFIG, bit[6] FIFO_MODE = 1.

If the FIFO buffer is empty, reading register FIFO_DATA will return a unique value of 0xFF until new data is available. Normal data is precluded from ever indicating 0xFF, so 0xFF gives a trustworthy indication of FIFO empty.

11.42 REGISTER 117 – WHO AM I

Register Name: WHO_AM_I

Register Type: READ only

Register Address: 117 (Decimal); 75 (Hex)

BIT	NAME	FUNCTION
[7:0]	WHOAMI	Register to indicate to user which device is being accessed.

This register is used to verify the identity of the device. The contents of *WHOAMI* is an 8-bit device ID. The default value of the register is 0x11. This is different from the I^2C address of the device as seen on the slave I^2C controller by the applications processor. The I^2C address of the ICM-20600 is 0x68 or 0x69 depending upon the value driven on AD0 pin.

11.43 REGISTERS 119, 120, 122, 123, 125, 126 ACCELEROMETER OFFSET REGISTERS

Register Name: XA_OFFSET_H

Register Type: READ/WRITE

Register Address: 119 (Decimal); 77 (Hex)

BIT	NAME	FUNCTION
[7:0]	XA_OFFS[14:7]	Upper bits of the X accelerometer offset cancellation. ±16g Offset cancellation in all Full-Scale modes, 15 bit 0.98-mg steps

Register Name: XA_OFFSET_L

Register Type: READ/WRITE

Register Address: 120 (Decimal); 78 (Hex)

BIT	NAME	FUNCTION
[7:1]	XA_OFFS[6:0]	Lower bits of the X accelerometer offset cancellation. ±16g Offset cancellation in all Full-Scale modes, 15 bit 0.98-mg steps
[0]	-	Reserved.

Register Name: YA_OFFSET_H

Register Type: READ/WRITE

Register Address: 122 (Decimal); 7A (Hex)

BIT	NAME	FUNCTION
[7:0]	YA_OFFS[14:7]	Upper bits of the Y accelerometer offset cancellation. ±16g Offset cancellation in all Full-Scale modes, 15 bit 0.98-mg steps

Figure C.2: Partial datasheet of tthe ICM20600 sensor component [7]. The No.117 register indicates that for one specific compnent, there are at most two selectable IIC addresses to use.

Appendix D

Codes

Listing D.1: C++ code to init the IIC multiplexer and four sensors.

```
1  //the full file could be found in https://github.com/UOL
       -Y2P-WearableSensor/PlatformIO.
2  //calibrating for three seconds
3  #define CALI_TIME_MS 3000
4  void Sensor::calibrating_external() {
5      Serial.print("IMU_external GYRO calibration, ETA=");
6      Serial.print(CALI_TIME_MS);
7      Serial.println("ms");
8
9      unsigned long start_time = millis();
10     int num = 0;
11
12     for (int port = 0; port < 8; ++port) {
13         memset(IMU_external_data[port].gyro_cali, 0, 3);
14     }
15
16     while (millis() <= start_time + CALI_TIME_MS) {
17         num++;
18
19         for (int port = 0; port < 8; port++) {
20             if (port == IMU_LEFT_FEMUR
21                 || port == IMU_RIGHT_FEMUR
22                 || port == IMU_LEFT_TIBIA
23                 || port == IMU_RIGHT_TIBIA) {
24                 I2CMulti.selectPort(port);
25                 memset(IMU_external_data[port].gyro, 0,
                       3);
26                 IMU_external.getGyroscope(
27                     &IMU_external_data[port].gyro[0],
28                     &IMU_external_data[port].gyro[1],
```

```
29                      &IMU_external_data[port].gyro[2]);
30                  for (int i = 0; i < 3; ++i) {
31                      IMU_external_data[port].gyro_cali[i]
                            -=
32                          (IMU_external_data[port].gyro[i
                               ]);
33                  }
34              }
35          }
36      }
37      Serial.print("cali data num: ");
38      Serial.println(num);
39      for (int port = 0; port < 8; port++) {
40          if (port == IMU_LEFT_FEMUR
41              || port == IMU_RIGHT_FEMUR
42              || port == IMU_LEFT_TIBIA
43              || port == IMU_RIGHT_TIBIA) {
44              for (int i = 0; i < 3; ++i) {
45                  IMU_external_data[port].gyro_cali[i] =
46                      IMU_external_data[port].gyro_cali[i]
                            / num;
47              }
48              Serial.print("IMU_external[");
49              Serial.print(port);
50              Serial.println("].gyro_cali");
51              Serial.print("\t\t[x]=");
52              Serial.println(IMU_external_data[port].
                   gyro_cali[0]);
53              Serial.print("\t\t[y]=");
54              Serial.println(IMU_external_data[port].
                   gyro_cali[1]);
55              Serial.print("\t\t[z]=");
56              Serial.println(IMU_external_data[port].
                   gyro_cali[2]);
57          }
58      }
59
60  }
```

