به نام خدا

گزارش نهایی آزمایشگاه سخت‌افزار

اعضای تیم:

امیرحسین نکوئي: ۹۹۱۰۹۹۲۸

علی صف‌آرافرد: ۹۹۱۰۵۵۸۳

محمد مولوی: ۹۹۱۰۵۷۵۳

فهرست مطالب

1. مقدمه

این گزارش به بررسی پروژه‌ی دوم می‌پردازد. در این پروژه هدف نهایی نمایش سه بعدی بی‌درنگ یک دستگاه، وسیله و ... بود. به گونه‌ای که با نصب و راه‌اندازی سنسور مورد نظر که در این پروژه از سنسور MPU6050 استفاده شد، بود بتوان این کار را انجام داد. همچنین برای گرفتن اطلاعات از سنسور از ماژول ESP32 استفاده نموده و آن را به یک پاوربانک وصل نمودیم. و در نهایت نیز با استفاده از Unity مدل سه‌بعدی وسیله‌ی مورد نظر را بارگذاری و ایجاد کرده و باحرکت وسیله به جهات گوناگون، مدل سه بعدی در Unity نیز به جهات گوناگون حرکت می‌کند.

در ادامه پروژه در چندین بخش مختلف بررسی می‌کنیم که به شرح زیر می‌باشد:

- بررسی کد TCP سمت ESP32 و همچنین دریافت اطلاعات در سمت سرور
- بررسی کدهای Unity و همچنین مدل سه بعدی
- بررسی سخت‌افزاری
- مراحل انجام
- چالش‌ها
- نتایج

2. پیاده‌سازی پرتوکل TCP

این پیاده‌سازی در دو بخش صورت گرفت. یکی در بخش کلاینت که همان سخت‌افزار و سنسور باشد که اطلاعات را می‌فرستد و دیگری هم در بخش سرور که همان مدل سه‌بعدی باشد و اطلاعات را دریافت می‌کند.

بخش کلاینت

```
void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");

while (1) {
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    delay(10);
  }
  break;
}
  mpu.setAccelerometerRange(MPU6050_RANGE_2_G);
```

```
  mpu.setGyroRange(MPU6050_RANGE_250_DEG);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);

  if (!client.connect(serverIP, serverPort)) {
    Serial.println("Connection to server failed");
  } else {
    Serial.println("Connected to server");
   }
}
```

ابتدا در تابع setup، هم پورت مورد نظر برای ارسال داده برای خواندن و نوشتن را تعیین می‌کنیم و هم به WiFi وصل می‌شویم. برای اتصال نیز از ماژول‌های آماده که برای همین کار ساخته شده‌اند استفاده می‌کنیم. به این صورت که ssid و رمز وای‌فای را می‌دهیم و بعد از اجرا شدن اگر ESP32 با موفقیت به وای‌فای وصل شود به مرحله‌ی بعدی می‌رود در غیز این صورت هر یک ثانیه یک بار دوباره تلاش می‌کند و به مرحله‌ی بعدی نمی‌رود.

تقریبا همین کار نیز برای وصل شدن به ماژول MPU6050 را نیز انجام می‌دهد و بعد از وصل شدن ستاپ اولیه‌ی آن را انجام می‌دهد. در نهایت سعی در اتصال کلاینت به سرور دارد که اگر وصل شود یا وضل نشود به مرحله‌ی بعدی می‌رود تا در آن‌جا اگر مشکلی بود و اتصال برقرار نبود، هندل شود.

```
void loop() {
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);
  if (client.connected()) {
    client.println("Acceleration X:" + String(a.acceleration.x) +
                ", Y:" + String(a.acceleration.y) +
                ", Z:" + String(a.acceleration.z) +
                " | Gyro X:" + String(g.gyro.x) +
                ", Y:" + String(g.gyro.y) +
                ", Z:" + String(g.gyro.z));
  } else {
    Serial.println("Disconnected from server");
    while (1) {
        if (!client.connect(serverIP, serverPort)) {
            Serial.println("Connection to server failed");
            delay(1000);
        } else {
            Serial.println("Connected to server");
            break;
        }
```

```
    }
  }
  delay(10);
}
```

