CSCI 502 - Part 2: Open ended assignment

Madina Yergibay, Kanagat Kantoreyeva

***Abstract*—This is the second part of the Project 5, where we did task 1 and task 4. We studied an open-source IMU sensor measurement fusion algorithm, implemented the algorithm in Raspberry PI, cross-compiled QT application for RPI and run it on Raspberry Pi QEMU.**

I. TASK 1. IMU SIGNAL PROCESSING (10 POINTS)

A. Download and study the code

We downloaded and studied an open-source IMU sensor measurement fusion algorithm in C code developed by Sebastian Madgwick. IMUs are consisting of tri-axis gyroscopes and accelerometers. The algorithm uses quaternions for computations of the new entries of IMU device (Fig. 1).

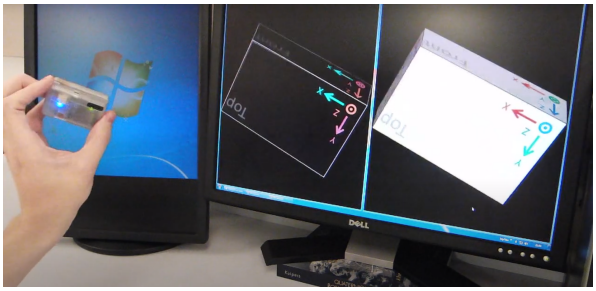

Figure 1. Sebastian Madgwick algorithm for IMU.

B. Run the algorithm

The algorithm was run on the Raspberry Pi. Firstly, the problem we faced was that the algorithm must be run in .c language, but the input values are in .mat file from MATLAB. We have tried different methods and came to the solution by converting the file with MATLAB code from .mat to .csv file that can be seen on Fig 2 (it was not available online):

```
disp(fieldnames(FileData));
dataMatrix = [FileData.Accelerometer, FileData.Gyroscope,
    FileData.Magnetometer, FileData.time];
writematrix(dataMatrix, 'imu.csv');

```

Figure 2. Conversion from .mat file to .csv.

Next, we needed to demonstrate the processing algorithm with example values in RPI. We wrote code based on the MadgwickAHRS.c and MadgwickAHRS.h architecture as you can see in Fig. 3.

```
#include "MadgwickAHRS.h"
#include <stdio.h>
#include <math.h>

int main() {
    FILE *file = fopen("imu.csv", "r");
    if (!file) {
        printf("Unable to open file\n");
        return 1;
    }
    float ax,ay,az,gx,gy,gz,mx,my,mz,time;
    while (fscanf(file, "%f,%f,%f,%f,%f,%f,%f,%f,%f,%f",
                  &ax, &ay, &az, &gx, &gy, &gz, &mx, &my, &mz,&time) == 10) {
        printf("%f,%f,%f,%f,%f,%f,%f,%f,%f,%f", ax, ay, az, gx, gy, gz, mx, my, mz);
        MadgwickAHRSupdate(gx,gy,gz,ax,ay,az,mx,my,mz);
    }
    fclose(file);
    return 0;
}
```

Figure 3. Test.c created to run the algorithm processing example data on QEMU Rasperry PI emulator and Real RPI.

We compiled with gcc and run the testing script on Real RPI (Fig. 4). The video of the task 1 is also attached to the submission.

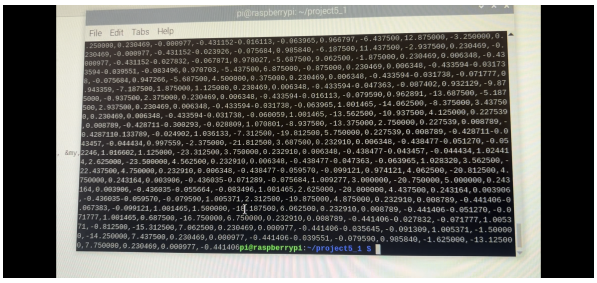

Figure 4. Test.c run the algorithm processing example data on Rasperry PI using C language.

II. TASK 4. CROSS-COMPILATION OF QT APP FOR RASPBERRY PI (60 POINTS)

For this task, we have used the Qt application from Project 1, a simple calculator, and the same zip folder that was created during Project 1. We unzipped the folder on Ubuntu and created repository named Pr5 on QEMU Raspberry Pi. We established ssh connection the same way as described in our Project 3 report by following the Chapter 2 of the ”Embedded Programming with Modern C++ Cookbook textbook” (Fig 5).

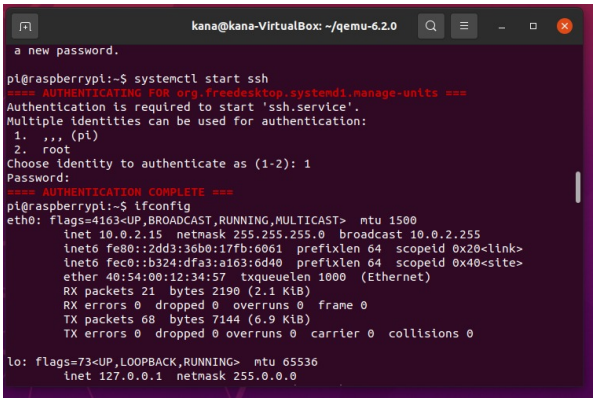

Figure 5. Establishing ssh connection.

Then we sent the folder file by file to QEMU Raspberry Pi through this ssh connection with docker container (Fig. 6).