Listing D.2: C++ code to package the motion data from Arduino into a JSON format.

```
1  //the whole file could be found in https://github.com/
       UOL-Y2P-WearableSensor/WearableSensor_Macos/tree/
       communicate_with_Arduino/src
2  #include "IMU_json.h"
3
```

```
4  std::string Euler_angle_t::toString() const {
5      return "{\"r\":" + data[0] + ",\"p\":" + data[1] + "
           }";
6  }
7
8  std::string IMU_Data_t::toString() const {
9      std::string json="{";
10
11     for (int imu = 0; imu < IMU_NUM; ++imu) {
12         json += "\"" + name[imu] + "\":[";
13
14         for (int pose = 0; pose < data[imu].size(); ++
               pose) {
15             json += data[imu].at(pose).toString();
16             if (pose != data[imu].size() - 1) json+=",";
17         }
18
19         json+="]";
20         if (imu != IMU_NUM-1) json+=",";
21     }
22
23     json+="}";
24     return json;
25 }
26
27 //This spilit function is copied from https://
       stackoverflow.com/questions/14265581/parse-split-a-
       string-in-c-using-string-delimiter-standard-c.
28 std::vector<std::string> split(const std::string &s,
       const std::string &delimiter) {
29     size_t pos_start = 0, pos_end, delim_len = delimiter
           .length();
30     std::string token;
31     std::vector<std::string> res;
32
33     while ((pos_end = s.find(delimiter, pos_start)) !=
           std::string::npos) {
34         token = s.substr(pos_start, pos_end - pos_start)
               ;
35         pos_start = pos_end + delim_len;
36         res.push_back(token);
37     }
38
39     res.push_back(s.substr(pos_start));
40     return res;
41 }
```

```
42
43
44 std::string send_IMU_data() {
45     //read the last five lines in IMU_data.txt, then
          phase2json
46     std::ifstream fs;
47     fs.open("../fileForServer/IMU_data.txt");
48
49     if (fs.is_open()) {
50         fs.seekg(-1, std::ios_base::end);
                              // go to one spot before the
             EOF
51
52         char ch;
53         int num = 0;
54         while (true) {
55             if (num >= FRAME2BROWSER+1) break;
56             fs.get(ch);
57             if (ch == '\n') num++;
58             fs.seekg(-2,
59                      std::ios_base::cur);
60         }
61         fs.seekg(2, std::ios_base::cur);
62
63         std::string tmp;
64         IMU_Data_t IMU_data;
65         Euler_angle_t Euler_angle;
66         int split_idx;
67         for (int pose = 0; pose < FRAME2BROWSER ; ++pose
             ) {
68
69             // Read the current line
70             getline(fs, tmp);
71             std::vector<std::string> v = split(tmp, " ")
                 ;
72
73             //except the time_idx "[19]"
74             split_idx = 0;
75
76             //store in the IMU_data struct
77             for (int IMU_idx = 0; IMU_idx < IMU_NUM; ++
                 IMU_idx) {
78                 if (split_idx > v.size() - 1) break;
79                 for (int angle_idx = 0; angle_idx < (
                     sizeof Euler_angle.data/ sizeof (std
                     ::string)); ++angle_idx) {
```

```
80                      Euler_angle.data[angle_idx] = v.at(
                           split_idx++);
81                  }
82                  IMU_data.data[IMU_idx].push_back(
                       Euler_angle);
83              }
84          }
85          fs.close();
86          std::string response="HTTP/1.1 200 Ok\r\nContent
               -Type: application/json";
87          response+="\r\n\r\n";
88          response+=IMU_data.toString();
89          std::cout<<response<<std::endl;
90          return response;
91      }
92
93  }
```