این قسمت مربوط به به ارسال اطلاعات است. برای این به صورت بی‌درنگ اطلاعات ارسال شود، تعیین کردیم که هر ۱۰ میلی‌ثانیه یک بار حلقه‌ی لوپ تکرار شود. در حلقه‌ی لوپ، چک می‌شود که اگر اتصال برقرار است، اطلاعات ارسال و در غیر این صورت، سعی در برقراری ارتباط داشته باشد و تا زمانی که ارتباط برقرار نشده باشد، لوپ دوباره اجرا نشود.

بخش سرور

در این بخش، کد نوشته شده با C# است که در Unity برای دیگر بخش‌ها نیز بتوان به صورت راحت‌تری از آن استفاده کرد.

وقتی سرور شورع به کار می‌کند، کد قسمت اصلی که دیتا را دریافت می‌کند به شرح زیر است (کد اصلی در کنار این فایل قرار داده شده است، به علت خطوط بالا فقط یک قسمت را آورده‌ام):

```
while (true)
                {
                    TcpClient client = server.AcceptTcpClient();
                    Debug.Log("Client connected");
                    NetworkStream stream = client.GetStream();

                    while (client.Connected)
                    {
                        byte[] data = new byte[1024];
                        int bytesRead = stream.Read(data, 0, data.Length);
                        if (bytesRead > 0)
                        {
                            string receivedData = Encoding.UTF8.GetString(data,
0, bytesRead).Trim();
                            string[] dataForProccessArray =
receivedData.Split('\n');
                            foreach(string dataForProcess in
dataForProccessArray) {
                                try{
                                    ProcessData(dataForProcess);
                                }
                                catch (Exception e)
                                {
                                    Debug.LogWarning("Failed to process data: " +
e.Message + " dataForProcess " + dataForProcess);
                                }
                            }
```

```
                }
                else
                {
                    Debug.Log("No data received. Closing connection.");
                    break;
                }
            }
            client.Close();
            Debug.Log("Client disconnected");
        }
```

این قسمت به این صورت است که یک Connection از سمت کلاینت قبول می‌کند و اگر این اتصال قطع شد، دوباره سعی در برقراری اتصال دارد. همچنین اگر اتصال برقرار بود، داده‌ی دریافت‌شده از سمت کلاینت را می‌خواند و به تابع ProcessData پاس می‌دهد که بعدا درباره‌ی آن صحبت می‌کنیم.