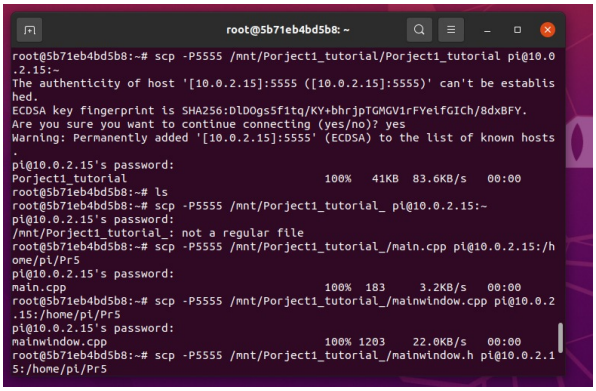

Figure 6. Sending the files.

After that, we installed cmake on QEMU (Fig. 7):

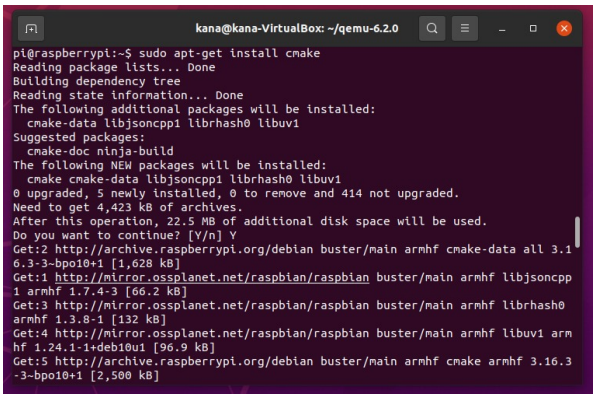

Figure 7. Installing cmake.

Then, we have created CMakeLists.txt with the following content(see Fig. 8).

```
cmake_minimum_required(VERSION 3.5.1)
project(Pr5)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTORCC ON)
find_package(Qt5 COMPONENTS Widgets REQUIRED)
add_executable(Porject1_tutorial main.cpp mainwindow.cpp)
target_link_libraries(Porject1_tutorial Qt5::Widgets)
```

Figure 8. CMakeLists.txt.

We had to also install qt dependencies in order for it to build (Fig. 9).

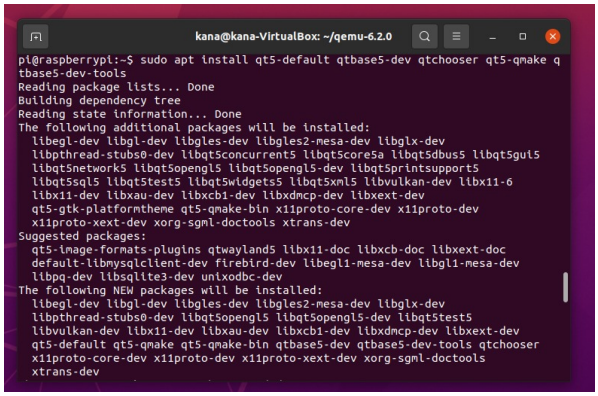

Figure 9. Installing qt dependencies.

And then install another dependency and build the project (Fig. 10).

```
sudo apt install libqt5widgets5
mkdir build && cd build && cmake ..
make
```

Figure 10. Build and cmake.

After all the described steps, the executable was created. When we tried running it, it showed an error that said it cannot connect to Display. After searching for solutions on the Internet, we found out it was because of the VirtualMachine. The solution described on the Internet suggested installing VcXsrv server on host Windows OS. After the installation, we typed command ”echo DISPLAY” on qemu, which returned nothing(Fig. 11).

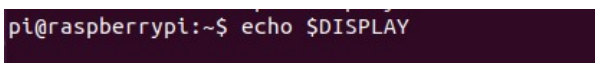

Figure 11. Returns nothing

We then used the commands ”export DISPLAY” and ”xhost” (Fig. 12) to add the correct display.

```
pi@raspberrypi:~$ export DISPLAY=192.168.56.1:0
pi@raspberrypi:~$ xhost +127.0.0.1
127.0.0.1 being added to access control list
pi@raspberrypi:~$ xhost +192.168.56.1
192.168.56.1 being added to access control list
pi@raspberrypi:~$ cd Pr5/build
pi@raspberrypi:~/Pr5/build$ ./Porject1_tutorial
```

Figure 12. Setting up correct display.

After this, we were all set, and finally run our executable, which appeared on our display on VcXsrv server (Fig. 13). The application was working correctly.

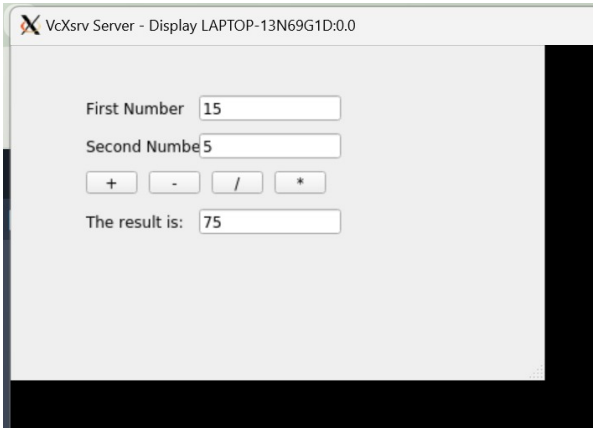

Figure 13. Qt app on QEMU Raspberry Pi

III. Conclusion

In the part 2 task 1 of this project, we have learned the quaternions, algorithm that uses them to calculate the IMU readings and run the example on RPI. The other part of the assignment taught us to be resilient to any circumstances and complete the project till the end: setting up emulation of RPI with QEMU, ssh connection, sending/receiving via ssh, installing dependencies, troubleshooting, etc.