

Documentation of the Wireless IMU setup

Biosignals and medical electronics

Module 8 CreaTe



release may 17th 2021; v2

Preface	3
Introduction	3
Setup	4
Wireless IMU block	5
Connecting to the block using Python	8
Python	8
Setting up Python	8
Python code step by step explanation	9
Data	10
Using an IMU	10
Explanation data	11
Filtering	11
Save to CSV	13
Appendix	14
A Bluetooth addresses	14

Preface

This setup and documentation is prepared for the 2021 version of the course Biosignals and Medical Electronics for the 8th module of the bachelors Creative Technology. This ip is developed and created during q3 and q4 of the academic year 2020-2021, by Jonathan Matarazzi, Joep Blanksma and Stijn Slebos. The setup and execution of the 2021 version of the course is managed by Jasper Reenalda, Erik Faber, Bouke Scheltinga and Robbert van Middelaar.

Because this is a new setup, this document remains a working version and will be updated during the span of the course; to provide more information based on the demands discovered. Students are therefore also asked to share their opinions, ideas and concerns to facilitate the development of this document and of the setup.

Since discord is this course's main means of communication: questions regarding the setup can be redirected to the dedicated channel. The github repository can be used to issue specific opinions, ideas and concerns.

Introduction

For the project of the course Biosignals and medical electronics you are asked to build a setup to provide live feedback on running movement. For this project you are given two wireless IMU setups. These setups can be used to gather raw accelerometer and gyroscope data. This documentation contains a description of the setup and a start-up guide, to get you started with the setups. The IMU-setups will be provided with code loaded on them, so they will work autonomously both wired and wireless. A description of the full setup is however still provided and the program can also be found in the github repository. The wireless connection of the setup uses bluetooth to transfer data. A python template to receive and interpret the data is provided in this guide (and code can be found in the repository).

Setup

The setup that you will use consists of two parts. Two wireless IMU's to obtain data and python code to receive this data on your laptop. The two IMU's can be connected to your laptop via bluetooth or cable. How to set this up is explained in the installation section.

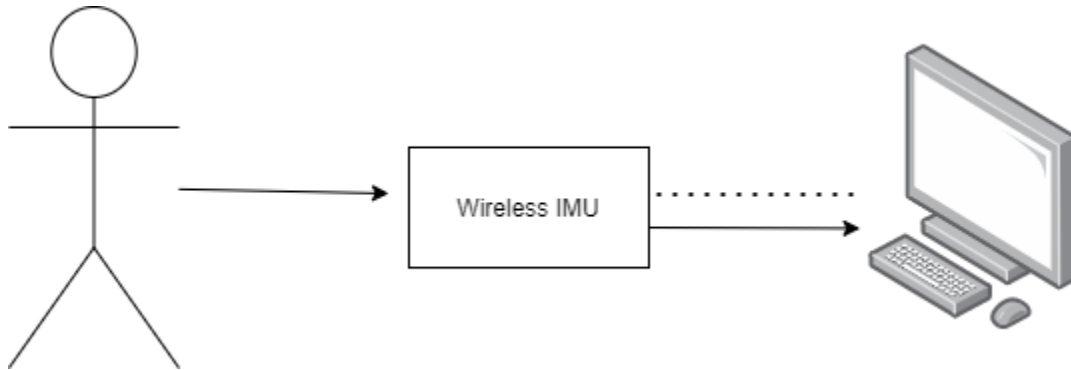


Figure 1: Overview of connections. Solid line is a physical connection and the dotted line is a bluetooth connection

The connection between the subject (human) and the Wireless IMU is physical, this means that the casing of the IMU is attached directly to the skin/person. How this is done is up to you but keep in mind that casing should be rigidly attached to reflect the body's movement.

The connection between the Wireless IMU and the laptop has two options, bluetooth and wired. Both are already set up in the code so an easy switch can be made. A wireless connection probably makes testing more easy.

Wireless IMU block

The IMU block you receive contains an MPU5060 IMU. This can be used to extract gyroscope and accelerometer data.

