MONASH University

ECE4095 TECHNICAL MANUAL- PROJECT B

Electrophysiology Experimental Setup for Brain Machine Interface

Students: Aaron Choong & Jesslyn Sutanti
Supervisors: Dr. Marcello Rosa, Dr. Kostas Chatzidimitrakis, Dr. Yan Wong, and Mr. Masoud Ghodrati

Department of Electrical and Computer Systems Engineering
In Collaboration with
Department of Physiology

Last Revised: 25 September 2017

	First Name	Last name	ID	Email
Supervisor	Yan T.	Wong	1116482	yan.wong@monash.edu
Student 1	Jiun H.	Choong	25101994	jhcho33@student.monash.edu
Student 2	Jesslyn S.	Sjofian	25400126	jsut20@student.monash.edu

Preface

This technical manual outlines safety procedures, hardware setups, required specifications of hardware and software, code functionality, limitations, and future potential improvements for the experimental real-time brain-machine interface (BMI) setup.

Furthermore, the function calls in this report assumes that two files have been downloaded from the Cambridge Electronics Design (CED) website. One file is the Use1401 library which contains the device driver to communicate between the Power1401 (data acquisition device) and the personal computer (PC). The other file is the matced64c library which is a Macro Editor Expound (MEX) file that compiles MATLAB commands into Use1401 commands to control the Power1401. Code testing has been done with MATLAB versions 2015 and 2016.

The hardware utilised in this setup are commercial products that are specifically designed for neural data analysis. MATLAB has been utilised for the entirety of the project to reduce data transfer overhead between applications as MATLAB has powerful analytical tools. The software can be sped up further with the use of .mex functions to create a more efficient analysis program in the future.

The experiment setup for this project has been built for a specific series of events for the experiment. The role of the authors for this project was to assist the physiology professor to set up a data acquisition system to analyse the neural data obtained from the experiment subject. Any user of this manual can utilise this manual to optimise the data acquisition setup.

Acknowledgements

The authors of this manual would like to thank the following people for the project:

Dr. Yan Wong - For guiding the authors by contributing technical support and ongoing supervision as the project supervisor.
Dr. Kostas Chatzidimitrakis - For providing an opportunity to assist in the hardware and software setup for his research.
Mr. Masoud Ghodrati - For assisting the authors in setting up the parts of the hardware and creating the GUI.
Cambridge Electronic Design (CED) - For providing additional information with regards to the devices.

Navigation tips

The following points below will assist in navigating through the technical manual:

1) The manual will use technical engineering and physiology terminologies, hence users can refer to the *Glossary* section for further understanding.
2) Any underlined, italicised, black text coloured phrases is a link to the relevant section of the manual. Users can click on the phrase and will be navigated to the corresponding section.
3) Any underlined, italicised, blue text is a website link that will require Internet for access.
4) There is a section containing common problems and solutions that might occur to the user which can be found in *Section 11.0*.
5) *Section 14.0* is the reference section which will have useful links to manuals, tutorials, training tools etc.
6) A list of abbreviations that are used throughout the manual can be found on the *page* below.
7) To start the experiment and analysis instantly, refer to *Section 6.0* and *Section 10.0*. These two sections will provide a basis on how to setup the experiment and use the analytical tools.
8) To utilise the sorted data for an experiment for self-written code, refer to *Section 8,4* as the data formats will be explained in that section.

List of Abbreviations

ADC	Analogue to Digital Converter
BMI	Brain Machine Interface
CED	Cambridge Electronics Design
CPU	Central Processing Unit
DLL	Dynamic-Link Library
DOS	Disk Operating System
GUI	Graphical User Interface
LED	Light Emitting Diode
kHz	kilo-Hertz
MCC	Micro Control Centre
MEX	Macro Editor Expound (file)
NaN	Not a Number
PC	Personal Computer
PSTH	Peri-Stimulus Time Histogram
RAM	Random Access Memory
UR5	Universal Robots (Type 5)
USB	Universal Serial Bus

1.0 Table of Contents

2.0 Safety Procedures

This next section highlights safety protocols that should be taken to minimise any potential damages to the equipment and the potential injuries for the people overseeing the experiment. Safety protocols include personal protective equipment to be worn by the user, handling equipment, and handling injuries.

2.1 Personal Protection

Personal protective equipment (PPE) for this experiment is important for the user to ensure no contraction of diseases are obtained. PPEs such as gloves and shoe covers can be found when entering the animal laboratory facility. Other PPEs including lab coat and safety glasses must be brought by the user when conducting experiments at the laboratory. Furthermore, consumption of any food and drinks are not permitted in the experiment rooms.

2.2 Equipment Handling

Special treatment is required while handling the experiment equipment as the equipment utilised for the experiments are costly in terms of actual price and delivery time. The most damaging equipment can occur from the UR5 Robotic Arm (Universal Robots, Denmark) during its initialisation stages. Care must be taken when releasing the UR5's joints when turning the device on from the PolyScope controller as the position of the UR5 is in proximity of the other lab equipment such as the hot glass, experiment frame, and the EyeLink camera (SR Research, Ottawa). Initialisation procedures will be explained thoroughly in Sections *6.0* and *16.0*.

The Amplifier System - A-M Systems 3600 (A-M Systems, Sequim) and the data acquisition device (Cambridge Electronic Design's Power1401 MKII) must be handled carefully. The A-M Systems 3600 feeds waveform inputs to the data acquisition device so it is suggested to keep the gain to the minimum value of 10. Inputs to the Power1401 MKII (CED, Cambridgeshire) should not exceed +/- 5V according to the Power1401 MKII *Owner's manual*. To ensure the electrodes are not damaged, it is suggested to not exceed +/- 1V. Furthermore, specific communication protocols must be followed when establishing and disconnecting a connection between the PC and Power1401 MKII. The steps will be outlined in the *Section 6.0* .

Lastly, the PC controlling the EyeLink (SR Research, Ottawa) runs on a disk operating system (DOS). Running on a DOS results in the PC having limited overhead that will is encountered on other operating systems such as Windows which maximises the capability of the PC such as RAM and processing power. However, due to this operating system, it is crucial to ensure that

the user disconnects any connection established from a second PC to the EyeLink PC. Furthermore, the user should switch off the PC controlling the EyeLink before switching off the EyeLink itself to prevent any corruption of data.

2.3 Handling Injuries

In case of injuries, first aid kits, washlets, and over sink eye wash are provided in the animal laboratory. There are a number of safety procedures posters attached adjacent to the first aid kits that handle case-by-case injuries. Any injuries must be reported to the relevant supervisor or to the office to ensure no diseases or infections has been contracted by the victim regardless of the size of the injury. Overall, the safety protocols will be provided during the safety briefing by the animal facility and it must be followed closely.

3.0 Introduction

This report is designed to allow readers to utilise the current experimental setup while understanding the reasons for the implementations. The technical manual will also highlight the current progress of the experimental setup and the possible improvements that can be implemented in the future. Furthermore, users of this manual will easily be able to modify the experimental setup to suit a different design while utilising the same hardware and software devices due to the in-depth analysis of the code.

The experimental setup involves acquiring the neural data off an experimental subject in real-time. Additional testing devices have been implemented such as a signal generator in order to test the data streaming code written by the authors. Limitations of devices and the software will be highlighted throughout the manual and within the supplied codes.

In terms of the software component, this report will encapsulate all function declarations created by the authors. Some functions include:

- the data streaming code to retrieve the data off the data acquisition device (*Data_Stream.m)*
- the data analysis code to perform basic neural data analysis such as spike detection and post-stimulus time histogram (PSTH) (*spike_detection.m* and *psth_analysis.m)*
- the graphical user interface (GUI) for the user for online data analysis *(Main_PSTH_Raster_GUI_2.m)*

3.1 Experimental Setup

The flow diagram in Figure 3A (high-level overview of experimental setup). depicts the end to end view of how neural data is extracted during the experiment and transferred to the computer for analysis. Subsequent sections will delve deeper into how each component is controlled during an experiment.

The neural data will initially be transferred through 16 micro-electrodes that are implanted within the subject. The electrodes have a preamplifier to amplify the signal before being filtered through the AM Systems 3600 Amplifier.

Currently, the number of electrodes that can be used is 16. This is due to the limited number of analogue-to-digital converter (ADC) inputs to the Power1401 MKII recorder as the analogue inputs are used to record the neural data. Furthermore, the waveform channels can be increased by implementing a second Power1401 MKII. However, this may result in waveform data being missed with the current PC processing power,

MATLAB will acquire the neural data through a USB cable by sending strings of commands to the Power1401. MATLAB will also be used to analyse the neural data through a decoder to produce an output signal state for the robotic arm.

