CS6650 Smart Sensing for Internet of Things

IMU based Haptic perception with an immersive virtual environment
FINAL REPORT

Rahul Bijarnya (CS19B070), Mekala Sai Kiran (ME19B132),
Ashish Kumar Shroti (ME19B083), Kanishkan M S (ME19B192)

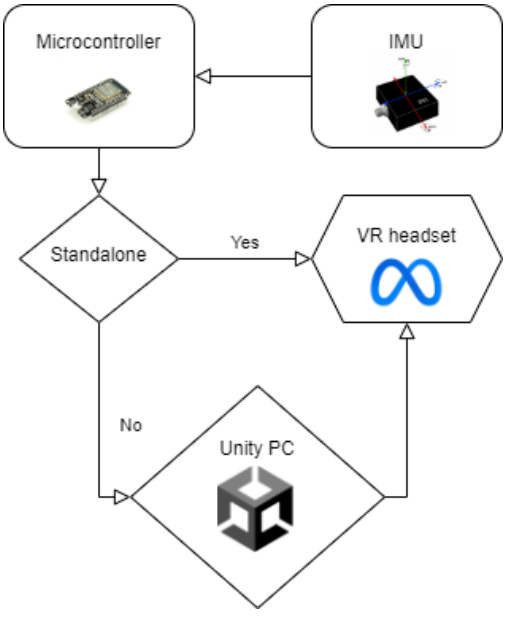
May 14 2023

1 Introduction:

- This project aims to develop an immersive VR game that utilizes IMU for object tracking and movement control along with providing haptic feedback that adds more immersion to the experience.
- This system involves developing software to interpret and process the IMU data, as well as designing and programming the game environment and mechanics.

2 System description:

The filtered data received from the Inertial measurement unit(IMU) sensor is sent to the Virtual environment created and uploaded to a Meta Quest 2 through an ESP32 microcontroller. The communication of this data is through WiFi case. And the Virtual environment is built for the Windows platform and streamed to Meta quest 2 via the Oculus link



3 Components involved:

1. ESP32 Microcontroller
2. IMU unit (MPU6050)
3. Meta Quest 2
4. **Softwares:** Unity,Arduino IDE

Complete hardware setup:

4 Methodology:

This project can be divided into three modules namely Data acquisition, Environment building, Data communication and VR integration

4.1 Data acquisition:

4.1.1 Inertial measurement unit

- The MPU6050 is a popular 6-axis motion tracking device that combines a 3-axis gyroscope and a 3-axis accelerometer on a single integrated circuit.
- It is commonly used in various applications such as motion sensors, game controllers, drones, and robotics. The device can measure angular velocity along three perpendicular axes (yaw, pitch, and roll) and linear acceleration along those same axes.
- The MPU6050 has an onboard **digital motion processor (DMP)** which performs motion processing tasks such as gesture recognition, orientation estimation, and sensor fusion, offloading these tasks from the main microcontroller.

For our project, we make use of the DMP unit to get quaternion orientation values of the IMU and to match the rotation mechanics in the Unity game environment

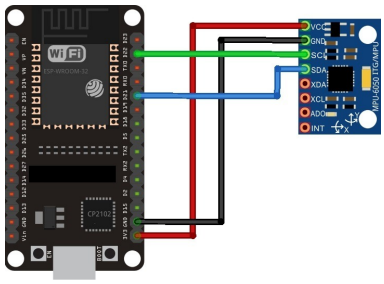
4.1.2 Microcontroller:

- The ESP32 is a low-cost, low-power system on a chip (SoC) microcontroller designed to be a versatile platform for Internet of Things (IoT) applications, featuring both Wi-Fi and Bluetooth connectivity, a dual-core processor, and a rich set of input/output peripherals.

- Compared to the popular Arduino microcontroller platform, the ESP32 has several advantages. First, the ESP32 has a much faster clock speed and more memory, allowing it to handle more complex applications and data-intensive tasks. It also has built-in Wi-Fi and Bluetooth connectivity, which eliminates the need for external shields or modules, and supports more advanced features like over-the-air updates and remote control.

4.1.3 Interfacing ESP32 and MPU6050

To get data from the MPU6050 sensor, it is connected to an ESP32 development board, and Arduino C code is uploaded via Arduino IDE



Note: *The interrupt pin of MPU6050 also should be connected to the D2 pin of ESP32 which is missing in the above diagram*

4.1.4 How is the data acquired:

- Instead of getting raw data from IMU, MPU6050 module has an inbuilt **Digital Motion Processor (DMP)** that provides hardware-filtered quaternion data, which represents the orientation of the device in 3D space.
- This data can be used to create more accurate and stable tracking of head movements in VR games. This also reduces the necessity of programming filtering algorithms such as Madgwick and Kalman filters.
- Quaternions are a mathematical concept used to represent rotations and orientations in three-dimensional space. They are an extension of complex numbers, which are used to represent rotations and orientations in two-dimensional space.
- In simple terms, a quaternion is a four-element vector with one scalar component and three vector components. The scalar component is known as the real part of the quaternion, while the vector components are known as the imaginary part of the quaternion.
- This kind of representation helps avoid the gimbal lock drawback that occurs when Roll, Pitch, and Yaw representation is used.

4.2 Unity Environment setup:

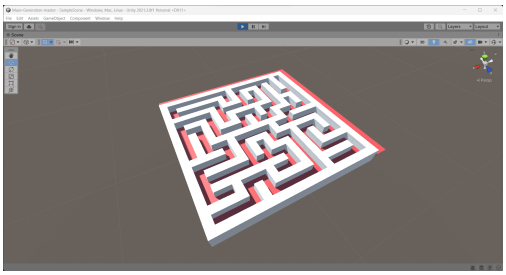
In the abstract, we mentioned about two different environments Handheld IMU maze setup and a Hand-held virtual fluid vessel.

