

# 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).

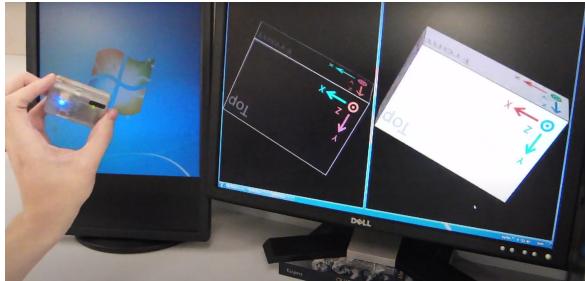


Figure 1. Sebastian Madgwick algorithm for IMU.

### 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):

```

1 disp(fieldnames(FileData));
2 dataMatrix = [FileData.Accelerometer, FileData.Gyroscope,
3 FileData.Magnetometer, FileData.time];
4 writematrix(dataMatrix, 'imu.csv');
5

```

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 Madgwick-AHRS.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",
                  &ax, &ay, &az, &gx, &gy, &gz, &mx, &my, &mz, &time) == 10) {
        printf("%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 Raspberry 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.

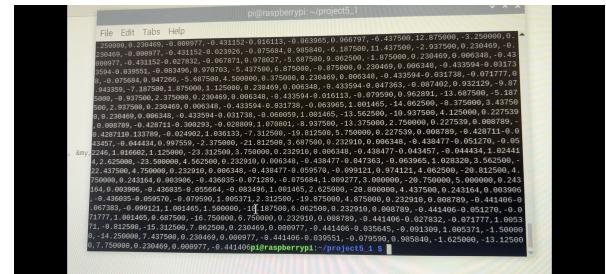


Figure 4. Test.c run the algorithm processing example data on Raspberry PI using C language.

## II. TASK 4. CROSS-COMPILE 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).

```

kana@kana-VirtualBox: ~/qemu-6.2.0
a new password.
pi@raspberrypi:~$ systemctl start ssh
==== AUTHENTICATING FOR org.freedesktop.systemd.manager-units ====
Authentication is required to start 'ssh.service'.
Multiple identities can be used for authentication:
1. ,,(p)
2. root
Choose identity to authenticate as (1-2): 1
Password:
==== AUTHENTICATION COMPLETE ====
pi@raspberrypi:~$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
        inet6 fe80::2dd3:36b0:17fb:6061 prefixlen 64 scoprid 0x20<link>
            fec0::b324:dfa3:1a16:3:6d40 prefixlen 64 scoprid 0x40<site>
    ether 00:54:5a:12:34:57 txqueuelen 1000 (Ethernet)
    RX packets 21 bytes 2190 (2.1 kB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 68 bytes 7144 (6.9 kB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0

```

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).

```

root@5b71eb4bd5b8:~# scp -P5555 /mnt/Project1Tutorial/Project1Tutorial pi@10.0.2.15:
The authenticity of host '[10.0.2.15]:5555 ([10.0.2.15]:5555)' can't be established.
ECDSA key fingerprint is SHA256:D1D0g5fItq/KY+bhrjpTGMW1rFYefiGICh8dxBFY.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[10.0.2.15]:5555' (ECDSA) to the list of known hosts.
pi@10.0.2.15's password:
Project1Tutorial                               100%   41KB  83.6KB/s  00:00
root@5b71eb4bd5b8:~# ls
root@5b71eb4bd5b8:~# scp -P5555 /mnt/Project1Tutorial/_pi@10.0.2.15:-
pi@10.0.2.15's password:
/mnt/Project1Tutorial/_pi@10.0.2.15: not a regular file
root@5b71eb4bd5b8:~# scp -P5555 /mnt/Project1Tutorial/_main.cpp pi@10.0.2.15:/h
ome/pi/Pr1S
pi@10.0.2.15's password:
main.cpp                                         100%  183     3.2KB/s  00:00
root@5b71eb4bd5b8:~# scp -P5555 /mnt/Project1Tutorial/_mainwindow.cpp pi@10.0.2.
15:/home/pi/Pr1S
mainwindow.cpp                                    100% 1203    22.0KB/s  00:00
root@5b71eb4bd5b8:~# scp -P5555 /mnt/Project1Tutorial/_mainwindow.h pi@10.0.2.1
5:/home/pi/Pr1S

```

Figure 6. Sending the files.