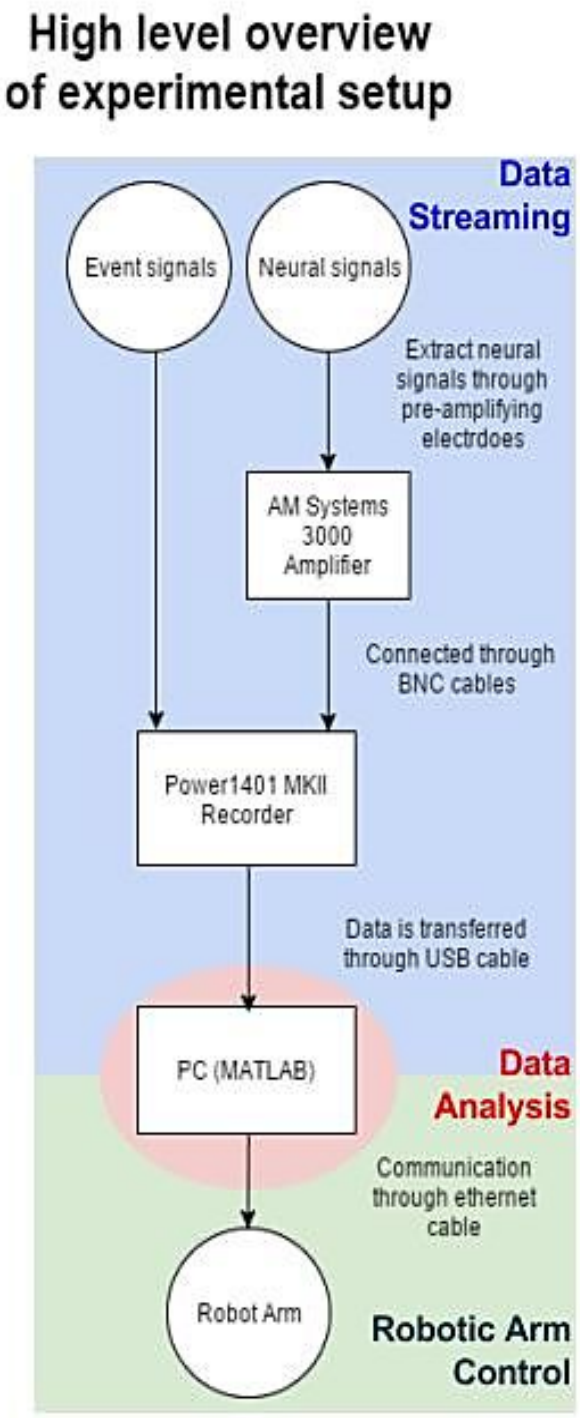


Figure 3A. High level overview of the hardware devices used from the initial data streaming to the robotic arm control.

3.2 Control Setup

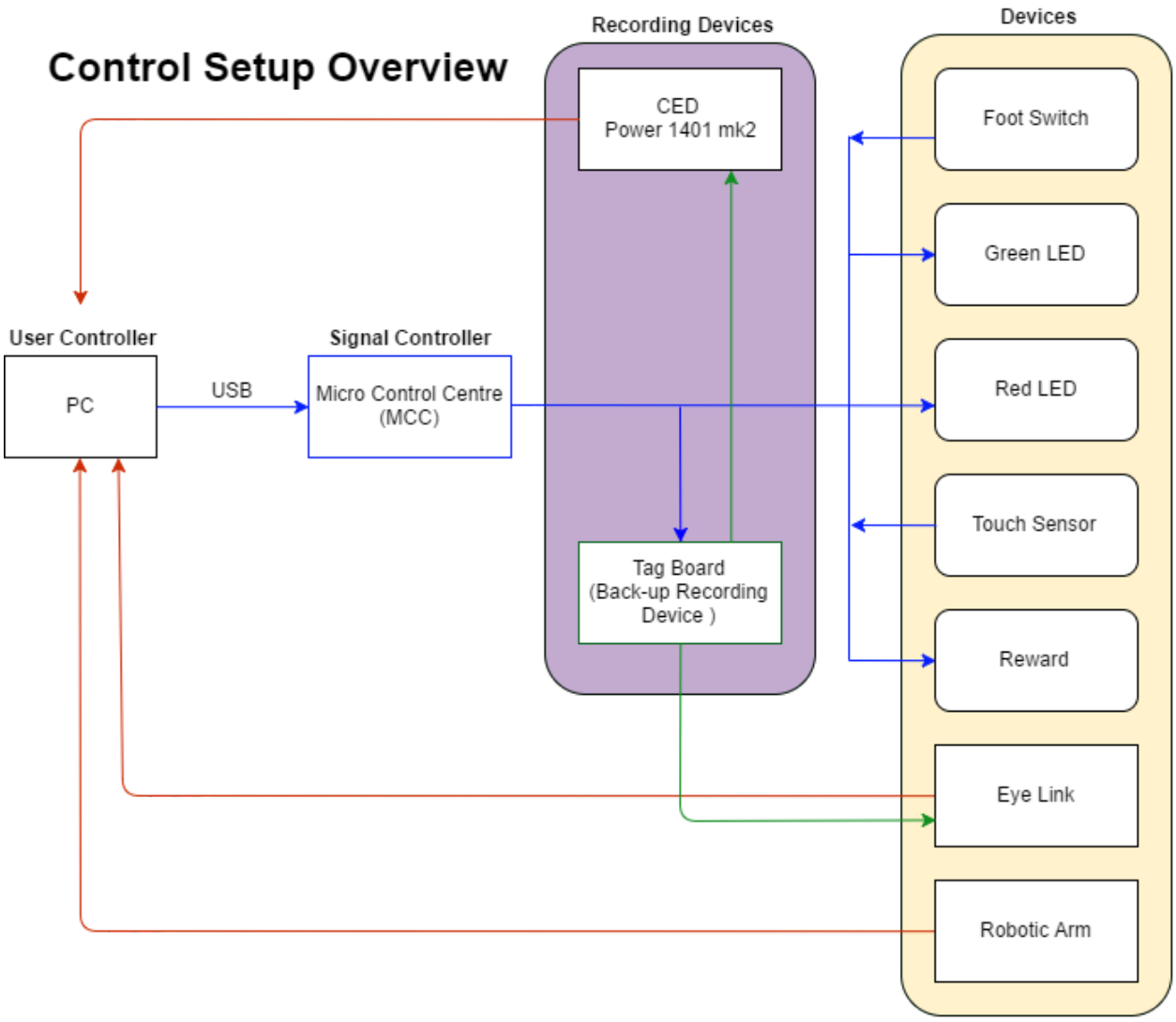


Figure 3B. The communication setup between the user and the devices that are used for the behavioural events in the experiment.

The above figure depicts the entire control setup for the experiment. The user utilises MATLAB to control signals from the PC feeding into the micro control centre (MCC) which is a microprocessor to send signals to the experimental devices. Two recording devices exist within the experimental setup: the Power1401 MKII recorder and the tag board. The Power1401 is used to stream the neural data to the PC whereas the tag board is used as a secondary recording device

for the control signals and to provide isolation of current spikes between MCC, eye link and the CED Power1401 MKII.

The controllable devices such as green LED, red LED, and reward are all unidirectional with all the information transferring from the MCC to the tag board. In contrast, footswitch and touch sensor will give the timing feedback to MCC and tag board. The information from the tag board will be sent as events signal to the Power1401 digital inputs while other information is sent to the eye link to ensure the subject of the experiment is fixated on the current stimulus.

The PC is also a control centre for the eye link and robotic arm. As the eye link runs on a DOS operating system, the information is transferred and stored on the main PC. This allows minimal overhead of other processes running in the DOS system as opposed to a normal windows setup. The user's PC communicates with the robotic arm by setting up several communication ports to transfer control signals and position data through an Ethernet cable.

3.3 Event Sequences

This section will highlight a brief overview of the sequence of events that will occur during a trial. As the research requires neural data that corresponds to the reaching and grasping motion of the experiment subject's hand, the experiment has been implemented in a specific method to mitigate irrelevant neural data. The controlled events will highlight the neural data that corresponds to expected behavioural attributes such as fixation on the task as opposed to the experimental subject saccading from the tasks which results in noisy data.

The initial stage requires the footswitch to be held down for a certain amount of time to ensure that the experiment subject is fixated on the tasks. The green LED flashing indicates to the experiment subject that the task has been completed successfully thus far. The red LED flashing is to ensure that the experiment subject has been fixated on the green LED to mitigate irrelevant neural data prior to the reach and grasping of the touch sensor. The touch sensor is used in order to acquire neural data corresponding to the reaching and grasping activity by the experiment subject.

The sequence of events is as follows:

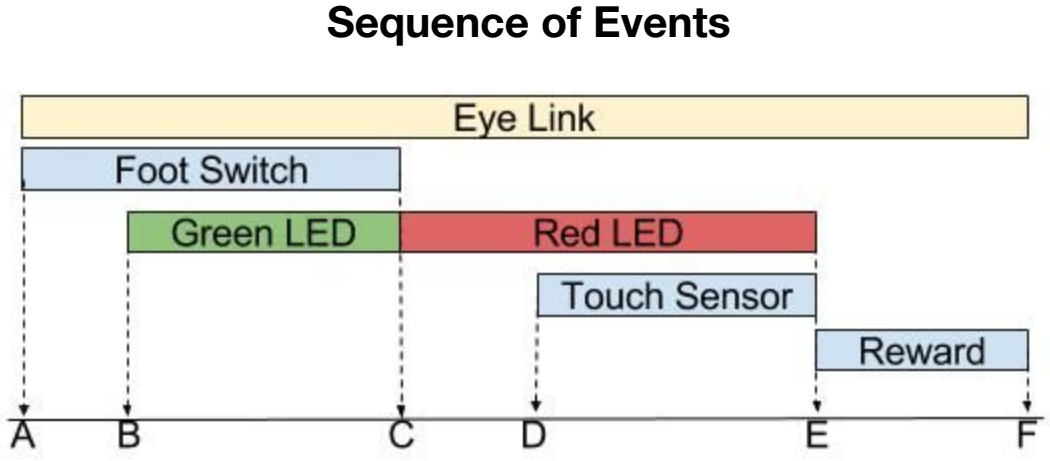


Figure 3C. The sequence of behavioural events that occur sequentially with the EyeLink ensuring the experimental subject is fixated on the task.

