Note: We are only using 13 channels. Channels 6,7,11 are not working. If all channels are working for you some changes need to be done.

1B Helmet to OpenBCI connection

OpenBCI GUI is used [https://docs.openbci.com/Software/OpenBCISoftware/GUIDocs/] to connect between the helmet and computer. The connection used is via dongle which sends data via BLE with 125Hz frequency. Choose "cyton(live)" from the menu, "Serial(from dongle)", change channel count to 16 channels and press "Auto-connect".

1B EEG-Helmet

The Helmet can be a bit tricky to get on correctly. But fortunately there is a few tricks. The challenge is to have good connectivity from the channels. Using the widget "cyton signal" can be used to check impedance to the different channels, lower impedance is wanted. Most important is the "Time series" widget were one can see the percentage of how close to railed it is or how noisy the signal is. If either the impedance is to high, a channel has to much noise or is railed one has to acknowledge the channel/s. For the bad channels try disconnect screw harder and connect them again. Try to remove as much hair in the way as possible. Sometimes removing the headpiece getting a better fit might be needed. When all channels have good values your ready for next step.

1B OpenBCI

Familiarise with the different widgets in the program. The OpenBCI GUI has a network widget were one can choose which communication protocol to send the data too. The LSL protocol had the least package loss, to use LSL follow the Method:2 in [https://docs.openbci.com/Software/CompatibleThirdPartySoftware/Matlab/]

to download the libraries needed. To stream via LSL choose what to stream, preferably "Time series" then the type needs to be configured. If two or more computers stream LSL simultaneously with same type the receiver will get all data from all computers such that an unique type could be used, "RevieveData.m" uses "PUKK" as type. OpenBCI GUI automatically saves down the recordings to text files for offline examination located in a "recordings" in OpenBCI GUI folder.

1B MATLAB

Firstly the program reads the trained matrixes "w.mat", "mdl.mat" and "bp-  
Filt.mat" such these locations need to be loaded correctly. The helmet from  
beginning had 16-channels during the project three channels were broken. In  
the MATLAB code "RecieveData.m" under the while loop the three broken  
channels are removed and channel 11 and 9 is changed. Channel 9 were on top  
of the cortex such chnnel 11 that is on the backside could read better values  
working as channel 9 instead. Configure these depending on the amount of elec-  
trodes used and placed. Run MATLAB if you see the text "Resolving an EEG  
stream..." it’s waiting for a stream from OpenBCI. Press start LSL stream in  
OpenBCI GUI. The text "Opening an inlet..." and "Now receiving data..." will  
be displayed if done correctly(connection is now established). The stream from  
OpenBCi GUI can now be started and the data will be read and stored in the  
"vec" variable for each new sample. If the system also should stream data over  
to comp three the IP needs to be configured(IP from the computer that will  
receive data). Port chosen were 1337 which also needs to be configured to be  
the same on both computers.

# 2B Offline

1. Run fixData.m. Change the load names to the filenames you are using. Be sure that the protocol is changing between relaxing and gripping each 6 seconds. Otherwise extractTrials.m need to be changes or another solution needs to be implemented. The output from fixData will result in two classes containing [samples X channels X trials].
2. Save the two classes as .mat files.
3. Run main.m, this will produce a model called Mdl and a spatial filter called W.
4. Type save(‘Mdl.mat’); in command window.
5. Type save(‘W.mat’); in command window.

# 2B Online

1. The saved variables Mdl.mat and W.mat needs to be sent to the Online file
2. Change the IP address and port in the UDP part of the code.

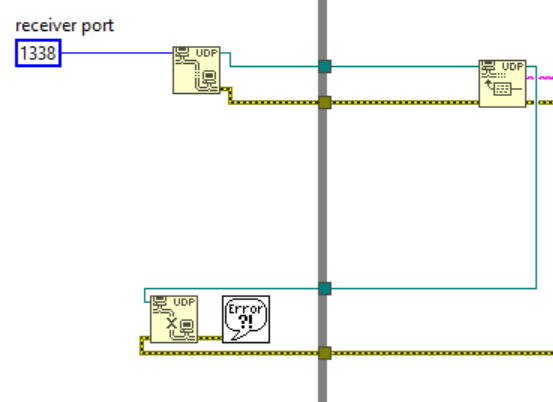
**Getting started guide: Component 3 – Robotic Hand**

Getting familiar with the LabVIEW software

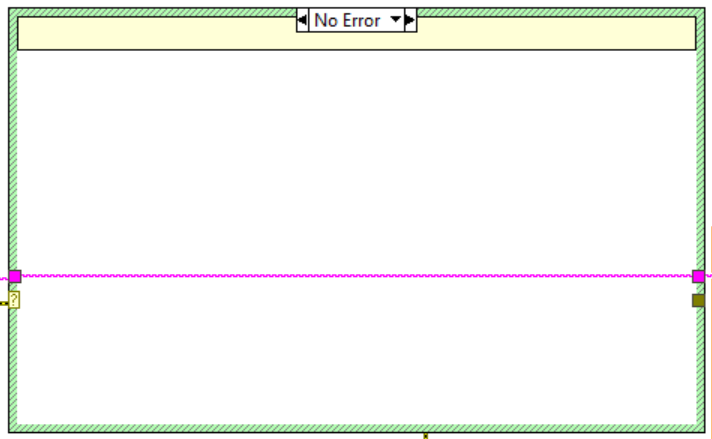
1. Main.vi

This is the complete LabView VI to receive data from Component 2, handle that data and send it to the RoboRIO.