Hardware

The [setup](#) contains the following elements:

- ❑ GY-521 (datasheet, [mpu5060](#))
- ❑ Wemos LOLIN32
- ❑ Li-Po Accu 3.7V 1250mAh
- ❑ Switch Toggle SPDT 3
- ❑ Push-button
- ❑ 3mm led Bi-Color; common anode

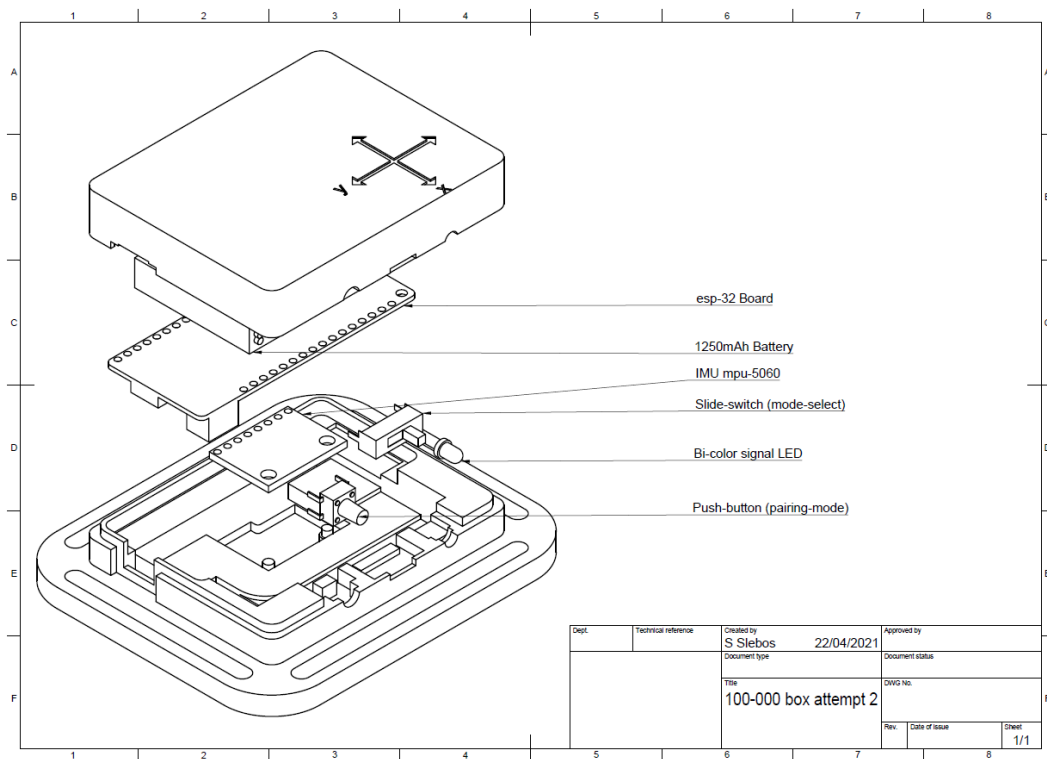


Figure 2: Overview of the hardware

The connections in the block are shown next.