At the beginning of the experiment, MATLAB will wait for a footswitch press **(A)** by the subject before proceeding with the experiment. Once the footswitch is held for a certain time **(B)**, the green LED will turn on for a certain duration while the eye link is tracking the subject's eye movement. If the eye movement deviates from the green LED stimulus, the experiment will restart the green LED. However, if the eyes are fixated onto the green LED, **(C)** the red LED will turn on after the green LED. Once the red LED is activated, the subject will have a time limit of

1 second to release the footswitch and press down on the touch sensor **(D)** otherwise the experiment will reset back to the green LED. If the subject achieves the touching requirement within the time frame, the subject will have to fixate on the red LED while holding down on the touch sensor. The reward will be given to the subject (**E**) when the subject releases the touch sensor within a certain time after the red LED switches off. The trial will be considered successful if the reward is given (**F**). Then it will restart with the subject pressing the footswitch **(A)** and the robotic arm will be signalled to move to a different position.

3.4 Current Capability

The following lists highlight the current capability for experimental setup. There are suggested fixes in *Section 13.0* of the manual that could further optimise the code to obtain a better throughput of data.

3.4.1 Real Time Data Streaming

The waveform and event data loss have been tested by recording all the data during an actual experiment run by the authors. A single PC was used to run 3 MATLAB codes along with a video recording application instead of 2 MATLAB codes only (*Data_Stream.m and Main_PSTH_GUI_Raster_2.m*).

The current capability of the real-time data streaming is summarised in the table below:

Description	**Result**
Number of waveform channels	16
Number of events	8
Sampling rate (waveform)	31.25kHz per waveform channel
Sampling rate (events)	1kHz per digital channel
Number of data points per transfer	1000 points per waveform channel
Precision	2 bytes
Transfer delay	~30ms per 1,000 points per channel at the above settings
Waveform data loss probability	Less than 0.0025% loss rate
Event data loss probability	Less than 1% loss rate

Figure 3D. Table of specifications for the experimental setup with version 1 of the data acquisition and analysis code.

The waveform data was tested by sending two sinusoids with different frequencies to two channels. The event data was tested by recording all the events that occurred during successful trials. Test trials have been accumulated over several experiments, reaching 200 successful trials over 5 experiments of data. The waveform data loss probabilities were calculated by comparing the expected number of peaks of the sinusoids while ensuring the raw waveforms are connected. Using both the conditions indicated the amount of waveform data that was lost during the trials.

The event data loss has been calculated by analysing all the events that were recorded during a successful trial while ensuring the robot position number was correct. This allowed the measurement of the amount of missing event data during a successful trial. Furthermore, analysis is done only on the event data that has recorded which will yield the correct event timings.

The above delays and data loss probabilities can be further decreased to zero by implementing the suggested future fixes in *Section 13.0*. The loss rates are due to the limitations of the hardware and software setup that will be discussed in the later sections of the technical manual.

3.4.2 MATLAB Overall Control

By utilising the various MEX functions that enable MATLAB to control the data acquisition device, the experimental setup has been compressed to only using MATLAB for the entire control. This allows convenience for users to focus on one programming language while minimising overhead in data transfer between two applications (such as Spike2 transferring data to MATLAB for analysis). The entire list of benefits can be seen in the table below:

Description	**Benefit**
Convenience of coding	One programming language
Data transfer between applications	Reduced data transfer overhead and bulkiness in data storage
MATLAB for data analysis	Powerful inbuilt tools in MATLAB

Figure 3F. Table of extended benefits that have arised from the sole reliance on MATLAB for the setup.

Although it allows users to program in a single language, the user is still required to learn the syntax requirements to communicate with the Power1401 as MATLAB has to send the command strings in the required format. However, the basis of the experiment has been organised for easy modification to the code.

4.0 Hardware Notes

The following section encapsulates an introduction to each hardware device that has been used for the experimental and data acquisition setup. The functionalities that has been utilised by each hardware device will be highlighted with any software solutions implemented to bypass the hardware limitations.

4.1 Experiment Devices

The following subsections will cover the hardware devices utilised in this experimental setup. Online links to the manuals for each device will be accessible by the user if the user requires more information about the component. Some devices will have schematics of the circuitry instead of the manual as the hardware setup was created by a separate party.

4.1.1 Footswitch

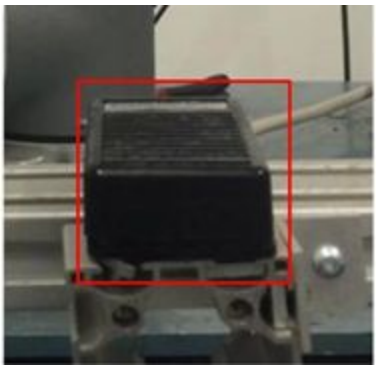


Figure 4A. Front view of the footswitch (Ojiden, Osaka) used for one of the behavioural events.

The footswitch (Ojiden, Osaka) is an imported product from Japan and the product specifications can be found *here*. The footswitch has been used to initiate the experimental trials. As the footswitch is a mechanical switch, the footswitch will switch between high and low digital states multiple times when the footswitch has been pressed and released. Hence, the software will debounce the footswitch, meaning that each footswitch press and release will only correspond to only one recorded number.

4.1.2 LEDs

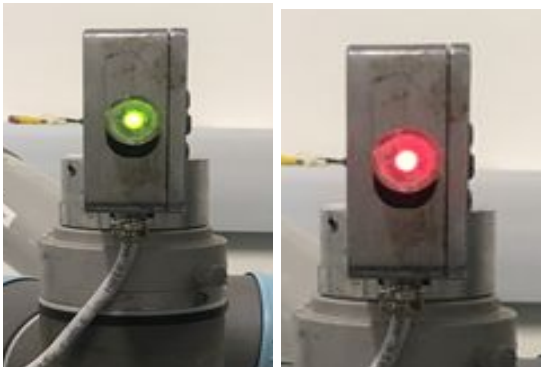


Figure 4B. Front view of the green and red LEDs respectively which is incorporated into the touchsensor. The entire device is held by the UR5 robotic arm.

The LEDs are encapsulated in the touch-sensor as one device. The touch-sensor has coloured LEDs that can be controlled through a micro-controller for the desired colour as shown above. The different LED colours will represent different sections of the trial.

4.1.3 Touchsensor

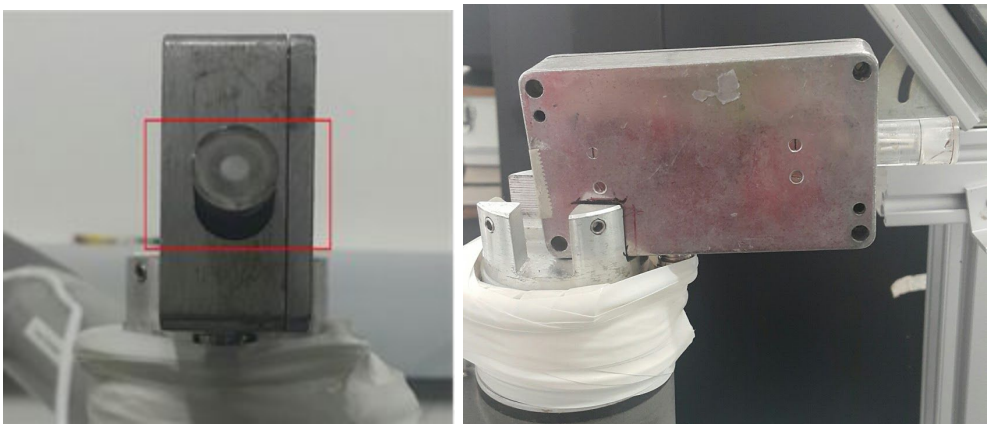


Figure 4C. Front and side view of the touchsensor.

These views shows the position of the green LED, red LED and the touch sensor in a box attached to the robotic arm. The touch-sensor is a switch that detects a change in value when the sensor is held down. Note that the touch sensor has an LED inside that is utilised as part of the experiment. The touch sensor and LEDs connection schematic can be found in the *Appendix*.

4.1.4 Reward System Controller

Figure 4D. Front view for the reward controller with all the available controls. The reward acquisition is controlled by a MCC through MATLAB.

The Rack Mount Controller by Crist Instrument Co., Inc. is used as reward controller for the experiment. The model of the reward controller is 5-RLD-D1 and the model specification can be found *here* (note that the interface will seem different). The reward controller will be activated at the end of a successful trial by MATLAB code which will feed a liquid reward to the macaque.

