**The instructions provided by Dr. Aggravi and Dr. Pacchierotti, for a proper setup and use of the armband devices, mainly refer for usage in Linux systems, while we must necessarily work in Windows system (see Framework requirements file).**

**This should not be a great issue, as the devices can properly work in both systems, as specified by Dr. Aggravi, whose comments are reported below. So, please read carefully this file to check what is actually requested to do on our side to make armbands work properly.**

**Dr. Aggravi’s comments about Windows setup**

*“The bracelets works perfectly in Windows too (actually, they work even better than in Linux, as the Windows Bluetooth layer is more stable…). There are three main differences:*

1. *Pairing in Windows is done automatically by the system (as far as I remember, there is no pin to communicate);*
2. *the device connecting port is hard to be found. While Linux clearly notifies which port your device is plugged to after connecting, Windows somehow “hides” this information. The best way to find the correct port is to check the list of COM ports before connection and check how the list changed after.*
3. *Windows connection is more stable.*

*The provided C++ classes should work seamlessly in Windows too (indeed, there should be some #define macro detecting the proper OS inside).*

*The general idea is to create an instance of HapticDevice and then use the functions startCommunication(const char\* serialport), closeCommunication() e sendSignal(int frequency,char interval) to communicate with the device.*

*It is possible that the header files are slightly outdated with respect to the source files and a couple of lines may need to be modified. The devices are expected to send/get 6 bytes on the serial port and not 4, as in a former version of the code. If you see anything weird while using them, do not hesitate to contact me and I will fix this.”*

***---------------- vvv Check if needed as most instructions are Linux-based vvv ----------------***

*Please find attached a zip files with all you need for working with our wearable devices. These files are useful if you are using Ubuntu; the guide for Windows or MacOs is not ready yet. In case you fall in the second category, please contact me for more details.*

*In details, the zip file contains:*

*1. a video that shows how to pair a device on Ubuntu 14.04; for 16.04, the process is similar. I haven't tested yet for 18.04.*

*2. the source of the a wiki guide for discovering, pairing, connecting, and communicating to the device from command line;*

*3. screenshots of the same guide;*

*4. a ROS node for communicating with the device (since it is using standard classes, any ROS distribution should work);*

*5. the Arduino code inside the device. This code is useful if what I wrote below is not clear;*

*6. two C classes for connecting and communicating with the device (to be included in your code; already included in the ROS node).*

*NOTE: In the wiki guide (points 2 and 3), the communication is done with an old protocol, who waits for four bytes; Please see below for the updated version (with six bytes).*

*The device at the moment expects six bytes, e.g., SffffE.*

*The first and the last byte must be a capital S (83 in decimal, 53 in hexadecimal) and a capital E (69 in decimal, 45 in hexadecimal).*

*The four bytes between S and E are the frequencies at which you want the motors to vibrate. In your version, each motor should be controllable independently.*

*When a motor is activated, it will vibrate at the commanded vibration until a different value is received (to stop the motor a value lower than 32 is required [32 is the equivalent of the space bar in ASCII]).*

*For example, if you send the device the following six bytes SaaaaE (83 97 97 97 97 69 in ASCII), motors 1 to 4 will vibrate at around 106 Hz (the motors have a vibration frequency between 0 and 280 Hz, but we control them with bytes between 0 and 255), until you stop them (for example S E will shut down the vibration).*