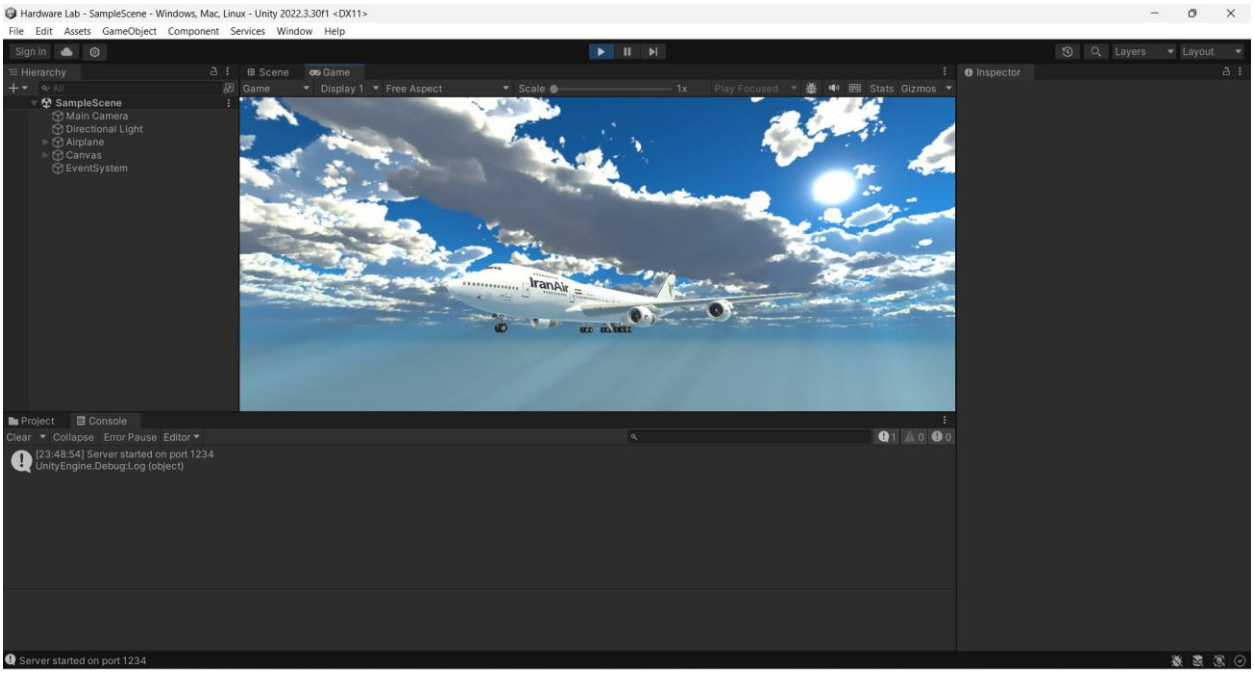
3. پیاده‌سازی در Unity و مدل سه‌بعدی

مدل سه‌بعدی

مدل سه‌بعدی به کار گرفته شده در حقیقت یک هواپیما است. برای همین نیاز شد تا هواپیما را از بازار سفارش دهیم. نمایی از آن را در تصویر شماره‌ی یک مشاهده می‌کنید.

تصویر 1 نمایی از هواپیما

همچنین مدل سه‌بعدی آن را در تصویر شماره‌ی دو به همراه محیط برنامه‌ی Unity می‌توانید مشاهده کنید.



تصویر 2 مدل سه بعدی هواپیما و محیط برنامه‌ی Unity

پیاده‌سازی در Unity

همانطور که بالاتر گفته شد، داده‌ها به تابع ProcessData پاس داده می‌شوند. این تابع با توجه به نحوه‌ی ارسال داده از سمت کلاینت، آن‌ها را پارس می‌کند و اطلاعات مورد نیاز شامل سرعت و شتاب را استفاده می‌کند. کد آن را در زیر می‌توانید ببینید.

```
    void ProcessData(string data)
    {
        string[] parts = data.Split('|');

        string accelerationData = parts[0].Trim();
        string gyroData = parts[1].Trim();

        string[] accelValues = accelerationData.Split(',');
        float accelX = float.Parse(accelValues[0].Split(':')[1]);
        float accelY = float.Parse(accelValues[1].Split(':')[1]);
        float accelZ = float.Parse(accelValues[2].Split(':')[1]);

        acceleration = new Vector3(accelCoeffs.x * accelY,accelCoeffs.y
*  accelX,accelCoeffs.z *  accelZ);
        Debug.Log(acceleration);
        velocity = new Vector3(accelY, accelZ, accelX);

        string[] gyroValues = gyroData.Split(',');
```

```
        float gyroX = Mathf.Round(float.Parse(gyroValues[0].Split(':')[1]) * 10f)
/ 10f;
        float gyroY = Mathf.Round(float.Parse(gyroValues[1].Split(':')[1]) * 10f)
/ 10f;
        float gyroZ = Mathf.Round(float.Parse(gyroValues[2].Split(':')[1]) * 10f)
/ 10f;

        angularVelocity = new Vector3(gyroY, gyroX, gyroZ);

        StringBuilder logMessage = new StringBuilder("Effective Values : Gyro :
");
        logMessage.Append(gyroX).Append(" , ").Append(gyroY).Append(" ,
").Append(gyroZ).Append(" Accel: ").Append(accelX).Append(" ,
").Append(accelY).Append(" , ").Append(accelZ);
        Debug.Log(logMessage.ToString());

        StringBuilder textMessage = new StringBuilder("Gyro : ");
        textMessage.Append(gyroX).Append(" , ").Append(gyroY).Append(" ,
").Append(gyroZ);
        textMessageString = textMessage.ToString();
    }
```