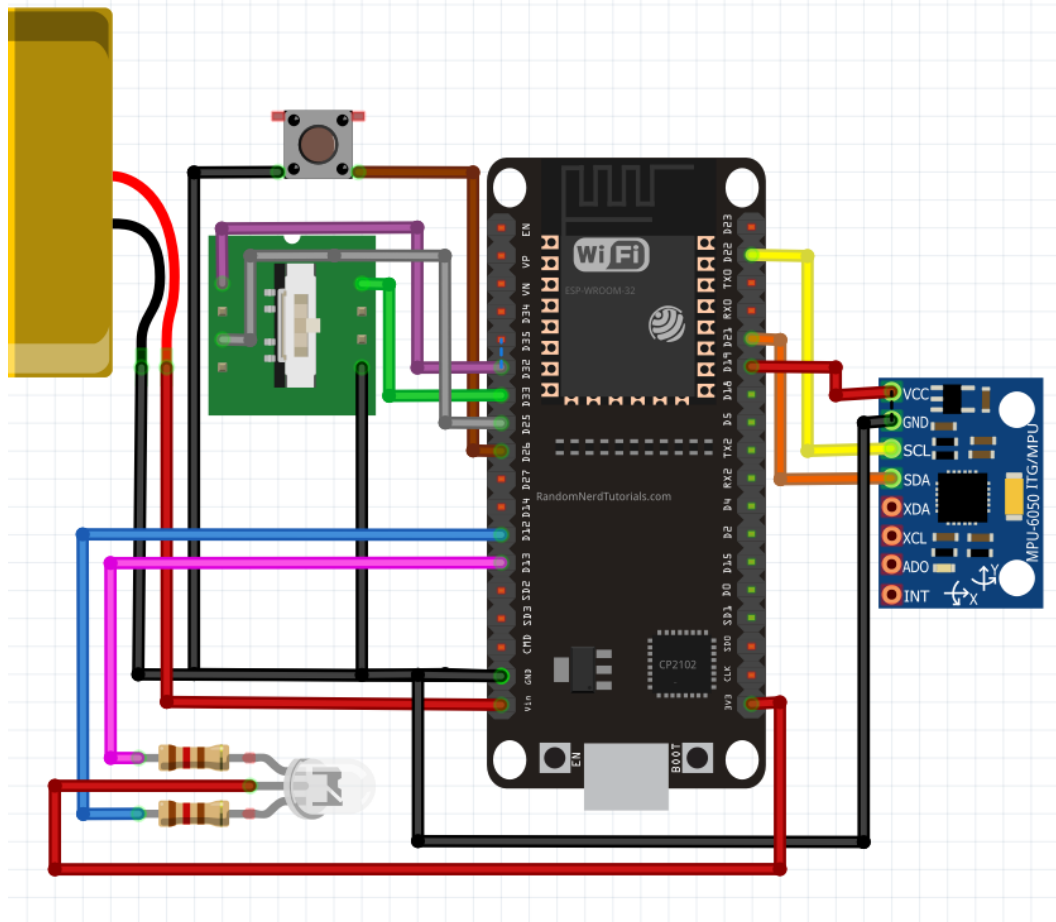


Figure 3: How the parts inside the WirelessIMU are connected. (The ESP32 inside the WirelessIMU has a deviating pin layout but the pin numbers used in the schematic do match the used pins)

Software

The full repository of this project can be found here:

https://github.com/StijnSlebos/Smart_Technology_BME_M8

The software written for the Wemos Lolin32 board uses I2C to request and receive data from the gy-521 board. The data is then depending on the mode, Serial-wired or Wireless transmitted. Transmission is done in a bitstring. This string should be decoupled in order to decode the information (See in python decoding).

Use of the block

Modes

The IMU block has three stable modes: A deepsleep/off mode, a wired mode and a wireless/bt mode. The block is equipped with a slide switch to toggle between the different modes. In order to connect with the device, the BT-pairing button can be pushed to switch temporarily to a pairing mode, to connect with the device. Lastly the block is equipped with a signal led, to show in which mode the block is in.

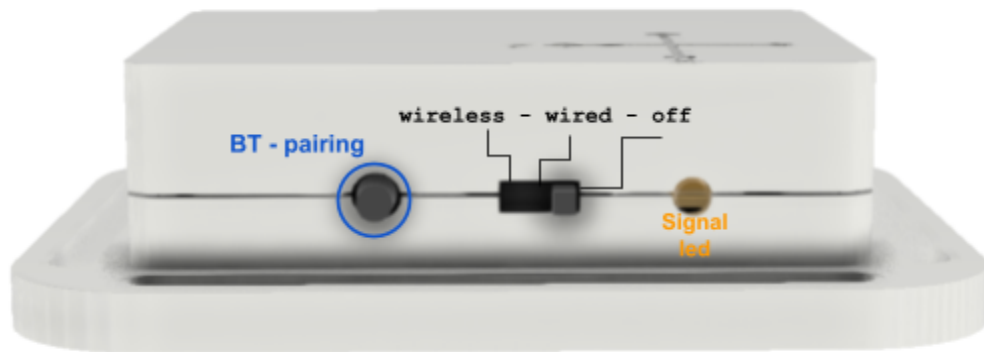


Figure 4: Layout of the button and switch on the casing.

Deep Sleep/off mode

When the device is in off-mode, all power is cut and the device is set into a deep sleep mode. This means that all functionality is dropped and only the RTC is still powered.

Wired mode

When the device is in the wired-mode, a cable can be plugged into the device to read out a serial bit stream. The led will be red if the system is in wired mode.