4.1.5 Reward System Dispenser

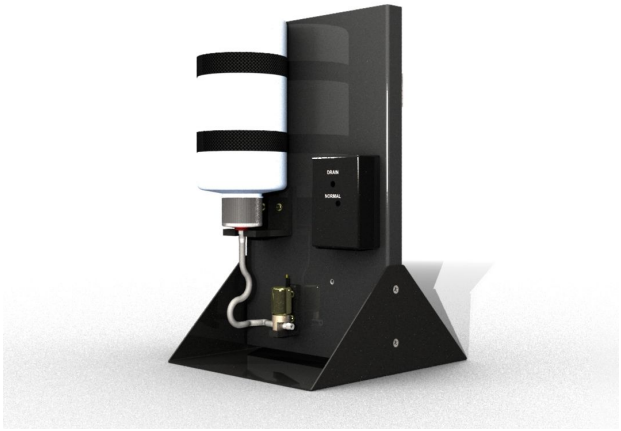


Figure 4E. The reward system dispenser that is used for the experiment.

The reward system dispenser is a product that comes from the same company as the reward system controller (Crist Instrument Co.). As both the dispenser and controller are manufactured by the same company, the dispenser and the controller are highly compatible. More information on the dispenser can be found *here*.

4.1.6 Tag Board

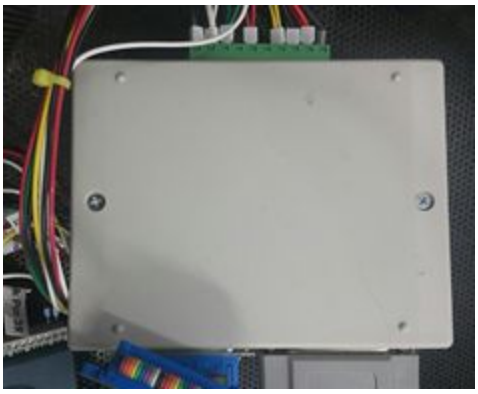


Figure 4F. A top view of the tagboard that encapsulates the wiring between different devices. It is also used as a secondary storage of digital information for the device trigger timings.

The tag board provides electrical isolation between the different components in the project setup and the micro-controller. It also stores all the digital information from the devices where it sends information to the Power1401 recorder as digital inputs for all the events.

4.1.7 Micro Control Centre (MCC)

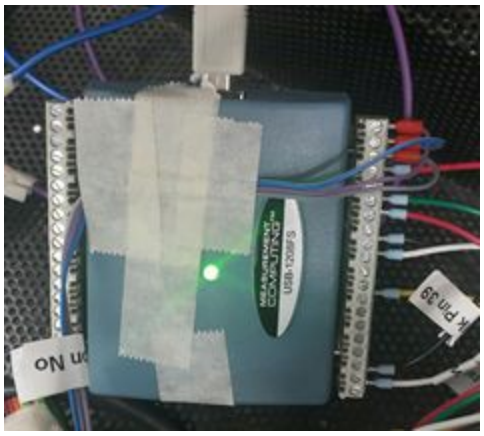


Figure 4G. A top view of the MCC that controls all the devices with the GPIO pins.

The micro-control centre (MCC) is a device (USB-1208FS) that controls the signals to the LEDs and rewards. The MCC is similar to other micro-controllers in functionality and in limitations. Any hardware changes to the MCC setup will require consultation with the technical manual to prevent any damages done to the micro-controller. The technical manual for USB-1208FS model can be found *here*. The outputs of the MCC feeds into the tag board to record the information which will control the experiment devices through the tagboard.

4.1.8 EyeLink 1000

Figure 4G. (Left) EyeLink monitor to ensure experimental subject is fixated on the tasks. (Middle) The EyeLink CPU that runs on a Disk Operating System (DOS). (Right) The EyeLink camera and mount that has been placed above the experiment subject. This will monitor the the subject's eye movements.

EyeLink is a device imported from Canada which will track the eye movement of the experiment subject (macaque).The EyeLink camera is positioned above the macaque so that it can observe and record the eye movements of the macaque during the trials to ensure the macaque is fixated on the tasks. The figure shows the CPU used for the EyeLink camera system which performs analysis using a DOS operating system to maximise processing speed for the EyeLink for real-time analysis. The figures display the user interface of EyeLink in the monitor where it enables the user to monitor the eye movement. The user guide for EyeLink can be found *here*. The authors of this manual did not modify the EyeLink program during the project duration except for the *gaze boundary* to test all the robot positions.

4.1.9 UR5 Robotic Arm

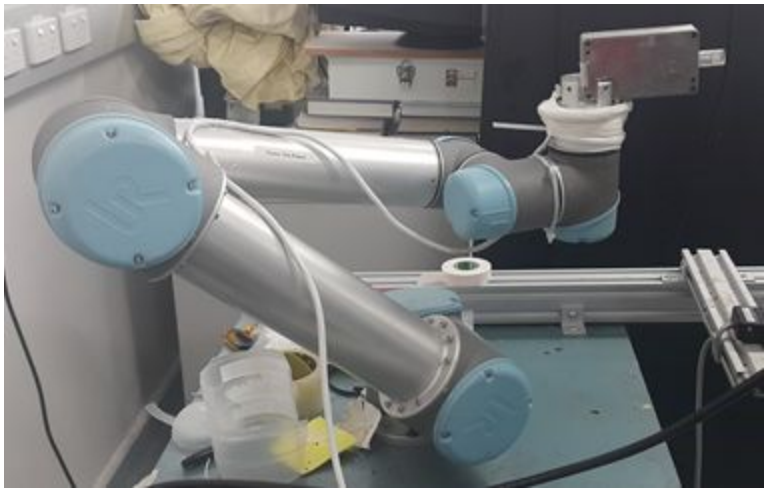


Figure 4H. Robotic arm side view with the touch sensor and LED box attached

A side view of the robotic arm shows the robotic arm at the initial starting position for the first experimental trial. The robotic arm will switch between 9 pre-defined locations for the macaque to reach where the positions will be aligned according to the macaque's eye level.

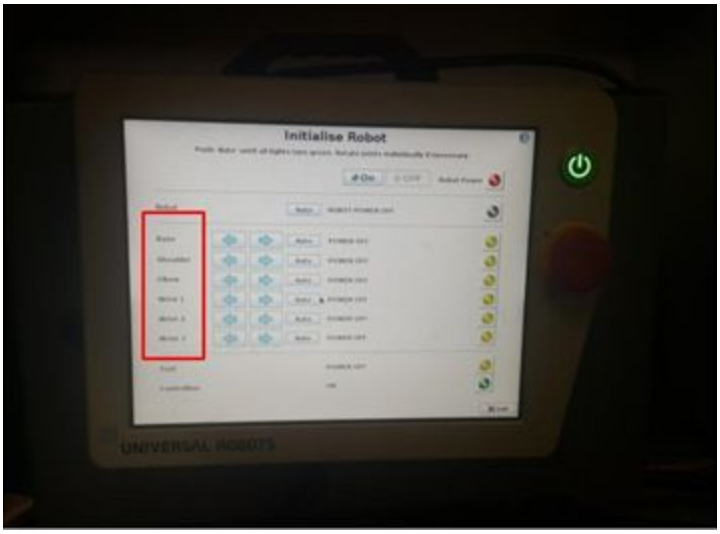


Figure 4I. The UR5 controller device at initialisation setup with all the manual joint controllers

The robotic arm which is used is UR5 by Universal Robots. This commercial product allows for 6 degrees of freedom for its movement and can handle a load of 5 kg. The controlling device which is shown in figure 8 can initialise the robotic arm as it turns on, control each robotic arm joint manually and stop the robotic arm movement after usage or in case of emergency. The robotic arm is also connected to the PC through an Ethernet connection.

4.2 Data Streaming System

The following subsection will discuss each of the data streaming hardware devices in detail.

4.2.1 Pre-Amplifying Micro-Electrodes

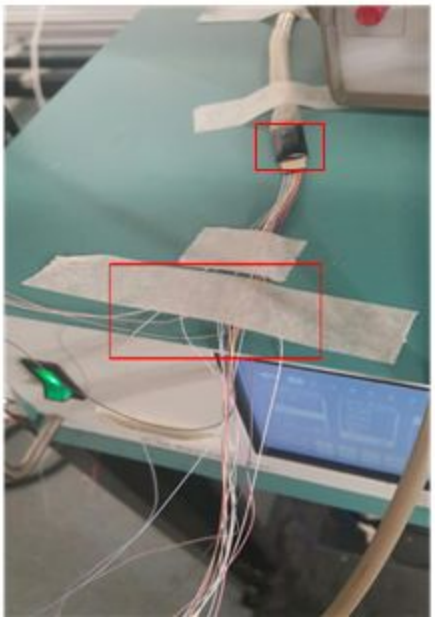


Figure 4J. A picture that includes the micro-electrodes and the pre-amplifier for the neural data which is fed into the A-M Systems 3600 Amplifier.