- **Maze environment:** This entire maze will move based on the quaternion orientation that you get from the IMU which will be attached to a box-like structure to give the haptic perception

- **Virtual fluid vessel:** The bottle rotates based on the IMU data and fluid pours out of the vessel accordingly

4.2.1 Maze:

The maze was generated and attached to a parent gameObject and this parent gameObject's rotation is changed based on quaternion values that are received. And ball moves through the maze based on the IMU movement. A Rigid body sphere gameObject with continuous dynamic collision detection was also placed in the Maze. And it moves based on the maze's rotation.

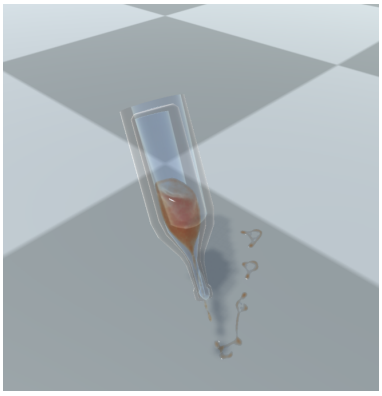


Issues:

- Sometimes the ball passes through the gameObject and falls down. Was not able to find a fix for that within the stipulated time.

4.2.2 Fluid:

Obi fluid package was used for creating the fluid simulation. The fluid particle emitter is placed inside a vessel and made sure it collides with the vessel.



Issues:

- The Obi Fluid renderer does not work in the case of VR. To make it work, some rendering settings have to tweaked. On trying this out, it resulted in the entire project not responding properly. Hence reverted back to the original version.
- Instead of a Fluid renderer, we used particle renderer to simulate the fluid movement. This is why the particles in the demo video also seems to be bursting rapidly, instead of smooth movements

4.3 Data communication:

- Websocketing is the concept used for sending the Quaternion data over WiFi to the VR application.
- This WebSockets is supported only for Windows VR applications. That is the reason for not building a standalone Android application for Meta quest 2 in our project.
- WebSocket uses the WebSocket Protocol, which is an independent TCP-based protocol designed for full-duplex communication between a client and a server over a single, long-lived connection. It provides a standardized way to establish and maintain a bi-directional communication channel between a web browser (or any client) and a web server.

Server code snippet:(Arduino C)
In void setup():

```
WiFi.begin(ssid, password);              // Connect to WiFi network
while(WiFi.status()!=WL_CONNECTED) {     // Wait for connection
 Serial.print(".");
 delay(500);
}
Serial.println("");
Serial.print("IP Address: ");
Serial.println(WiFi.localIP());

webSocket.begin();                       // Start the WebSocket server
```

In void loop():

```
   // Send the quaternion data to all connected WebSocket clients
   webSocket.broadcastTXT(dt.c_str(), dt.length());                    // Start the
       WebSocket server
```

Client code snippet:(Unity C
Void Start():

```
private void Start()
    {
        // Create a WebSocket instance and set up event handlers
        ws = new WebSocket("ws://192.168.184.42:81/");
        ws.OnMessage += OnMessageReceived;
        ws.OnOpen += OnWebSocketOpen;
        ws.OnMessage += OnMessageReceived;
        ws.OnError += OnWebSocketError;
        ws.OnClose += OnWebSocketClose;
        ws.Connect();
    }
```

Void update():

```
private void OnMessageReceived(object sender, MessageEventArgs e)
    {
        Debug.Log("Received: " + e.Data);
```

```
        // Process the received data here
        string[] values = e.Data.Split(',');

        if (values.Length >= 4)
        {
            // Parse the quaternion values
            float qw = float.Parse(values[0]);
            float qx = float.Parse(values[1]);
            float qy = float.Parse(values[2]);
            float qz = float.Parse(values[3]);

            // Create the new quaternion
            targetQuaternion = new Quaternion(qx, qz, qy, qw);

            //targetQuaternion = new Quaternion(qx, qy, qz, qw);

            // Set the flag indicating a new quaternion value is available
            hasNewQuaternion = true;
        }
    }
```

4.4 VR Integration:

The process of integrating the Oculus platform into the VR environments took a significant amount of time and effort. It involved various tasks such as enabling developer settings on the Meta Quest 2 device and resolving errors related to the **Oculus integration** packages in Unity. This integration phase also brought to light the issue of fluid rendering in VR, which required additional attention and troubleshooting.

5 Conclusion:

In the project, we tried implementing a pipeline of sending data from a custom hardware setup to a virtual environment. This pipeline could help us make more such projects with more interesting outcomes. For this particular project, there are some more improvements that could be looked. They are as follows:

1. The Maze environment does not have a start and end position for it to be called a game.
2. Altering the rendering pipeline for the fluid rendering issue to be fixed

3. Haptic perception is not provided completely, as the hardware module is not attached to similar looking object (Glass bottle incase of Fluid environment)

Beyond the scope of the project:
A Shifting-Weight Interface of Simulated Hydrodynamics for Haptic Perception of Virtual Fluid Vessels This paper deals almost with the same idea as ours. This is something that we could expand upon post the finishing this course project

6 References:

1. Quaternions and 3d rotation, explained interactively. https://www.youtube.com/watch?v=zjMuIxRvygQ
2. MPU6050 Explained https://mjwhite8119.github.io/Robots/mpu6050
3. Obi VirtualMethod simulation package http://obi.virtualmethodstudio.com/manual/6.3/
4. Unity VR documentation https://docs.unity3d.com/Manual/VROverview.html
5. Websockets Arduino https://arduino.stackexchange.com/questions/68319/