Embedded System Final Project Report

B10901112 陳品翔, B10901156 柯育杰, B10901161 張梓安

June 13, 2024

1 Introduction

This report introduces the final project we've done including motivation, approach, outcome, work distribution and our reference. We write a Parkour-like game similar to "Subway surfers", which uses our body motions to control the role in the game. For details, please see the following parts. Also welcome to visit our GitHub repository.

2 Motivation

Many people have experience in playing "Subways Surfers", a famous mobile Parkour game. But we can only move the role on screen using fingers, which is a little bit boring. So we came up with an idea, why don't we design a similar game that has to use our body motions to control the role in the game? With the idea, we create this Parkour-like game.

Figure 1: Subway Surfers, a interesting game that many people have played it before.

3 Approach

The whole design can be separated into three parts. The first part is that we use STM32 board to collect data of body motion, then use our own algorithm to detect what the motion is. Afterwards, we send "LEFT", "RIGHT", "UP" or "DOWN" via WiFi to the game server. Then the role in the game will move depending on the string we sent. The following are our design process.

3.1 Motion Detection

The algorithm to detect body motions is not easy to design. I tried four different algorithms: AI Keras model, Difference method, Priority method and Standard deviation method, respectively.

3.1.1 AI Keras Model

First, I wanted to use AI to detect body motions, because I thought using the model could raise the accuracy of detection. I referred to a web called "How to perform motion sensing on STM32L4 IoTnode", which provided a complete method. I finished the part from 1. to 7., and also using the provided Keras model and dataset to detect body motion such as stationary, walking and running successfully.

However, when I tried to construct my own dataset, two problems occurred. The first one is that there is a trade-off of the Keras model. If I want to raise the accuracy, I have to collect more data of motions, which is hard for me to collect such a large amount of data. Besides, more data means more comparison time and thus more delay. If I use a small dataset, the accuracy will not fit my expectation. The second problem is that the motions in the reference web are easy to distinguish, because stationary, walking and running have significantly different data. But what I want to distinguish is moving left, right, up and down, all of them are very similar, so the detection accuracy is not that high as I expected.

I still left my AI Keras model and other code in the GitHub repository (the zip file name "Detecting Motion.zip"). If you think that maybe you can conquer the two problems that I just mentioned, feel free to try. Note that in order to use the AI Keras model, you will need to use STM32CubeIDE instead of Mbed Studio, and the setting in STM32CubeIDE is a little bit hard. But don't worry, just follow the procedure in "How to perform motion sensing on STM32L4 IoTnode". (Hint: you may need to generate your own Keras model using this Colab link.)

3.1.2 Difference Method

I tried to use another method on Mbed Studio. The first method that came to my mind is that I can determine the body motion depending on the difference of data. We use BSP to collect the accelerometer data. Then I calculate the difference between the coming data and previous data, to determine the moving direction. If the difference of x-axis is less than 0 and higher than a threshold value, the moving direction is "LEFT"; if it's greater than 0 and higher than a threshold value, the moving direction is "RIGHT". The z-axis is similar for "UP" and "DOWN". But now a new problem occurred: during the time moving left, there is a stage where I stop to raise my speed. If the STM32 board collects the data, the motion detection may change to "RIGHT". However, this is not correct. Another problem is that if I rotate the board accidentally, the motion detection will be executed. That is, it is very sensitive. No matter how I changed the threshold value, the accuracy is not as good as I expected.

3.1.3 Priority Method

The second method that I wanted to implement on Mbed Studio is using priority. I separate the four motions into two groups: horizontal group and vertical group. Since the STM32 board seems more sensitive to vertical motions, this algorithm determines whether the motion detected is vertical first. If it is, use the same method that I mentioned in the Difference method to determine "UP" or "DOWN". If it isn't, the motion belongs to the horizontal group, so just determine "LEFT" or "RIGHT" using the same method in the Difference method. This method seems to have a higher accuracy, but I thought that it did not fix the whole problem at all, so I discarded it and tried to find another algorithm.

3.1.4 Standard Deviation Method

Finally, I used a method a little bit similar to this web, but these two methods still have lots of differences. I still used STM32 BSP to collect the accelerometer data. Note that we create a buffer to store the data (the maximum capacity is 20), and collect data very 5 millisecond. When the buffer is full, start processing.

First, detect whether the board is stationary or moving. This is determined by the conditions that one of the differences of standard deviation between each two of x-axis, y-axis and z-axis are greater than a threshold value. This value is determined depending on the data we collect. If the board is moving, then start to determine the moving direction. If the standard deviation of the x-axis is the highest, the motion is horizontal. If the standard deviation of the z-axis is the highest, the motion

is vertical. Now, we separate the motion into two groups. Then determine "LEFT" or "RIGHT", or "UP" or "DOWN". If the first value of the x-axis collected is less than a threshold value, the moving direction is "LEFT", otherwise it is "RIGHT". Similarly, if the first value of the z-axis collected is less than a threshold value, the moving direction is "DOWN", otherwise it's "UP".

Note that all the motions have a duration, but I just need to send one string to the game server (or the role in the game will move in the same direction consecutively, but this is not what we want). Thus, I use a timer to prevent the board sending the same string consecutively.