The red boxes in the above figure display the pre-amplifier and the electrodes used to record the neural data through invasive methods. The top red box is the pre-amplifier which is used to boost the signal to reduce the amount of noise recorded in the Power1401 MKII. The bigger red box shows all 16 electrodes (taped down for separate testing) where it will be used to access the macaque's brain. The authors of this manual have no information with regards to the specifications of the electrodes and pre-amplifiers.

4.2.2 Amplifier - A-M Systems 3600

Figure 4K. A snapshot of the controls for the A-M Systems 3600.

The amplification system used is the A-M Systems 3600 (A-M Systems, Sequim). It is a 16 channel electrode amplifier that has filtering capability along with safety protections while recording the neural data. In order to expand the number of waveform channels recorded, a second amplifier device must be purchased as the limit for this device is 16 channels. The A-M System 3600's gain can be adjusted by the user before initiating the experiment. It is suggested to keep the gain of the A-M Systems to a minimum of 10 to not exceed the rating of the Power1401 MKII (5V) or damage the micro-electrodes. Additionally, A-M System 3600 also has an inbuilt 1 kHz sinusoidal signal as a calibration signal. This signal can be fed to the recorder for testing the code. The signal amplitude can be adjusted for its amplitude in the calibration option. For more information regarding the setup of the device, refer to *Section 6.1*.

4.2.3 Data Acquisition Device - CED Power1401 MKII

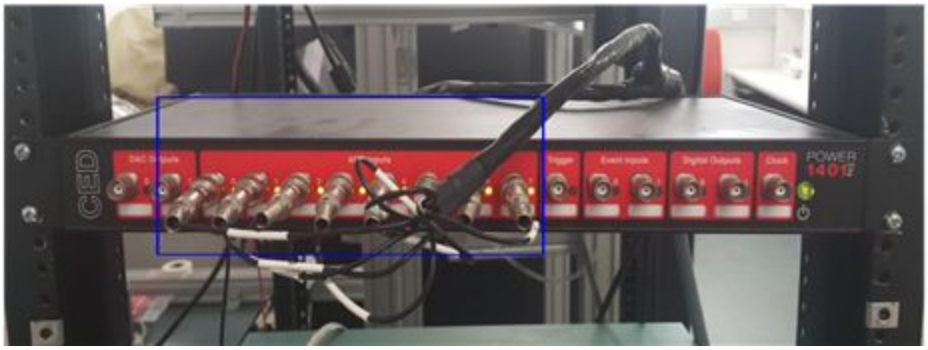


Figure 4L. (Top) A front view of the Power1401 data acquisition device with the first 8 waveform channels.
(Bottom) A back view of the 8 digital inputs used for the experimental setup. Waveform channels 9-16 are also fed into the back of the Power1401.

The data streaming of the neural signals requires compatibility of the software and the provided recording hardware (CED Power1401 MKII). CED is the distributor of the recording hardware and contributed software included a communication pathway between the hardware and MATLAB through the use of MEX functions. The MEX function converts MATLAB's strings into programming commands in the Power1401 by using a provided windows language (Use1401) to transfer the commands.

4.2.4 Device Driver

The most important device driver for this experiment is the Power1401 as the data acquisition device is imperative in order to for the experiment to retrieve any data. The device driver for the Power1401 can be found on the *CED website* by downloading the "1401 Installations" and "1401 Programming Support". Run the installer to download the device driver for the Power1401. There will be a pdf file that gives instructions on how to install the device driver properly in the Power1401 folder once the installer has been executed.

EyeLink will also require a device driver to communicate between the host PC and the EyeLink PC. The EyeLink device driver has been pre-installed on the host PC but a significant windows update might cause instability to the device driver, hence the host PC will require a reinstallation of the EyeLink device driver.

Furthermore, another important MATLAB function that has been sourced online to be compatible with the experimental setup code (*RobotExperiment_22Feb2017.m*) is the neuroscience MATLAB toolbox (psychtoolbox). The toolbox creates compatibility between the host PC's MATLAB code and the EyeLink program, hence making it invaluable to have the

psychtoolbox. If the toolbox gets removed by any chance or becomes incompatible with the new Windows update, the download link can be found in the *References* section or *here*.

4.3 Computer Specifications

For the BMI setup to run to the optimal level of the code, the computer specification for the data streaming code and the data sorting code must be considered for optimal real-time streaming and control of the robotic arm. The two PCs that are required should have the following specifications:

Rating	
Processor	I7-4790 CPU @ 3.60 GHz 3.6
RAM	16.0 GB

Figure 4M. The computer specifications used for simultaneously running the data acquisition, data analysis, and experiment control.

During the initial stages of the project, testing was done only on one PC which ran 3 MATLAB executables in parallel due to convenience. It is highly suggested to split the processing power into two PC to achieve optimal real-time streaming and analysis capability for the project. It is also imperative to set the MATLAB applications to high priority (steps can be found in the *Appendix* section), especially for the PC running the real-time streaming and data analysis MATLAB codes.

5.0 Software Notes

The following section encapsulates the current developed code for the BMI setup. Different sections of the developed code will be covered to discuss each code's reasoning, progress, and limitation. All the programming has been done through MATLAB including communication with the data acquisition hardware and the robotic arm. However, although MATLAB is the end-to-end software used in this experimental setup, users must learn the syntax that is required to control the Power1401. The manual can be found in the *References* section.

5.1 Software Programs

This experimental setup will utilise several programs in order to communicate between commercial devices and the host PC. Furthermore, the extra programs can be used for debugging and setup purposes such as Spike2 checking if the electrodes have been attached properly.

5.1.1 Spike2

Spike2 is the Power1401's manufacturer's software to analyse the data from the Power1401. It can also set some of the settings of the Power1401 (such as either reading EVENT 0 and EVENT 1 from the front digital inputs or the back digital inputs). The software is useful in debugging the experiment code to ensure all the devices are functioning correctly. Spike2's manual and video tutorials can be found on the *CED website.* There are also open source Spike2 codes that can be found on the *website* for other useful analysis such as using MATLAB to communicate with the Power1401 as used in this experimental setup.

5.1.2 MATLAB

The MATLAB that is used to run the experimental codes are the 2015a/b and 2016a/b versions. The most prominent coding glitch in the MATLAB code is accessing the .mat files in real time which might create incompatibility of other MATLAB versions to run the code. To optimise the processing speed of the code, the code can be written into MEX functions in the future to increase the capability of the code to operate in real time. This will be discussed in later sections of the manual.

5.1.3 INTERACT Program

To get a better understanding of how the Power1401 functions, users can use the INTERACT program that has been developed as an interactive GUI between the PC and the Power1401. The GUI will translate all the user inputs into strings of commands to the Power1401 and the user can

learn how to control the Power1401 via the scripting language. Hence, this program can be used in conjunction with Spike2 and the MATLAB code in order to grasp a better understanding of the Power1401's capability. The INTERACT program can be found via the *Power1401 installer*.

5.2 Communication with the Power1401 via MATLAB

The data streaming of the neural signals requires compatibility of the software and the provided recording hardware (CED Power1401 MKII). CED is the distributor of the recording hardware and contributed software included a communication pathway between the hardware and MATLAB through the use of MEX functions. The MEX function converts MATLAB's strings into programming commands in the Power1401 by using a provided windows language (Use1401) to transfer the commands. An overview of the communications can be seen in the figure below:

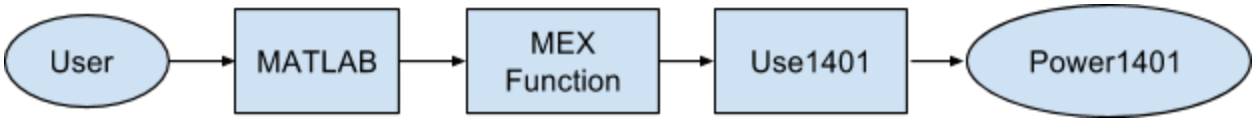


Figure 5A. The flow diagram of communication from the user to the Power1401

Spike2 can also be used to communicate with the Power1401 as it is CED's software to retrieve outputs from the Power1401. The Spike2 software can be used to debug the event and waveform channels to ensure that the experimental setup is running smoothly. This software can run with *RobotExperiment_22Feb2017.m* for debugging purposes.

5.3 Software Commands

The following section has an entire list of the important function calls for each program including the commercial software (Use1401 and Progmanw - both to control the Power1401) and the functions created by the authors of this manual. This is not an exhaustive list of all the existing functions but only covers the functions that are deemed to be important by the authors of this manual. For more detailed descriptions of each function, refer to the corresponding software manuals distributed by the original manufacturers of the programs. All the links to each manual can be found in the *Reference* section.

5.3.1 Power1401 MKII Commands

The programming language for the Power1401 MKII is a unique language developed by CED. The Power1401 MKII can be connected to a MAC, Linux, or Windows operating system. The language used to communicate with the Power1401 MKII is the *Use1401*. It is important to understand the Power1401 MKII's language and structure of the hardware and software to be able to use the device effectively.