Listing D.3: JavaScript code for flexible-inputting function

```
1  /**
2   * logic:
3   * input node_child["parent_terminal"] first,
4   * then loop all joints to find node_parent whose "
       child_terminal" matches the node_child["
       parent_terminal"]
5   * where the this.mesh[0] is the zero terminal, "
       mesh_origin_point"
6   * calibrate the binding
7   * add a ball on the child terminal
8   * At last, this.meshes[node_parent].add(this.meshes[
       node_child]);
9   */
10
11      //the child_terminal is regarded as the index of one
            node.
12  let node_info, node_mesh, parent_knot, child_knot;
13  for (let i = 0; i < this.IMU_schedule.length; i++) {
14      // console.log(this.IMU_schedule[i]);
15      node_info = this.IMU_schedule[i];
16      node_mesh = this.mesh_joints[i];
17
18      //find parent_knot
19      if (node_info["parent_terminal"] === "0") {
20          console.log("\tfind the parent_knot: mesh_origin
               ");
21          parent_knot = this.mesh_origin;
```

```
22      } else {
23          //range of idx: 0,1,2,3,4; excluding 5(this.
               IMU_schedule.length)
24          let idx = 0;
25
26          for (; idx < this.IMU_schedule.length; idx++) {
27              if (this.IMU_schedule[idx]["child_terminal"]
                    === node_info["parent_terminal"]) {
28                  console.log("\tfind the parent_terminal:
                        ", this.IMU_schedule[idx]["
                       child_terminal"]);
29                  console.log("\tparent_knot idx = ", idx)
                       ;
30                  parent_knot = this.mesh_knots[idx];
31                  break;
32              }
33          }
34
35          if (idx === this.IMU_schedule.length) {
36              console.log("\tfailed to find parent_knot");
37              console.log(this.IMU_schedule[i]);
38              break;
39          }
40
41      }
42
43      //add the current joint to the parent_knot(one knot)
44      child_knot = this.mesh_knots[i];
45      parent_knot.add(node_mesh);
46      node_mesh.add(child_knot);
47
48  }
```

Listing D.4: JavaScript code for matching and interpolating data.

```
1  //Note that only the processing to right femur and right
       tibia are shown here as a mutual explanation.
2  // Other processing for the left part is similar and
      discarded for better illustration.
3
4  //threshold to avoid jittering
5  const threshold=0.02;
6
7  //process the pitch data of right femur
8  idx = 1;
9  name = "R_F";
10
```

```
11 //data matching process
12 current_pitch = this.mesh_joints[idx].rotation.x;
13 input_roll=new_data[name][0]["r"];
14 input_pitch=new_data[name][0]["p"];
15 if (Math.abs(input_roll - 180) < Math.abs(input_roll -
      0)
16     || Math.abs(input_roll + 180) < Math.abs(input_roll
         - 0)) {
17     k = 1;
18 } else {
19     k = -1;
20 }
21 target_pitch = (input_pitch / 180 * Math.PI + Math.PI /
      2) * k; //p
22 diff = (target_pitch - current_pitch) / (this.
      interpolation_num + 1);
23
24 //interpolating the processed data
25 this.IMU_data_R_F.splice(0, this.IMU_data_R_F.length);
26 if (Math.abs(diff) > threshold) {
27     //valid input
28     for (let i = 1; i <= this.interpolation_num; i++) {
29         let tmp = current_pitch + i * diff;
30         this.IMU_data_R_F.push(tmp);
31     }
32     this.IMU_data_R_F.push(target_pitch);
33 } else {
34     //invalid input, keep last pose
35     for (let i = 0; i <= this.interpolation_num; i++) {
36         this.IMU_data_R_F.push(current_pitch);
37     }
38 }
39
40
41
42 //process the pitch data of right femur
43 idx = 3;
44 name = "R_T";
45
46 //data matching process
47 current_pitch = this.mesh_joints[idx].rotation.x;
48 input_roll=new_data[name][0]["r"];
49 input_pitch=new_data[name][0]["p"];
50 if (Math.abs(input_roll - 180) < Math.abs(input_roll -
      0)
51     || Math.abs(input_roll + 180) < Math.abs(input_roll
```

```
        - 0)) {
52      k = 1;
53  } else {
54      k = -1;
55  }
56  //this.mesh_joints[1] is right_femur
57  target_pitch = this.mesh_joints[1].rotation.x
58      - (new_data[name][0]["p"] / 180 * Math.PI + Math.PI
        / 2) * k;
59  diff = (target_pitch - current_pitch) / (this.
    interpolation_num + 1);
60
61  this.IMU_data_R_T.splice(0, this.IMU_data_R_T.length);
62  if (Math.abs(diff) > threshold) {
63      for (let i = 1; i <= this.interpolation_num; i++) {
64          let tmp = current + i * diff;
65          this.IMU_data_R_T.push(tmp);
66      }
67      this.IMU_data_R_T.push(target_pitch);
68  } else {
69      for (let i = 0; i <= this.interpolation_num; i++) {
70          this.IMU_data_R_T.push(current);
71      }
72  }
```