Wireless mode

In the wireless mode, the block publishes data to a bluetooth serial connection. In order to connect to the block, the pairing button can be pressed, after which the esp-board will be visible for BT-devices for 30 seconds. Every block has a unique bluetooth address based on its mac-address. A list with all bt-addresses will be made available (see appendix). The led will be blue if the system is in wireless mode.

Pairing mode

If you press the pairing button the pairing mode will be activated, this will allow you to connect to the Wireless IMU. The led will blink blue if the system is in pairing mode.

Charging

The Wemos LOLIN32 board has a charging controller on-board implemented. In every mode, the battery on board can be charged by connecting a usb cable to the board. When collecting the block, it could be that the battery is not yet connected to the board. In this case you can open the case and connect the battery: *please check if you connect the positive and negative pole correctly.*

Connecting

For optimal connection with the IMU you can connect the IMU first with your computer via normal bluetooth settings using the pairing button. Now you do not need to press the pairing mode when you run your python script. If you can not connect your IMU to your computer using normal bluetooth settings, then pressing the pairing button once every time you start the python script will work as well.

Connecting to the block using Python

For the setup to work correctly some steps need to be taken regarding the software. Below you will find detailed instructions about where to find the software and how to install the necessary libraries. Necessary code is available on this github repository: [Github link](#) (same link as given in the software section)

Python

Setting up Python

Step 1: Get the correct code from github (clone or download)

We advise you to work in a pipenv environment. To set up a pipenv environment we refer to the programming manual of module 6 or the internet. (in pycharm this is quite easy to do)

Step 2: Make sure you have python 3.8 installed and run the program on python 3.8, another version of python will not work. Your pipenv environment does not need to be 3.8

Step 3: Open the connect.py in python (the example pictures are in pycharm but any other python IDE should be fine)

Your project bar on the left can look like:

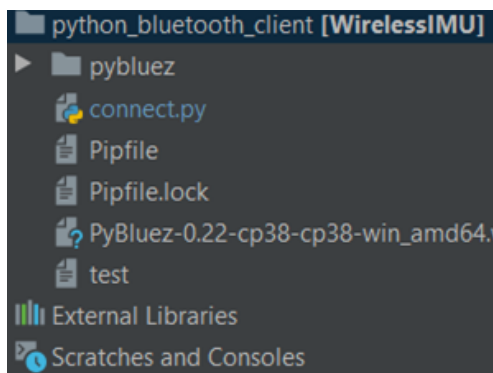


Figure 5: Project bar after opening connect.py in pycharm.

Step 4a: Installation of the pybluez library, for correct installation you make sure that the pybluez file is in the root folder. If you made sure it is there then go to the terminal/powershell in the directory (in pycharm this is found in the bottom layer of your screen otherwise, you can use

cd to navigate to your work folder)

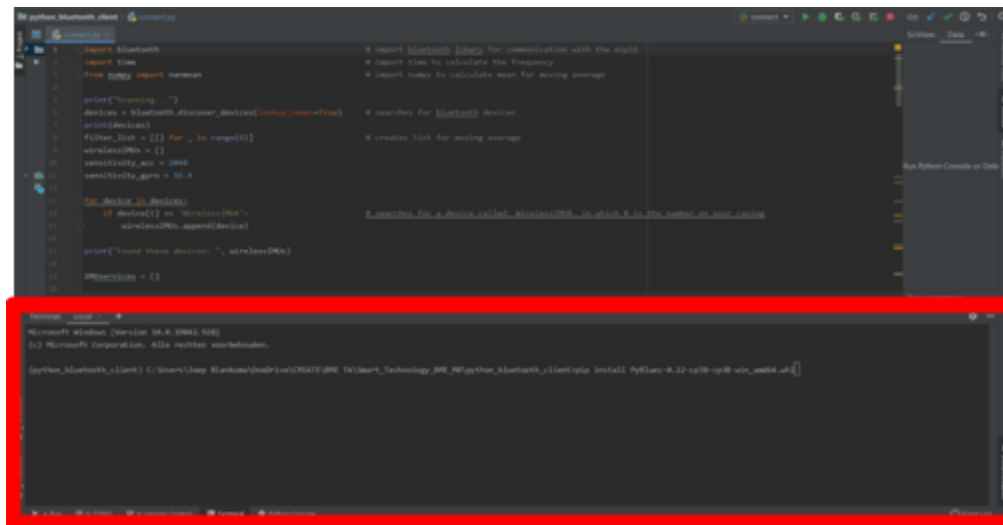


Figure 6: Location of the terminal

Step 4b: Type into the terminal: (use tab, do not copy)

```
pip install PyBluez-0.22-cp38-cp38-win_amd64.whl
```

Now the necessary library should be installed.