Command	Description	Author's notes
ADCBST Refer to pages 33-35 of the Progmanw manual.	This function communicates to the Power1401 MKII to sample the **waveform channels** through an interrupt system that detects changes in the analogue input. It samples in burst mode where the points are sampled as close together as possible. Controllable variables include: the byte size, buffer pointer, buffer size, sampling rate, the specific waveform channels and number of cycles around the buffer.	This function is not used in the current experiment setup but can be useful for different setups requiring non-linear sampling.
ADCMEM Refer to pages 28-30 of the Progmanw manual.	This function samples the **waveform channels** in a periodic amount set by the user. Up to 32 waveforms can be sampled at once. Controllable variables include: the byte size, buffer pointer, buffer size, sampling rate, the specific waveform channels and number of cycles around the buffer.	This function is crucial for acquiring the data through the Power1401 MKII's waveform channel.
DIG Refer to page 41of the Progmanw	Simple digital I/O detection of **all of the digital ports**. It is intended for occasional changes for reading the inputs or changing the outputs. Controllable variables include: input/output specification, and the bit value.	This command is useful for testing events.

manual.		
AUDAT Refer to pages 75-76 of the Progmanw manual.	Stores changes in event levels (rising and falling edges) for **EVENT 0 and EVENT 1 at the front digital inputs**. Controllable variables include: the event channel, buffer pointer, buffer size, sampling rate, and number of cycles around the buffer. This function allows the buffer pointer to cycle back to the start of the event buffer which makes it powerful compared to the AUDATM.	This function is not used in the current experiment setup but future expansions of events can use the **front digital ports** of the Power1401.
AUDATM Refer to pages 77-79 of the Progmanw manual.	Stores changes in event levels (rising and falling edges) for **the digital inputs at the back of the Power1401 MKII**. This function differs from *AUDAT* as it can control the buffer for 8 digital inputs. This function can control the buffer pointer, buffer size, sampling rate, and number of cycles around the buffer. Controllable variables include: the event channel, buffer pointer, buffer size, sampling rate, and number of cycles of the buffer. This function requires manual reset of the event buffer.	This requires manual resetting of the event buffers when it has filled up with values. Furthermore, multiple of the same values will be recorded due to the hardware properties of the Power1401. The function has a circular buffer, thus, a manual reset to the event buffer is required.

Figure 5B. A list of useful internal Power1401 commands with a short description. Longer descriptions can be found by referring to the Progmanw manual.

5.3.2 Use1401

The Use1401 is the language provided by CED for users to communicate to the Power1401's family of devices with *Lang1401* as the software. However, although this experimental setup does not explicitly use the Use1401 language, the MatCED functions uses the commands of Use1401 in order to sends commands to the Power1401 through MATLAB. The structure of the communication languages used is outlined in *Section 5.2* and the similar Use1401 commands used by MATLAB will be highlighted in the *Section 5.3.3*.

5.3.3 MatCED

MatCED is the open source code that is published on the *CED website* for Power1401 users to use MATLAB to communicate with the Power1401 data acquisition devices. MatCED is a *MEX* function that uses C code to call Use1401 commands to send strings to the Power1401 and it is used in a similar method to a MATLAB function. The MatCED code cannot be altered without

changing the dynamic linking library (DLL) so it is not recommended by the authors of this manual to alter the contents of the MatCED MEX function. The current setup utilises the 64 bit version of the MatCED code (named *matced64c*). An important note for using the MatCED code is to compare the function calls with the Use1401 functions as they directly link to each other as seen in the table below. Function's comments by the original authors of the MatCED code is kept to a minimum so utilisation of the Use1401 manual to understand how to use the MatCED functions is essential.

Function in MatCED	**Equivalent function in Use1401**	**Description**	**Author's notes**
cedOpenX	U14Open1401	Establishes the connection between MATLAB and the Power1401.	Standard initalisation between host PC and Power1401.
cedCloseX	U14Close1401	Closes the connection between MATLAB and the Power1401.	Standard closing of connection.
cedLdX	U14Ld	Loads Power1401 commands into the device specified by the user.	Used at the start of MATLAB.
cedToHost	U14ToHost	Transfers data from the Power1401 to the PC. Controllable variables include: address of starting pointer and size of data transfer.	Regular transfer of waveform and event data.
cedTo1401	U14To1401	Transfers data from the PC to the Power1401. Controllable variables include: address of starting pointer, size of data transfer and data to be transferred.	Empties the buffer in the Power1401 by sending zeros to the event buffer as the event buffer is not circular.
cedSendString	U14SendString	Sends strings of Power1401 commands to the Power1401.	Used in conjunction with cedGetString.
cedGetString	U14GetString	Retrieves single strings of Power1401 commands from the Power1401.	Used regularly to retrieve pointers.

cedLongsFrom 1401	U14LongsFrom 1401	Retrieves long strings of Power1401 commands from the Power1401, typically with 2 to 3 return variables.	This command is not used in the current code. However, this function can be used for specific cedSendString commands such as AUDATM to get the pointer value and the buffer condition.

Figure 5C. A list of Windows OS (CED) and MATLAB reliant (MatCED) commands to send commands to the Power1401.

5.3.4 MATLAB Functions

Some of the following MATLAB functions are written by the authors of this manual. The MATLAB code used to run the experimental devices is *RobotExperiment_22Feb2017.m* and it communicates with all the devices including EyeLink and the UR5 robotic arm. *Data_Stream.m* and *Data_Sorting.m* are the main functions written by the authors of this manual to acquire the waveform data in real time and sort out the data according to the data stream settings. The GUI has been written up by an assisting Ph.D. student for the physiology professor to analyse the neural data in real time.

Function	**Description**	**Author's notes**
RobotExperiment_22Feb2017.m	Controls the whole experiment including communication of data with the EyeLink to ensure the subject is focussed on the experiment, communication with the equipment to trigger the events, and calculating the successful and unsuccessful trials during the experiment.	The robot position settings; such as the number position, can be altered through one of the code's function. This is in case if one of the position line gives error). However it is not recommended to change the settings in the function if it is not necessary.
Data_Stream.m	Controls the data streaming from the Power1401. The exhaustive list of the controllable variables can be seen in *Section 5.1.1*.	The code has not been optimised to have zero errors.

		Additionally, the number of trial does not match with the *RobotExperiment_22 Feb2017.m* as the unsuccessful trial condition is not the same.
Data_Sorting.m	Sorts out the data obtained by the Power1401 and outputs a structure with all the relevant information for analysis.	The speed has not been optimised for the data sorting.
Main_PSTH_Raster_GUI_2.m	Works in conjunction with *Data_Sorting.m* to provide users with a graphical interface for analysing the output	The GUI is still in its beta version as MATLAB's GUIs does not operate at the expected level.

Figure 5D. List of the functions that have been created for the experimental setup.

6.0 Getting Started

This section will cover how to use the experimental setup without the required technical knowledge of the setup. It will cover how to start the experiment, using the GUI and terminating the experiment. Any problems that are not highlighted in this section can be found in *Section 11.0* of the manual.

6.1 Starting the Experiment

These are the steps that should be taken in order to start the experiment. Images of the devices are shown in *Section 4.1.* More information with pictures on setting up the devices can be found in the *Appendix* section of the manual.

1) Turn on the the following devices: A-M Systems 3600, Power1401 and Reward Systems.
    a) For the A-M Systems 3600:
        - first calibrate the device by following the point in the screen.
        - press the number "1"at the top of the screen or "(-) Input" option.
        - setup each channel for the low pass filter to 3kHz, "NOTCH" to on, and set it to record "REC".
        - press "COPY" to copy the settings to all the other 15 channels.
        - There is also the calibration signal that can be found in the "A" tab on the upper left of the screen or "Calibration signal" option for any testing.
    b) Ensure that the USB cable is connected from the Power1401 to the PC. Turn on the Power1401 and wait until the power button is green. If the Power1401's power button is red and the first six LEDs keep flashing, close the MATLAB on the connected PC and restart the Power1401.
    c) Turn on the Reward Systems by switching on the power button.
2) Turn on the EyeLink camera power point before turning on the PC for the EyeLink.
3) Turn on the robotic arm by powering up the PolyScope controller.
4) Calibrate the UR5. This step can be referred to *Section 16.1* for pictorial demonstrations on the calibration of the robotic arm. The required steps are:
        - Calibrate the robotic arm's position by initializing each joint manually. Press "AUTO" at individual joints such as *Base, Shoulder, Elbow, Wrist 1, Wrist 2, Wrist 3*.
        - If the robotic arm is approaching collision with the other lab equipment, let go of the "AUTO" button and press it again to calibrate the robotic arm towards the opposite direction.
        - Calibrate all the joints and a green circle will appear for each joint that has been calibrated properly.

- Once all the joints have a green circle, place the PolyScope controller back to the holder.

5) Open up 3 MATLABs and open the following .m files:
   a) ***RobotExperiment_22Feb2017.m*** which can be found in: *C:\Data\UR5\KostasExperiment\RobotExp_left_repeatTrial_seems2Work\gazeOnline not used, storing gazeCheckOutput\RobotExperiment_22Feb2017.m*
   b) ***Data_Stream.m*** which can be found in: *Documents/Aaron_and_Jesslyn*
   c) ***Main_PSTH_Raster_GUI_2.m*** which can be found in: *Documents/Aaron_and_Jesslyn*
6) Set all MATLABs to high priority by:
   a) CTRL + ALT + DELETE
   b) Task Manager
   c) Find the 3 MATLAB applications under "Processes"
   d) Right click on each MATLAB and set "Priority" to "High"
7) Initiate the experiment by running *RobotExperiment_22Feb2017.m* that is connected to the EyeLink and the MCC. Wait until the UR5 robotic arm reaches the first position which can be seen directly or by viewing MATLAB's command window.
8) Start *Data_Stream.m* to initialise the buffer setups with the Power1401. When prompted by the MATLAB command window, press a key to start the streaming.
9) Wait until all the robot positions have been trialled (at least 9 correct trials). Afterwards, start the real time analysis by running *Main_PSTH_Raster_GUI_2.m* on the PC running *Data_Stream.m*. Select the file that is required for the online analysis.