همچنین یک تابع آپدیت وجود دارد که می‌توانید کد آن را در قسمت زیرین ببیند. این تابع به این صورت کار می‌کند که بر اساس داده‌های شتاب‌سنج یا سرعت زاویه‌ای، چرخش یک شی را کنترل می‌کنیم. اگر useAccelerationForRotation فعال باشد، چرخش را محاسبه می‌کنیم تا جسم را با بردار گرانش تراز کنیم، چرخش 90 درجه را اعمال می‌کنیم و چرخش فعلی را به سمت مورد نظر هدایت می‌کنیم. اگر useAccelerationForRotation فعال نباشد، مستقیماً چرخش را با استفاده از سرعت زاویه‌ای اعمال می‌کنیم. سپس چرخش نهایی را روی تبدیل شی اعمال می‌کنیم.

```
    void Update()
    {
        if (useAccelerationForRotation)
        {

            Vector3 normalizedRefGravity = referenceGravity.normalized;
            Vector3 normalizedAcceleration = acceleration.normalized;

            Quaternion targetRotation =
Quaternion.FromToRotation(normalizedAcceleration, normalizedRefGravity);
            Quaternion clockwise90Rotation = Quaternion.Euler(offset);
            targetRotation *= clockwise90Rotation;
```

```
            airplaneRotation = Quaternion.Slerp(airplaneRotation, targetRotation,
Time.deltaTime * 2f);

            Vector3 steerVector = Vector3.Scale(angularVelocity, impact);
            if (steer) {
            steerVector = new Vector3(0,-steerVector.y,0);
            Quaternion deltaRotation = Quaternion.Euler(steerVector *
Time.deltaTime);
            airplaneRotation *= deltaRotation;
            }
        }
        else
        {
            Quaternion deltaRotation =
Quaternion.Euler(Vector3.Scale(angularVelocity, impact) * Time.deltaTime);
            airplaneRotation *= deltaRotation;
        }
        transform.rotation = airplaneRotation;
        textM.text = textMessageString;
    }
```

همچنین کلاسی وجود دارد تا نحوه‌ی حرکت دوربین را مشخص کند. همانطور که مشخص شده است، متغیرها با این مقدار خاص کالیبره شده‌اند.

```
public class CameraController : MonoBehaviour
{
    public float moveSpeed = 10.0f;
    public float lookSpeed = 2.0f;

    private float yaw = 0.0f;
    private float pitch = 0.0f;

    void Update()
    {
        yaw += lookSpeed * Input.GetAxis("Mouse X");
        pitch -= lookSpeed * Input.GetAxis("Mouse Y");

        pitch = Mathf.Clamp(pitch, -90f, 90f);

        transform.eulerAngles = new Vector3(pitch, yaw, 0.0f);

        Vector3 direction = new Vector3(Input.GetAxis("Horizontal"), 0,
Input.GetAxis("Vertical"));
        transform.Translate(direction * moveSpeed * Time.deltaTime, Space.Self);
    }
```

```
}
```