Step 5: Installation of the numpy library

Type into the terminal:

```
pip install numpy
```

Step 6: Correcting the default name of the bluetooth devices. By default the name here will be 'WirelessIMU-'. But every WirelessIMU will have its own unique name corresponding with the casing number. So if you have WirelessIMU 1 the name you will need to give can be found in appendix A.

```
for device in devices:
    if device[1] == 'WirelessIMU-':
        wirelessIMUs.append(device)
```

Figure 7: Part of the code where a device is named so a sole connection can be established.

```
for device in devices:
    if device[1] == 'WirelessIMU-67A4':
        wirelessIMUs.append(device)
```

Figure 8: Example of correct naming in case of having a connection to case 15.

Step 7: The python code should be ready to go.

If your code is not ready to go consult your peers and/or contact us on discord.

Python code step by step explanation

To make; please let us know if this is something you miss or need.

Data

Using an IMU

Understanding IMU (inertial measurement unit) data starts with understanding how an IMU works. The IMU has two distinct parts, the accelerometer and the gyroscope, and therefore also measures two distinct values. So in essence the IMU merges two sensors into one device.

The accelerometer will measure the acceleration in g (m/s^2). 1g is equal to 9.81 m/s^2 which you can recognize as the gravitational acceleration. If you put the IMU on a flat surface the values will read: x: 0g, y: 0g, z: 1g. This is because the accelerometer also measures gravity. If the gravity component is neglected then the workings of the accelerometer are well explained in figure xx.

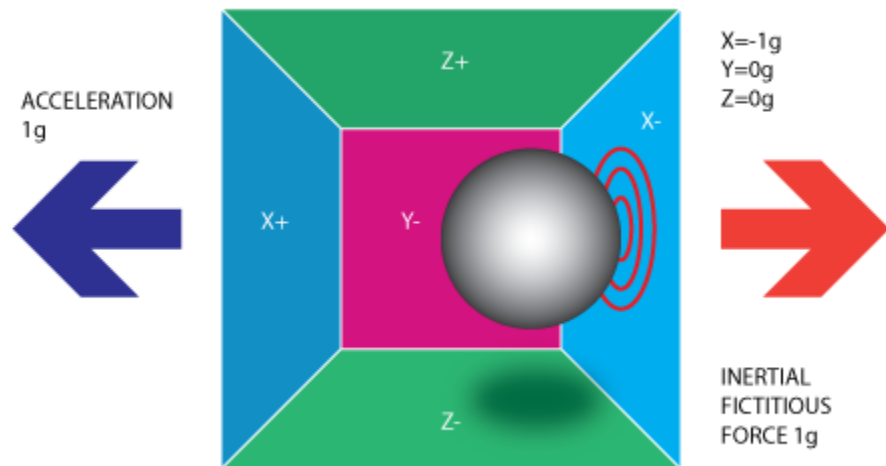


Figure 9: Visualization of the workings of an accelerometer (source: http://www.starlino.com/imu_guide.html)

In figure 9 the accelerometer is moved to the left and because a weight in the accelerometer now puts pressure on the right side the accelerometer knows that there is an acceleration to the left side

