SKATRIXX

Technical Document

URBANSPORTS
PERFORMANCE CENTRE

Wessel van der Heijden
Thupten Rekonkati
Nazar Bachynskyy
Fleans Metsi

Tabel of contents

Introduction

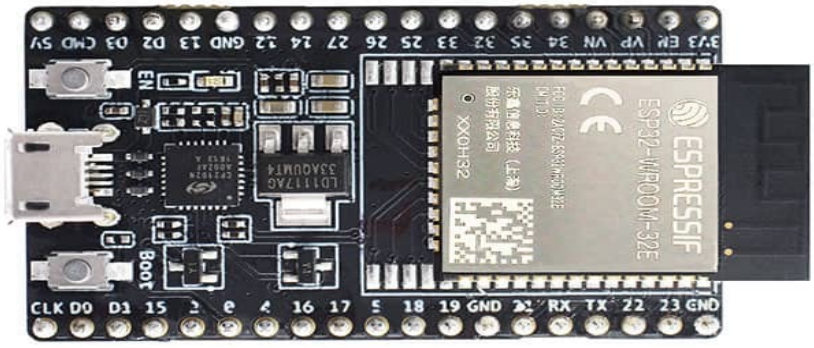
This document will describe the technical aspects of the prototype that was made by our team as part of the continuation of the Skatrixx project which was assigned to us by Fontys and the Urban Sport Performance Centre. The initial project that was presented to us was a skateboard with an Arduino and a motion sensor (MPU-9250) connected to it. With this hardware component measured data of the skateboard (e.g., acceleration and rotation speed) could be recorded and stored. It was the goal of our team to attempt to make a proof-of-concept product of what could be done with this data or the sensor in general.

This document will be spilt in two parts: Hardware and software solutions that our team created. It will go into detail about the implementation and the decisions made for our final proof-of-concept.

Hardware

To start off, we looked at the hardware side of the project. The initial project that was giving to us most importantly consist of an Arduino Nano 33 IoT, to handle the on-board control of the sensor and calculations required and a motion-sensor the MPU-9250. The documentation and the code that was used of the initial project was also provided to us. We also had access to the actual created prototype and skateboard. But due to COVID restrictions, we eventually could not have a thorough look at the project. Fortunately, the provided material of the project was enough to give us a good starting point.

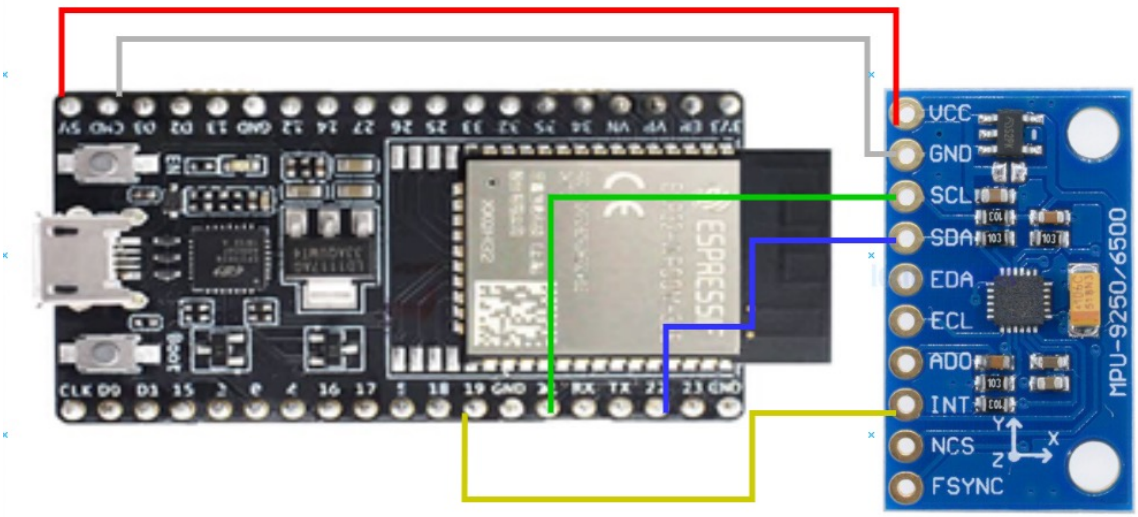
We had to explore what we could do with this project, so we looked that what could be improved. First, we looked at the data stream. In initial project, the data was being recorded and saved by having a wired connection to the Arduino. This was because it was prototype, and the initial project had the requirement of having a wireless connection low. We decided that we should make it priority to have a wireless connection. We chose to do this by replacing the Arduino with an ESP-32. This a microcontroller, like an Arduino, but has Wi-Fi and Bluetooth functionality already shipped with it. This would make our process easier, as we did not have to take and Arduino and attach a Bluetooth module to it before we could start and send the movement data to our UI. Additionally, the ESP32 is programmed with the same language as an Arduino and therefore did not require us to acquire new knowledge.