* UDP – Receiving data (block diagram)

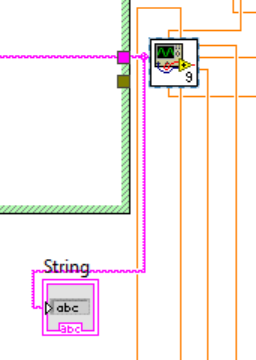


First receiving UDP messages.

En bild som visar text

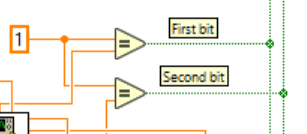
Automatiskt genererad beskrivning

Error handling of these messages. Error if no message available.



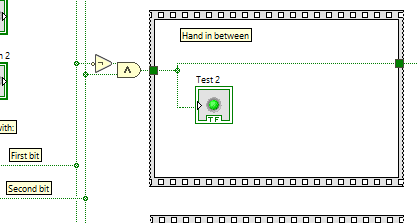
After receiving data, conversions to double are made.

* Handling data part 1 (block diagram)



After receiving the first two bits, they are converted from Double (orange) to Boolean (green) as the figure above shows.

* Handle data part 2 (block diagram)



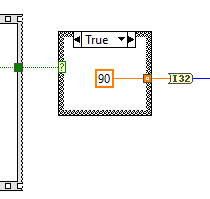
The figure above shows what would happen with command “Hand in between”, i.e. with first two bits equal “01” (see table below). The “Test 2” indicator confirms that this is the case.

The two bits of data received from Component 2 via UDP decides the following:

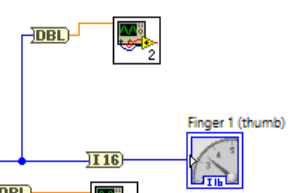
|  |  |  |
| --- | --- | --- |
| First bit | Second bit | Command |
| 0 | 0 | “Hand open” |
| 0 | 1 | “Hand in between” |
| 1 | 0 | “Hand closed” |
| 1 | 1 | “Individual fingers” |

Where “Hand open” means 0 degrees of flexion, “Hand in between” about 90 degrees and “Hand closed” about 180 degrees. This is for all fingers. Then there is “Individual fingers” that allows the user to move one or more fingers at a time to states “open” or “closed” with the last 5 bits in a message (a message has 2+5=7 bits).

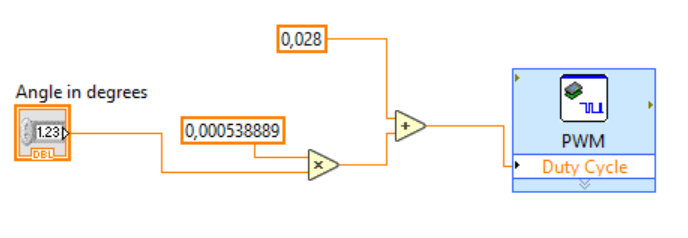
* Handle data part 3 (block diagram)



The received command then specifies how fingers should move, as the figure above shows. Note that this is not the final angle but describes about what angle represents a state. 90 represents “in between”.



The data about each finger is multiplied and added with constants to translate the angle to a PWM number that will eventually move the servo(s) the desired angle. These calculations are made in subVIs for each finger, the figure above is displaying that subVI for thumb + indicator for graphical representation/verification of the thumb movement. This is then sent to RoboRIO in that subVI, as shown in figure below.



1. All hand commands template.vi

There is also an “All hand commands template.vi”. This VI demonstrates how the system (Main.vi) works in theory. Instead of receiving messages from Component 2 and displaying these with the RoboRIO, there are buttons and indicators.

Quick Startup Guide

1. Install LabVIEW 2016 32-bit with roboRIO toolkit and drivers from the DVDs included with the roboRIO
2. Install Python 3.x
3. Plugin the roboRIO: USB, external power and servos. Thumb is PWM0, index finger is PWM1… Be careful and make sure that the servos are not plugged in the wrong way, there is no reverse polarity protection. Black is ground. Plug in the servos before connecting the roboRIO to the computer or external power.
4. Download LabVIEW and Python files from <https://github.com/Emanuel-Bjurhager/robot_hand/>
5. Edit the file called ”UDP relay.py”. Make sure the roboRIO IP variable “IP” is correct  
   Text

   Description automatically generated with low confidence
6. In the file “UDP relay.py” change the variable “UDP\_IP” to your computer IP address (you can find your IP using the command “ipconfig” in CMD) and save the fileA picture containing text

   Description automatically generated
7. Run the Python code using the CMD “ python ‘.\UDP relay.py’ ”
8. Open the LabVIEW project by double clicking the file called “Neuroteknik projekt.lvproj”.
9. In the LabVIEW project, open and run the filed called Main.vi
10. The hand should now move when commands are sent to your computers IP and port 1337. If it does not work, try disabling Windows Firewall.