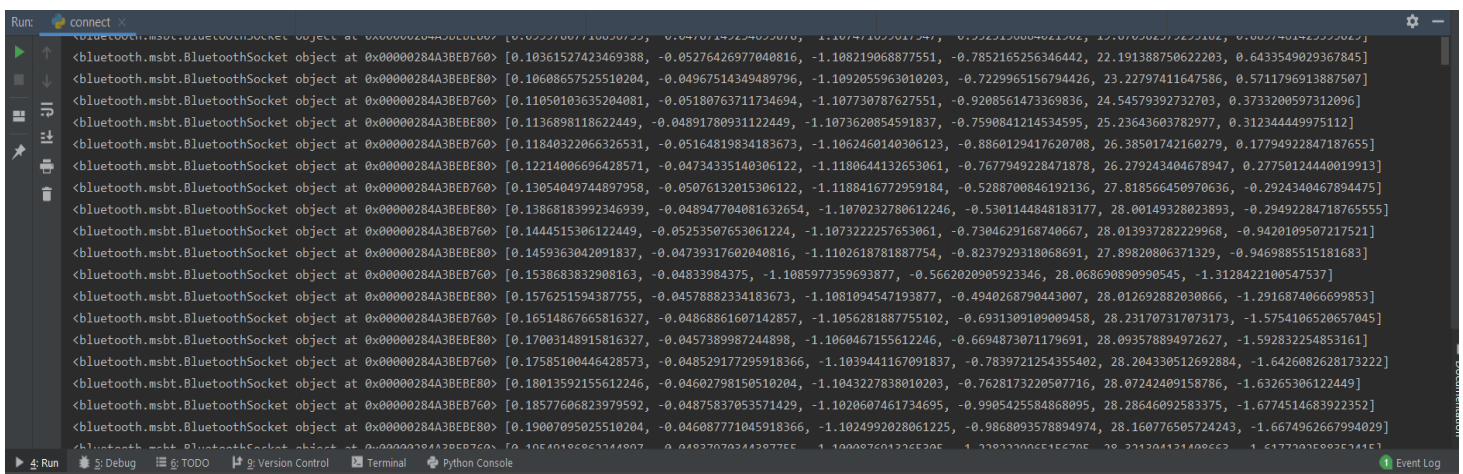
The gyroscope will measure the rotation in deg/s. This is done around the x, y and z axis. If you put the gyroscope on a flat surface the values will read: x: 0 deg/s, y: 0 deg/s, z: 0 deg/s. Pay attention to the fact that if the gyroscope reads 2000 deg/s it has not actually turned 2000 degrees but 20 deg if the frequency of the measurements is 100 Hz.

Explanation data

The MPU5060 is a chip with an accelerometer and gyroscope. The chip outputs this data on the bounds of a 16bit integer. This is sent in a bitstring over either the serial or the bluetooth serial.

The python code will receive the data from the Wireless IMU and transfer it into integers with a range of -32678 to 32678. This range is just a scale and so has no value without knowing the sensitivity and range of the IMU. In our case the range of the accelerometer is -16g to 16g and the range of the gyroscope is -2000 deg/s to 2000 deg/s. So a value of -32678 relates to -16g or -2000 deg/s. The sensitivity of our accelerometer is 2048 LSB/g which means that a difference of 2048 in the raw values means a change of 1g, for the gyroscope the sensitivity is 16.4 LSB/deg/s.