4. پیاده‌سازی سخت‌افزاری

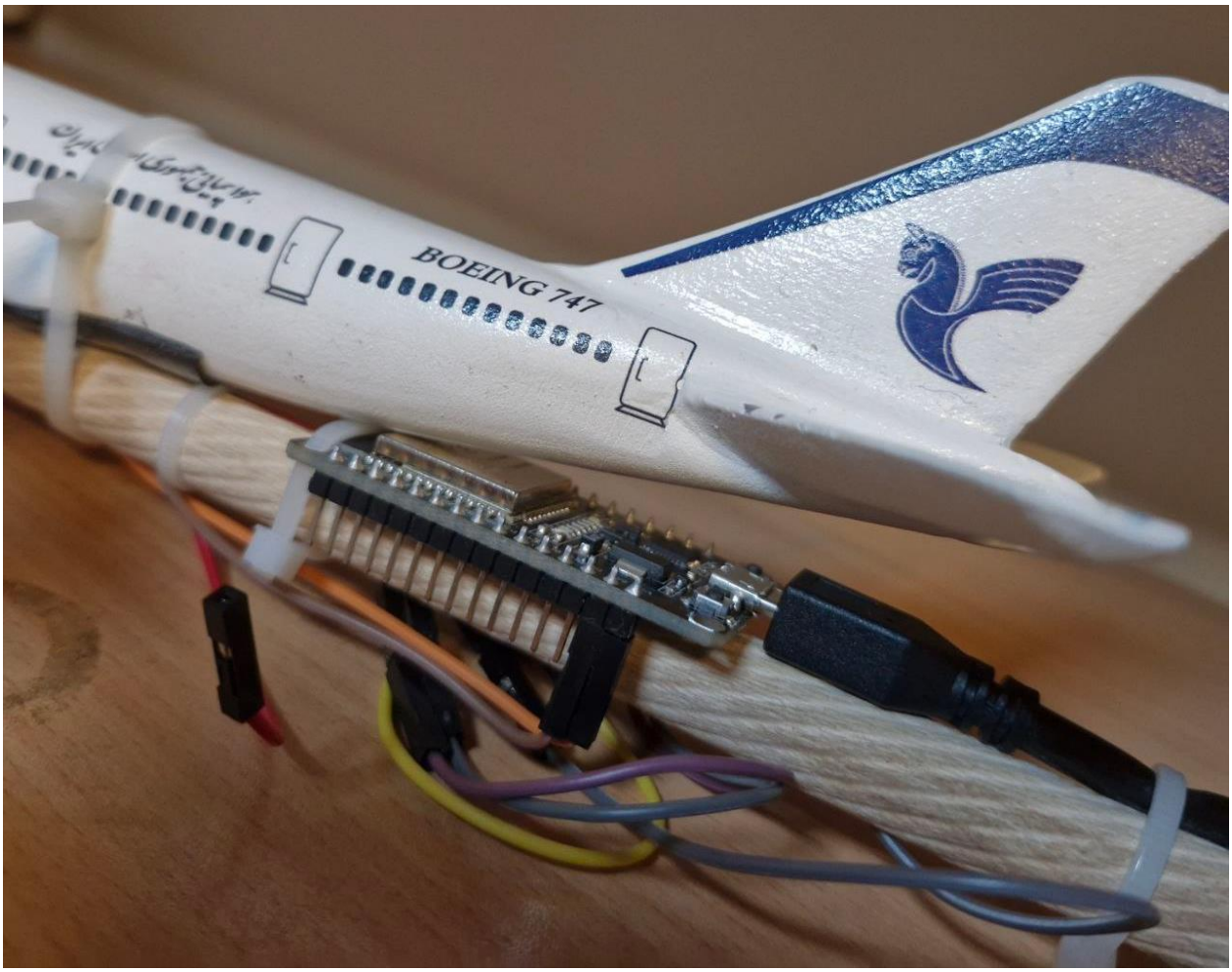
در این قسمت، چند کار باید انجام می‌شد:

- اتصال MPU6050 به ESP32: این قسمت تقریبا آماده بود با توجه به اینکه سخت‌افزارها روی یک چوب سوار بودند. تنها برخی از پایه‌های آن را درست در سر جای خود قرار دادیم.
- اتصال MPU6050 به هواپیما: در این قسمت همان چوب را به هواپیما متصل کردیم و با تکان دادن هواپیما، چوب و سنسور هم تکان می‌خوردند و اطلاعات به درستی دریافت می‌شد.