After some testing, the algorithm is good, but still a little bit imprecise. So we decided to use a FIR lowpass filter to filter the collected data. For details, please see the following part.

3.2 Data Filtering, Rotation Calibration and Data Transfer via WiFi

3.2.1 Data Filtering

For more precise sensor data and to avoid small noise, we applied an FIR lowpass filter similar to the one taught in class. When the buffer is full, the data in the buffer is applied to the lowpass filter. With the filtered data, we modified some detection threshold value in the above mentioned detection algorithm, and got a more accurate detection result.

3.2.2 Rotation Calibration

In the above algorithm part, we encounter a problem related to tilt angle. I tried two methods to compensate for the tilt angle but in vain. But I still introduce these methods in case someone can conquer the problem with which I faced and use the methods to do rotation calibration.

1. **Gravity Calibration**

   The accelerometer on the STM32 board uses the force it is exerted to sense the acceleration. Thus, when the Board rotates, the force on the z-axis will be distributed to the x-axis and y-axis. By using some trigonometric calculation, you can use the component of gravity on x-axis and y-axis to get the tilt angle with respect to x-axis and y-axis (For more details search for "Pitch and roll angle"). This gives us a very precise angle calibration. However, it turns out that the method is not suitable for our application. When we move, the additional acceleration makes the calibration result wrong, so we have to abandon this method.

2. **Gyroscope Method**

   By using gyroscope on the STM32 board, we can get the angular velocity of rotation with respect to every axis. We can theoretically get the rotation angle by integrating the angular velocity. Using Riemann sum, we can calculate the approximated angle. Unfortunately, the result is not as expected: a ridiculous deviation that can not be even called an "approximation". It is possible that the calculation can be done more properly to get a precise "approximation", but we did not do that due to time issues (As mentioned above, we change not only the algorithm various times but also the development environment from STM32cubeIDE to Mbed studio due to failure of using AI model).

3.2.3 Data Transfer via WiFi

The transmission is quite straightforward because of the well designed ism43362(WiFi module on board) driver API. I utilized the driver API to define my own WiFi sender class. The class has member functions that help us to easily connect to WiFi AP and transmit the data through a TCP socket to the game server.

3.3 Parkour-like Game Design

My part is to design a Parkour-like Game which can receive data from STM32L4 via WiFi.

3.3.1 Design Game

I use Ursina to help me design the game. Ursina is a Python-based game engine designed for quick prototyping and educational purposes. I chose Ursina because It is an open source game engine, so I can read the source code on their website. In addition, it' s Python-based which allows me to combine the game and the socket part directly.

One of the difficulty to design this game is to construct complex 3D models such as trains, rail and barriers.Ursina provides a useful class "Entity" which can help users to consturct a 3D object easily with an obj file of a 3D model and a picture file of its texture. But it' s not easy to find the proper 3D model and its texture, most 3D models come without texture so I have to check many pages to finally find the 3D model I want. All the 3D models I apply in this game come from CGTrader.

3.3.2 Receive Data

Another difficulty is to receive data and run the game simultaneously without latency. My first thought is to put the receiving part under the update function, which will be implemented repeatedly while the program is running. But it becomes so delayed because the frame rate is so much higher than the data sending speed, so the game will pause until it receives the next data. To solve this problem, I import a module called "Threading" in python, Threading is used for creating and managing threads. It allows you to run multiple threads concurrently, and implement functions like Semaphore or Mutex. With this tool, I let the receiving part execute in the background, once it receives new data, the update function will update the motion of the character, if it receives nothing it won' t cause delay, the update function just updates other parts.

4 Outcome

The Parkour-like game we designed is easy to play and very interesting. To play the game, follow the "README.md" file in our GitHub repository to download. When playing, just move your STM32 board horizontally left or right, or vertically up or down, and the role in the game will do the corresponding motion. You will find that the delay of the game is very low, since we use WiFi instead of BLE to send data.

Though the game seems easy to design, but it isn't in fact, especially the motion detection algorithm. In the meanwhile, the calibration, data transfer and game server are all important, none is dispensable. If you want to see our demo, please refer to this link.

5 Work Distribution

1. 陳品翔
    - Motion detection algorithm (four in total)
    - Proposal
    - Progress Report
    - Demo
    - This report
2. 柯育杰
    - Data Filtering
    - Rotation Calibration
    - Data Transfer
    - Proposal
    - Progress Report
    - Demo
    - This report

3. 張梓安

    - Game design (the whole game server)
    - Proposal
    - Progress Report
    - Demo
    - This report

6 Reference

1. GitHub

    - Previous Work by Other Team

2. Course Handouts

    - STM32 IoT Node Onboard Sensors.pdf, Week 3
    - STM32 IoT node wifi Lab.pdf, Week 3
    - stm32 cubeide-freertos.pdf, Week 3
    - CubeIDE WiFi example lab.pdf, Week 8

3. Websites

    - How to perform motion sensing on STM32L4 IoTnode
    - Human Activity Recognition Colab
    - Ursina
    - CGTrader
    - Threading