An example of the data you should receive in your python code is:



```
Run: connect x
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEB760> [0.10361527423469388, -0.05276426977040816, -1.108219068877551, -0.785216525634442, 22.191388750622203, 0.6433549029367845]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEBE80> [0.10608657525510204, -0.04967514349489796, -1.1092855963010203, -0.7229965156794426, 23.22797411647586, 0.5711796913887507]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEB760> [0.11050103635204081, -0.05180763711734694, -1.107730787627551, -0.9208561473369836, 24.54579392732703, 0.3733200597312096]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEBE80> [0.1136898118622449, -0.04891780931122449, -1.1073620854591837, -0.7590841214534595, 25.23643603782977, 0.312344449975112]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEB760> [0.11840322066326531, -0.05164819834183673, -1.1062460140306123, -0.8860129417620708, 26.38501742160279, 0.17794922847187655]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEBE80> [0.12214006696428571, -0.04734335140306122, -1.1180644132653061, -0.7677949228471878, 26.279243404678947, 0.27750124440019913]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEB760> [0.130540497444897958, -0.05076132015306122, -1.1188416772959184, -0.5288700846192136, 27.818566450970636, -0.2924340467894475]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEBE80> [0.13868183992346939, -0.048947704081632654, -1.1070232780612246, -0.5301144848183177, 28.00149328023893, -0.2949228471876555]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEB760> [0.1444515306122449, -0.05253507653061224, -1.107322257653061, -0.7304629168740667, 28.013937282229968, -0.9420189507217521]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEBE80> [0.1459363042091837, -0.04739317602040816, -1.1102618781887754, -0.8237929318068691, 27.89820806371329, -0.946885515181683]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEB760> [0.1538683832908163, -0.04833984375, -1.1085977359693877, -0.5662020905923346, 28.06869089090545, -1.3128422100547537]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEBE80> [0.1576251594387755, -0.04578882334183673, -1.1081094547193877, -0.4940268790443007, 28.012692882030866, -1.2916874066699853]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEB760> [0.16514867665816327, -0.04868861607142857, -1.1056281887755102, -0.6931309109009458, 28.231707317073173, -1.5754106520657045]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEBE80> [0.17003148915816327, -0.0457389897244898, -1.1060467155612246, -0.6694873071179691, 28.093578894972627, -1.592832254853161]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEB760> [0.17585100446428573, -0.048529177295918366, -1.1039441167091837, -0.7839721254355402, 28.204330512692884, -1.6426082628173222]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEBE80> [0.18013592155612246, -0.04602798150510204, -1.1043227838010203, -0.7628173220507716, 28.07242409158786, -1.63265306122449]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEB760> [0.18577606823979592, -0.04875837053571429, -1.1020607461734695, -0.9905425584880895, 28.28646092583375, -1.6774514683922352]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEBE80> [0.19007095025510204, -0.046087771045918366, -1.1024992028061225, -0.9868093578894974, 28.160776505724243, -1.6674962667994029]
<bluetooth.msb.BluetoothSocket object at 0x0000284A3BEB760> [0.19540186863344807, -0.04837070244428775, -1.1009076012365705, -1.320320065166705, 28.23120412140862, -1.6177307509252415]
Run Debug TODO Version Control Terminal Python Console Event Log
```

Figure 10: an example of how the data is retrieved with python (two WirelessIMU's).

From left to right the data represents:

accX accY accZ gyroX gyroY gyroZ

After applying code explained in the filtering section the accelerometer data will be in g (m/s²) and the gyroscope data will be in deg/s.

Filtering

The data that is coming from the IMU is first going through two functions before it ends up in output_real. Those functions are moving_average and real_numbers.

The moving_average function does, as the name suggests, apply a linear moving average to the data. This means that the function stores N number of data points then takes the mean of those N points and returns the mean to the output. If a new number is added to the list of N data

points the oldest one is removed from the list so a new average is calculated. In practise this results in a very simple low pass filter. The default code will have N set to 1, so no filter is applied. It is up to you to find a value of N that fits your application.

The `real_numbers` function transfers the data from the `moving_average` function into real numbers. So the IMU gives values between -32678 and 32678 and the `real_numbers` function transfers this into -16g to 16g and -2000 deg/s to 2000 deg/s. This is done by dividing the data by the sensitivity that is found in the data sheet.

Save to CSV

It is possible to save your data to a CSV file so you can analyse the data without having a Wireless IMU connected. In this section it will be explained how it is done and how you can choose how many data points you want.

```
if cycles == 1000:  
    np.savetxt("data.csv", csv_list_transpose, delimiter=",", fmt='%s')  
    print("adding to csv is done")
```

Cycles is a variable that increases by 1 every iteration, so 1 cycle is equal to 1 datapoint. If you set cycles == 1000 you will receive a csv file that has 1000 data points. If you need more or less for your measurement you can alter this number.

If you want to create a file that is not called data.csv just change the name to anything you want and it will create a new file with the data.

Excel

For proper export to excel you will need to follow the steps listed below.

Step 1: Open an empty excel file.

Step 2: Select data in the top bar.

Step 3: Go to get external data on the left side.

Step 4: Select from file > from text/CSV > Select the data file that python created

You can now plot and analyse your data.

Appendix

A Bluetooth addresses

Device number	Bluetooth address
1	WirelessIMU-562A
2	WirelessIMU-CF0E
3	WirelessIMU-6642
4	WirelessIMU-7A0A
5	WirelessIMU-EF62
6	WirelessIMU-8732
7	WirelessIMU-F94E
8	WirelessIMU-250A
9	WirelessIMU-890E
10	WirelessIMU-FD92
11	WirelessIMU-FEA6
12	WirelessIMU-F672
13	WirelessIMU-7786
14	WirelessIMU-0BE6

Table 1: Bluetooth addresses of the WirelessIMU's