In the image below, the user can use the settings provided on the right of the *Main_PSTH_Raster_GUI.2_m* GUI to control the following:

Variable	**Description**
Bin width	This controls the width of the PSTH bins when sweeping from minimum to maximum time relative to an event.
Time before event (ms)	This controls the minimum epoch time for the PSTH.
Time after event (ms)	This controls the maximum epoch time for the PSTH.
Channel	This controls the waveform channel that will be analysed.
Trials to analyse	Ability to choose to analyse successful, unsuccessful, or all trials.
Event marker	The event marker corresponds to the PSTH alignment to the event specified. The corresponding events can be seen in the blue box of

	the GUI.
Threshold	The threshold corresponds to a static spike detection voltage threshold.

Figure 6A. List of controls for the GUI during real-time analysis.

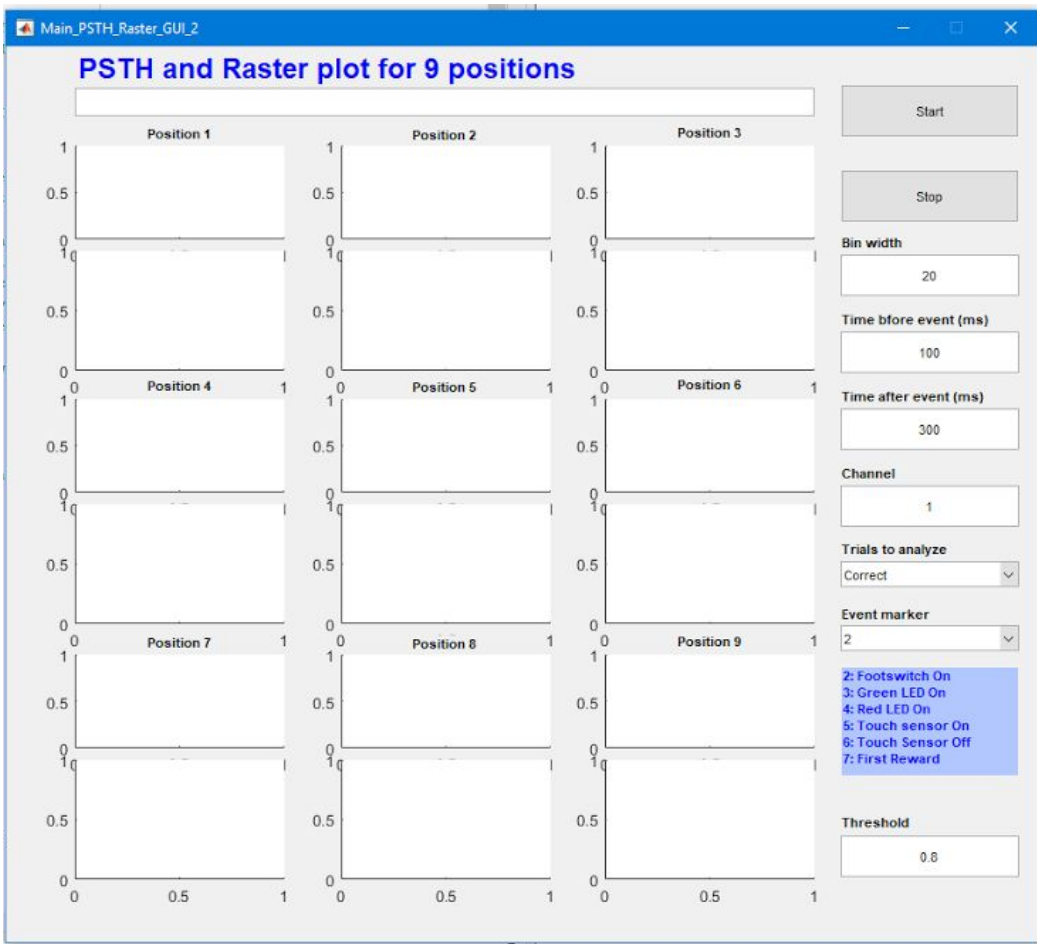


Figure 6B. A snapshot of the GUI before entering any data to analyse.

If any problems occurs during the start-up of the experiment, follow the above steps again. If the problems persists, refer to *Section 11.1* for common problems that users can encounter.

6.2 Settings of the GUI

This section will cover briefly about the settings that can be changed for the GUI. For more extensive descriptions of all the variables and controllability of the GUI, refer to *Section 10* of the manual.

If the experiment has been started without any errors from *Section 6.1*, PSTH and raster plots will be displayed on the GUI as seen below:

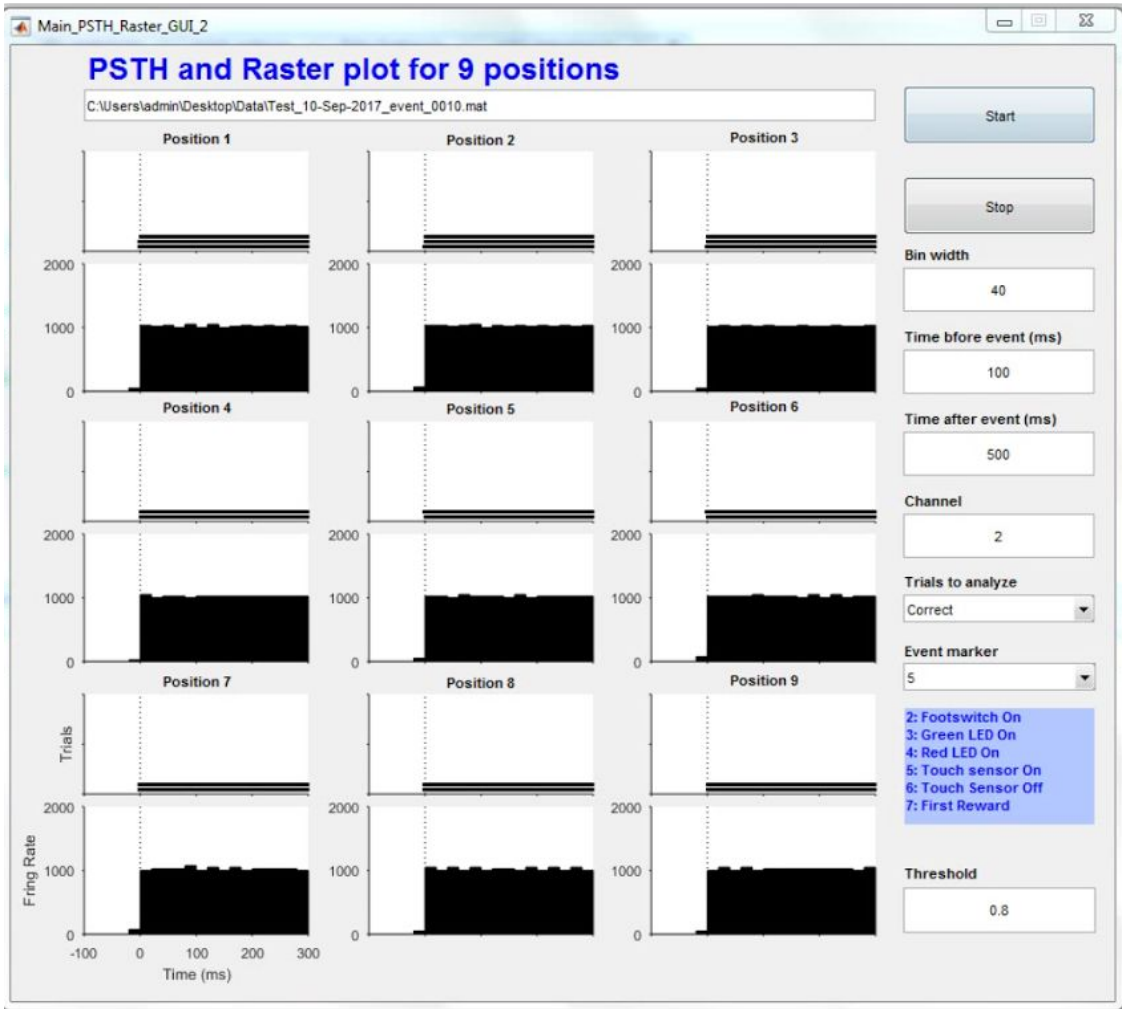


Figure 6C. A snapshot of the GUI displaying one of the waveform channels streaming in a 1kHz sinusoidal signal for each robot position. The PSTH and raster plots have been displayed on the GUI.