Listing D.5: Log for the Euler angle output in Arduino.

```
1  08:57:23.619 > Connected to server
2  08:57:23.629 > [0] time consumed: 55
3  08:57:23.684 > [1] time consumed: 56
4  08:57:23.739 > [2] time consumed: 55
5  08:57:23.796 > [3] time consumed: 56
6  08:57:23.852 > [4] time consumed: 56
7  08:57:23.908 > [5] time consumed: 58
8  08:57:23.965 > [6] time consumed: 56
9  08:57:24.021 > [7] time consumed: 56
10 08:57:24.078 > [8] time consumed: 57
11 08:57:24.134 > [9] time consumed: 57
12 08:57:24.191 > [10] time consumed: 55
13 08:57:24.247 > [11] time consumed: 57
14 08:57:24.305 > [12] time consumed: 56
15 08:57:24.362 > [13] time consumed: 56
16 08:57:24.418 > [14] time consumed: 57
17 08:57:24.474 > [15] time consumed: 56
18 08:57:24.530 > [16] time consumed: 57
19 08:57:24.588 > [17] time consumed: 58
20 08:57:24.645 > [18] time consumed: 57
21 08:57:24.702 > [19] time consumed: 56
22 08:57:24.759 > [20] time consumed: 57
23 08:57:24.814 > [21] time consumed: 56
24 08:57:24.871 > [22] time consumed: 56
```

```
08:57:24.927 > [23] time consumed: 57
08:57:24.983 > [24] time consumed: 56
08:57:25.039 > [25] time consumed: 55
08:57:25.096 > [26] time consumed: 56
08:57:25.151 > [27] time consumed: 56
08:57:25.207 > [28] time consumed: 56
08:57:25.264 > [29] time consumed: 56
08:57:25.318 > [30] time consumed: 56
08:57:25.375 > [31] time consumed: 56
08:57:25.431 > [32] time consumed: 56
08:57:25.486 > [33] time consumed: 55
08:57:25.541 > [34] time consumed: 56
```

Listing D.6: A text config to generate a model of human lower body.

```
{
  "IMU_schedule": [
    {"name": "body",
        "parent_terminal": "0", "child_terminal": "1",
        "Length":  "6", "Width":  "1", "Height": "1"},
    {"name": "right_femur",
        "parent_terminal": "0", "child_terminal": "2",
        "Length":  "5", "Width":  "1", "Height": "1"},
    {"name": "left_femur",
        "parent_terminal": "0", "child_terminal": "4",
        "Length":  "5", "Width":  "1", "Height": "1"},
    {"name": "right_tibia",
        "parent_terminal": "2", "child_terminal": "3",
        "Length":  "4", "Width":  "1", "Height": "1"},
    {"name": "left_tibia",
        "parent_terminal": "4", "child_terminal": "5",
        "Length":  "4", "Width":  "1", "Height": "1"}
  ]
}
```

Listing D.7: Another text config to generate a model of full body

```
{
  "IMU_schedule": [
    {"name": "body",
        "parent_terminal": "0", "child_terminal": "1",
        "Length":  "6", "Width":  "1", "Height": "1"},
    {"name": "right_femur",
        "parent_terminal": "0", "child_terminal": "2",
        "Length":  "5", "Width":  "1", "Height": "1"},
    {"name": "left_femur",
        "parent_terminal": "0", "child_terminal": "4",
        "Length":  "5", "Width":  "1", "Height": "1"},
    {"name": "right_tibia",
        "parent_terminal": "2", "child_terminal": "3",
```

```
14          "Length":  "4", "Width":  "1", "Height": "1"},
15      {"name": "left_tibia",
16          "parent_terminal": "4", "child_terminal": "5",
17          "Length":  "4", "Width":  "1", "Height": "1"},
18      {"name": "left_arm",
19        "parent_terminal": "1", "child_terminal": "6",
20        "Length":  "4", "Width":  "1", "Height": "1"},
21      {"name": "right_arm",
22        "parent_terminal": "1", "child_terminal": "7",
23        "Length":  "4", "Width":  "1", "Height": "1"},
24      {"name": "left_forearm",
25        "parent_terminal": "6", "child_terminal": "8",
26        "Length":  "3", "Width":  "1", "Height": "1"},
27      {"name": "right_forearm",
28        "parent_terminal": "7", "child_terminal": "9",
29        "Length":  "3", "Width":  "1", "Height": "1"},
30      {"name": "head",
31        "parent_terminal": "1", "child_terminal": "10",
32        "Length":  "2.5", "Width":  "1", "Height": "1"}
33    ]
34  }
```