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

```

pi@raspberrypi:~$ sudo apt-get install cmake
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  cmake-data libjsoncpp1 librhash0 libuv1
Suggested packages:
  cmake-doc ninja-build
The following NEW packages will be installed:
  cmake cmake-data libjsoncpp1 librhash0 libuv1
0 upgraded, 5 newly installed, 0 to remove and 414 not upgraded.
Need to get 4,423 kB of archives.
After this operation, 22.5 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:2 http://archive.raspberrypi.org/debian buster/main armhf cmake-data all 3.1
6.3-3-bpo10+1 [1,628 kB]
Get:1 http://mirror.ossplanet.net/raspbian/buster/main armhf libjsoncpp1
armhf 1.7.4-3 [66.2 kB]
Get:3 http://mirror.ossplanet.net/raspbian/buster/main armhf librhash0
armhf 1.3.8-1 [132 kB]
Get:4 http://mirror.ossplanet.net/raspbian/buster/main armhf libuv1 arm
hf 1.24.1-1+deb10u1 [96.9 kB]
Get:5 http://archive.raspberrypi.org/debian buster/main armhf cmake armhf 3.16.
3-3-bpo10+1 [2,500 kB]

```

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_AUOTUIC ON)
set(CMAKE_AUTORCC ON)
find_package(Qt5 COMPONENTS Widgets REQUIRED)
add_executable(Projct1_tutorial main.cpp mainwindow.cpp)
target_link_libraries(Projct1_tutorial Qt5::Widgets)

```

Figure 8. CMakeLists.txt.

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

```

kana@kana-VirtualBox: ~/qemu-6.2.0
pi@raspberrypi:~$ sudo apt install qt5-default qtbase5-dev qtchooser qt5-qmake q
tbase5-dev-tools
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libegl-dev libgles-dev libgles2-mesa-dev libglx-dev
  libgbptread-stubs0-dev libgbpt5currents libgbt5core5a libgbt5dbuss libgbt5gu
  libgbt5networks libgbt5openpgl5 libgbt5openpgl5-dev libgbt5printsupport5
  libgbt5sql5 libgbt5stest5 libgbt5wldget5 libgbt5xml5 libvulkan-dev libx11-6
  libx11-dev libxvau-dev libxcb1-dev libxdmcp-dev libxext-dev
  qt5-gtk-platformtheme qt5-qmake-bin x11proto-core-dev x11proto-dev
  x11proto-xext-dev xorg-sgml-doctools xtrans-dev
Suggested packages:
  qt5-image-formats-plugins qtwayland libx11-doc libgbt5dbuss libgbt5gu libgbt5wldget5 libgbt5xml5 libvulkan-dev libx11-6 libxvau-dev libxcb1-dev libxdmcp-dev libxext-dev
  libgbt5printsupport5 libgbt5stest5 libgbt5wldget5 libgbt5xml5 libvulkan-dev libx11-6 libxvau-dev libxcb1-dev libxdmcp-dev libxext-dev
  qt5-default qt5-qmake qt5-qmake-bin qbase5-dev qbase5-dev-tools qtchooser
  x11proto-core-dev x11proto-dev x11proto-xext-dev xorg-sgml-doctools
  xtrans-dev
The following NEW packages will be installed:
  libgbt5dbuss libgbt5gu libgbt5wldget5 libgbt5xml5 libvulkan-dev libx11-6 libxvau-dev libxcb1-dev libxdmcp-dev libxext-dev
  libgbt5printsupport5 libgbt5stest5 libgbt5wldget5 libgbt5xml5 libvulkan-dev libx11-6 libxvau-dev libxcb1-dev libxdmcp-dev libxext-dev
  qt5-default qt5-qmake qt5-qmake-bin qbase5-dev qbase5-dev-tools qtchooser
  x11proto-core-dev x11proto-dev x11proto-xext-dev xorg-sgml-doctools
  xtrans-dev

```

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).

```

pi@raspberrypi:~$ echo $DISPLAY

```

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$ ./Project1_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.

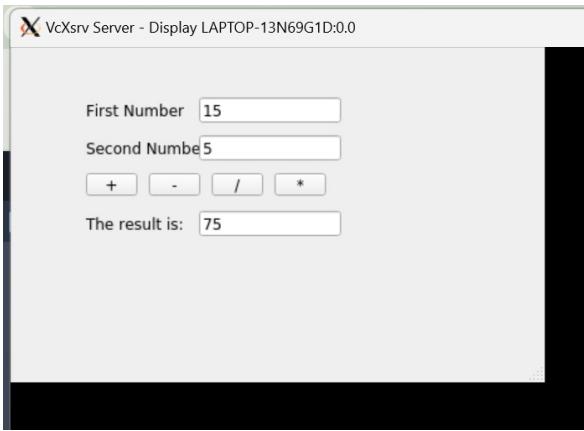


Figure 13. Qt app on QEMU Raspberry Pi

### 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.