Next, we looked at the sensor that was being used. This was an MPU-9250 9-axis Motion Processing Unit, commonly used in smartphones and other wearable smart devices to measure motion data. This sensor has many different sensors, including an accelerometer, gyroscope, magnetometer, and temperature measurer. But most importantly, it also had a DMP (Digital Motion Processor). This sensor can measure data about the orientation of the sensor in commonly used 3D position data such as Euler angels and quaternions. This sensor was unused in the initial project, but was just what we needed for our project, were we wanted to make a 3D model of a skateboard reflect on the screen what movement the skateboard was making in the real world.

These two were combined so that the data could be recorded and then send via Bluetooth to our PWA application. The ESP32 and MPU-9250 were connected with wires on the pins shown below:

ESP	MPU9250
VCC	VCC
P22	SCL
P21	SDA
P19	INT
GND	GND



Further, to make sure the whole set-up is wireless a power-bank was also attached to the skateboard to power the ESP32. This whole set-up is quite bulky and is of course only a prototype. It is however out of the scope of our project to develop the final hardware product. This set-up was made purely to test and see is the sensor could be used effectively and if the data could be successfully sent via a wireless connection.

The code that was used for the hardware can be found in the repository of the whole project.

Software

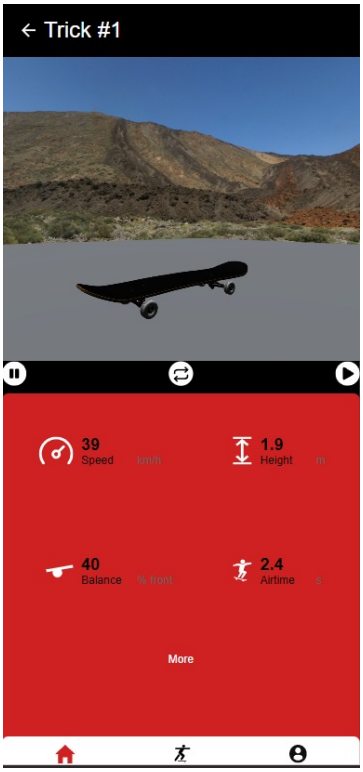
Now looking at the software part of the project. We decided to make a PWA. This was done for a few reasons:

- Languages that our team is most comfortable (HTML, CSS, JS)
- Can be used on both Android and iOS devices
- Has a great library for 3D visualization

The 3D visualization was made using a JavaScript library called three.js. It an open-source 3D computer graphics animator of web browsers using WebGL. Using this library, we were able to load in a 3D model of a skateboard. In short, the steps that taken when creating our 3D environment in three.js are as follows:

1. Create a Scene Object
2. Create a Renderer Object
3. Adding the renderer into a HTML element, the visualization will be shown here
4. Create a Camera Object
5. Create 2 Light sources
6. Adding OrbitCameraControls, so that the camera can be moved around
7. Adding a skybox and a ground plane
8. Loading in the skateboard model

A representation of the visualization can be seen here:



Another part was the code written for connecting to the ESP32 via Bluetooth. This makes uses of the fact that nowadays you can access many different native features of smartphones.

```
function connectToBLE() {

    initTHREEjs()
    navigator.bluetooth.requestDevice({filters: [{services: [primaryServiceString]}]})
        .then((device) => {
            console.log('Bluetooth device connected')
            return device.gatt.connect()
        })
        .then((server) => {
            console.log('Service found')
            return server.getPrimaryService(primaryServiceString)
        })
        .then((service) => {
            console.log('Service found')
            return service.getCharacteristic(sensorDataCharacteristicString);

        }).then(characteristic => {

        sensorDataCharacteristic = characteristic;
        return sensorDataCharacteristic.startNotifications().then(_ => {
            console.log('Notifications started');
            connected = true
            sensorDataCharacteristic.addEventListener('characteristicvaluechanged',
                handleNotifications);
        });
    })
```

An interesting note is that there is the ability to filter of specific devices by providing with the identifier that is also defined on the ESP32 code. First the device is found, then the server, service and finally the characteristic. Because the data send by the ESP32 is a constant stream, an eventListener is defined to handle everything a new data set is send. This gets handled in the “handleNotifications” method.

```
function handleNotifications(event) {

    let receivedData = new Uint8Array(event.target.value.byteLength);
    for (let i = 0; i < event.target.value.byteLength; i++) {
        receivedData[i] = event.target.value.getUint8(i);
    }
    let DMString = new TextDecoder().decode(receivedData)
    let DMPJson = JSON.parse(DMString)

    // If recording on the movement has started, the data is saved to a array for further use in the creation of ani
    if (record) {
        saveMovementData(DMPJson)
    }

    let q0 = DMPJson.q0;
    let q1 = DMPJson.q1;
    let q2 = DMPJson.q2;
    let q3 = DMPJson.q3;

    let quat1 = new THREE.Quaternion(q1, q2, q3, q0);
    let quat2 = new THREE.Quaternion(quatInit.qx, quatInit.qy, quatInit.qz, quatInit.qw);
    skateBoardMesh.quaternion.multiplyQuaternions(quat1, quat2);
}
```

Here the data is first converted from the Unsigned Int 8-bit form to a standard string. This string is then parsed into a JSON Object so that the values can easily be applied to the 3D model. A quaternion multiplication is applied on the 3D model. On the screen is this then seen as a representation of the position and orientation of the skateboard.