تصاویر پیاده‌سازی نهایی را می‌توانید در شکل یک، سه و چهار و پنج مشاهده نمایید.

تصویر 3 نمایی از آماده‌سازی نهایی

تصویر 4 نمایی از آماده‌سازی نهایی



تصویر 5 نمایی از آماده‌سازی نهایی

5. مراحل انجام

این پروژه در چندین هفته انجام شد که در ادامه کارهایی که در هر هفته انجام شد، توضیح داده خواهد شد.

هفته‌ی اول

در این هفته، ابتدا به بررسی و تحقیق درباره وسایل داده شده پرداختیم و با نحوه کار کردن و استفاده از آنها آشنا شدیم. سپس سنسور را به ESP32 متصل کردیم. سپس در نرم‌افزار Arduino، پکیج‌های مربوط به ESP32 را دانلود و نصب کردیم و تنظیمات لازم برای کار با پکیج را اعمال کردیم.
بعد از راه‌اندازی و تست کردن کار با ESP32 در این نرم‌افزار، شروع به نوشتن کد مورد نیاز برای دریافت داده از سنسور کردیم. در ادامه با نوشتن کد و همچنین تست کردن و دیباگ کردن آن، توانستیم کد مورد نیاز را تکمیل کنیم و اطلاعات را به درستی از سنسور دریافت کرده و نمایش دهیم.

هفته‌ی دوم

در این هفته طبق برنامه‌ی گفته پروتوکل TCP را سمت سرور با زبان پایتون و همچنین بر روی دستگاه ESP32 پیاده‌سازی کردیم. پیاده‌سازی به این صورت است که هر ۱۰ میلی‌ثانیه یک بار، دیتا به سمت سرور فرستاده می‌شود.

هفته‌ی سوم

در این هفته مدل سه‌بعدی توسعه یافت و همچنین وسیله‌ی مورد نظر که هواپیما بود نیز خریداری شد. مدل سه بعدی با استفاده از Unity توسعه یافت و همچنین کد سمت server که می‌بایست اطلاعات را از ESP32 دریافت کند نیز به زبان C# نوشته شد.

هفته‌ی چهارم

در این هفته سخت‌افزار به هواپیما متصل گردید و همچنین باگ‌های کوچکی که در سمت Client و Server وجود داشت برطرف گردید. همچنین پارامترهای مناسب برای متغیرهایی که وظیفه‌ی کنترل حرکت هواپیما را داشتند تعیین شد تا به درستی کار کند.

6. چالش‌ها

به طور کلی به چالش خیلی سختی برخورد نکردیم که برطرف کردن آن ما را با سختی مواجه کند. اما فیلتر بودن سایت‌هایی که برای دانلود و نصب ابزار و برنامه‌ها مثل Arduino و مدل‌های Unity وجود داشت کمی کار را سخت کرده بود. همچنین کالیبره کردن هواپیما و حرکت آن نیز کمی سخت بود پیدا کردن پارامترهای مناسب کمی زمان‌بر بود.

یکی دیگر از چالش‌ها که تقریبا تا آخر پروژه نیز وجود داشت، نحوه‌ی آپلود کردن کد نوشته شده در Arduino به ESP32 بود. زمان‌بندی اینکه چه وقتی ESP32 را در حالت دانلود بگذاریم خیلی مهم بود.

7. نتایج

همانطور که در بخش مقدمه نیز بیان شد، هدف این پروژه نمایش سه‌بعدی بی‌درنگ یک دستگاه بود. در نهایت با اتاصل سنسور به هواپیما و اراسل اطلاعات حرمت هواپیما به سرور، حرکت هواپیما به صورت بی‌درنگ در سمت سرور نمایش داده می‌شود و از آن برای شبیه‌سازی می‌توان استفاده نمود.