Another plot will be also displayed in a separate window as seen below:

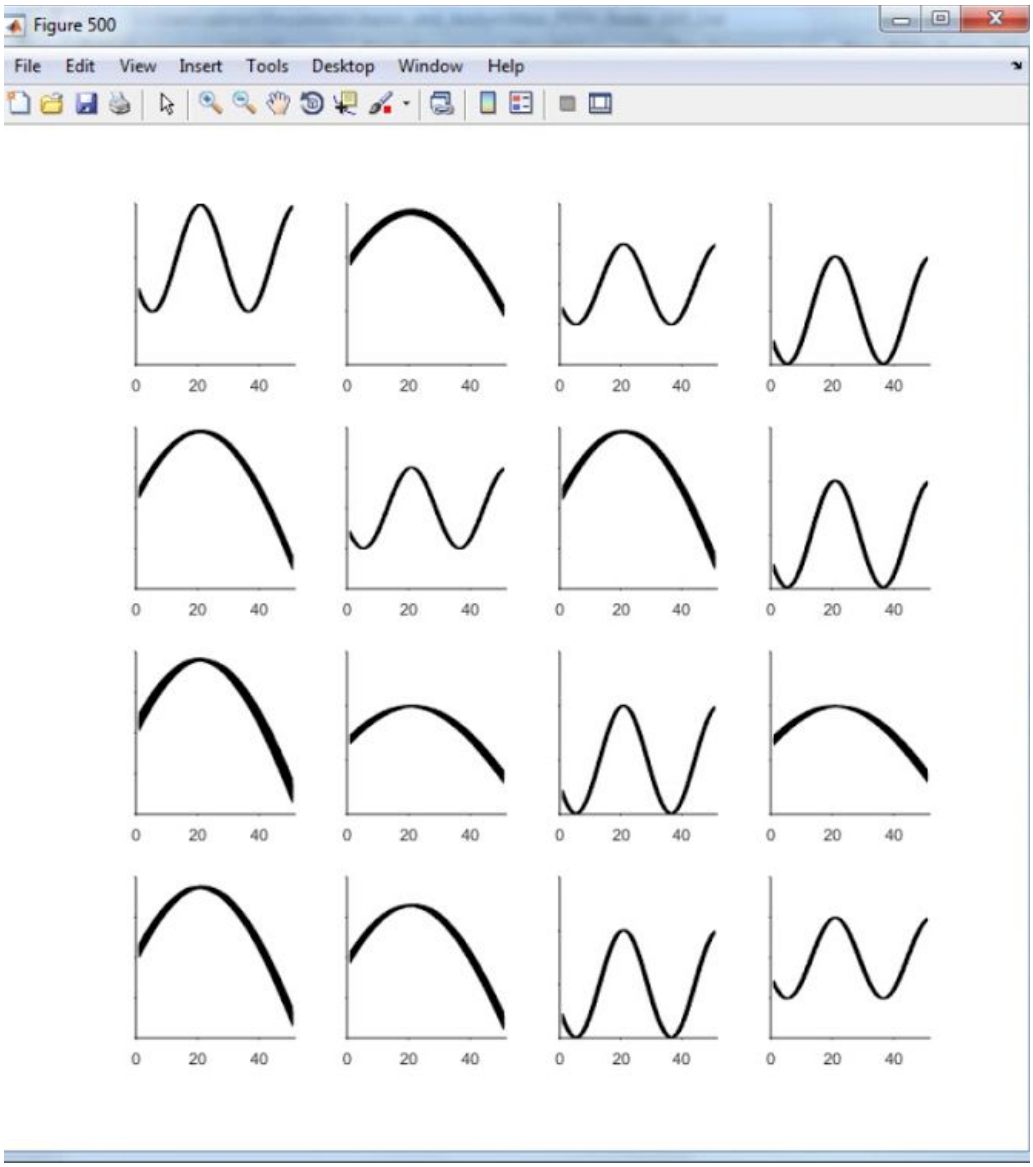


Figure 6D. A plot of the previous 5 trials for each spike waveform in each channel. Half of the channels are tested with 300Hz sine waves while the other half are tested with 1kHz sine waves.

This figure displays the spike waveforms for each channel. Note that the x-axis displays the vector index of the voltage value instead of the time value while the y-axis have not been standardised to zero voltage. The spike waveform shows every spike that was detected for the past 5 trials and are plotted into the same figure. This can be used to re-adjust the electrodes if similar neural clusters were being recorded at that specific location. In this example, half of the

channels are connected to a 300Hz sinusoid while the other half are connected to a 1kHz sinusoid. Also, the peak of the spikes are plotted at index 21 with 20 values before the spike and 30 values after the spike in the example above.

For more information on the GUI, refer to *Section 10* of the manual.

6.3 Terminating the Experiment

These are the steps that should be taken in order to terminate the experiment.

1) Terminate the MATLAB codes by pressing CTRL + C in the MATLAB command window for: *RobotExperiment_22Feb2017.m* and *Data_Stream.m*.
2) Terminate the GUI by pressing "STOP" in the GUI. Exit the GUI by closing the window.
3) Under the MATLAB window that was running *Data_Stream.m*, type this command into the command window:

$$res = matced64c('cedCloseX')$$

   Ensure that the value returned is zero. If it is anything other than zero, check the *Use1401 manual* for the error code in the appendix.
4) Close all the MATLAB windows
5) Switch off the power for:
   a) the UR5 robotic arm (by turning off the power at the PolyScope controller),
   b) Power1401 (by the switch button)
   c) Reward System (by the switch button)
   d) A-M Systems 3600 (by the switch button)
6) Turn off the EyeLink at the computer by clicking the "ABORT TRIAL" button. Press CTRL + ALT + Q or click the "TERMINATE PROGRAM" button to terminate the EyeLink camera connection to the PC.

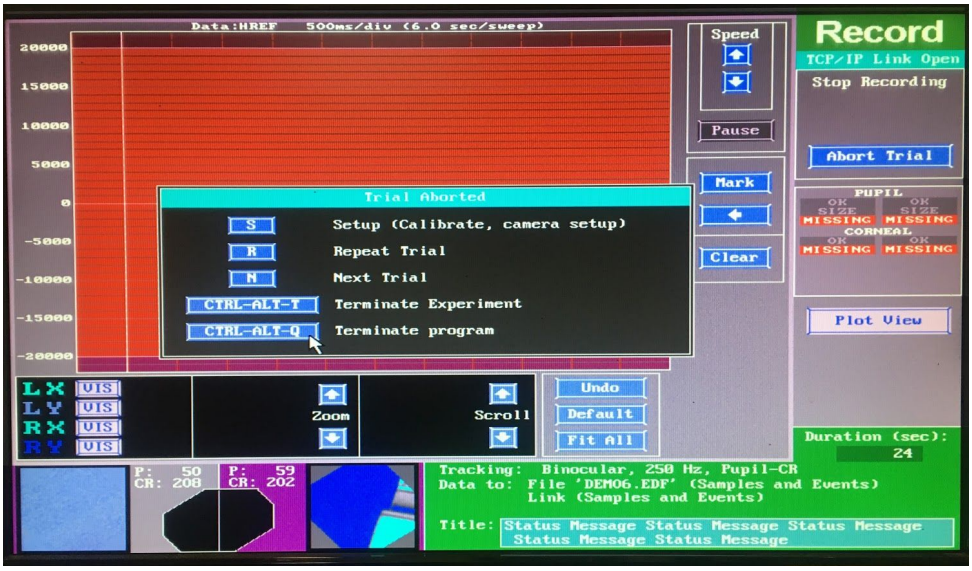


Figure 6E. Termination of the EyeLink program.

Switch off the EyeLink PC by pressing the power button located at the EyeLink CPU.

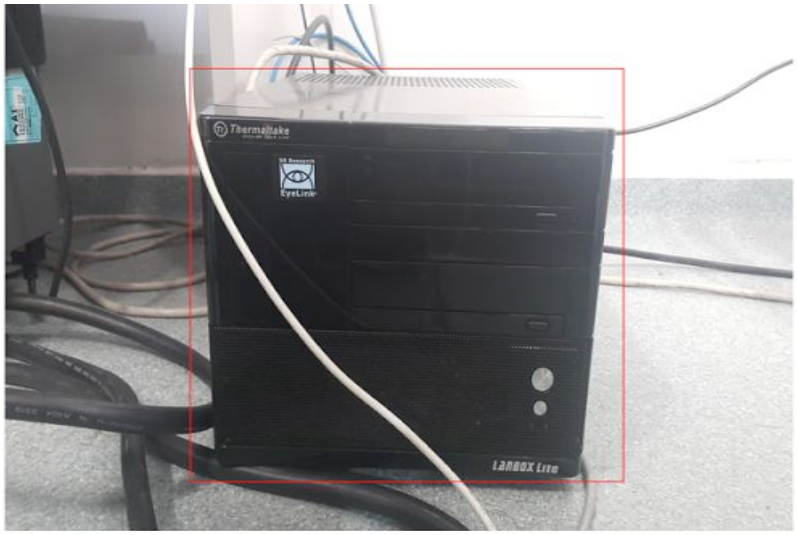


Figure 6F. Turning off the EyeLink through the CPU.

Switch off the EyeLink camera at the power source (located behind the UR5 robotic arm).

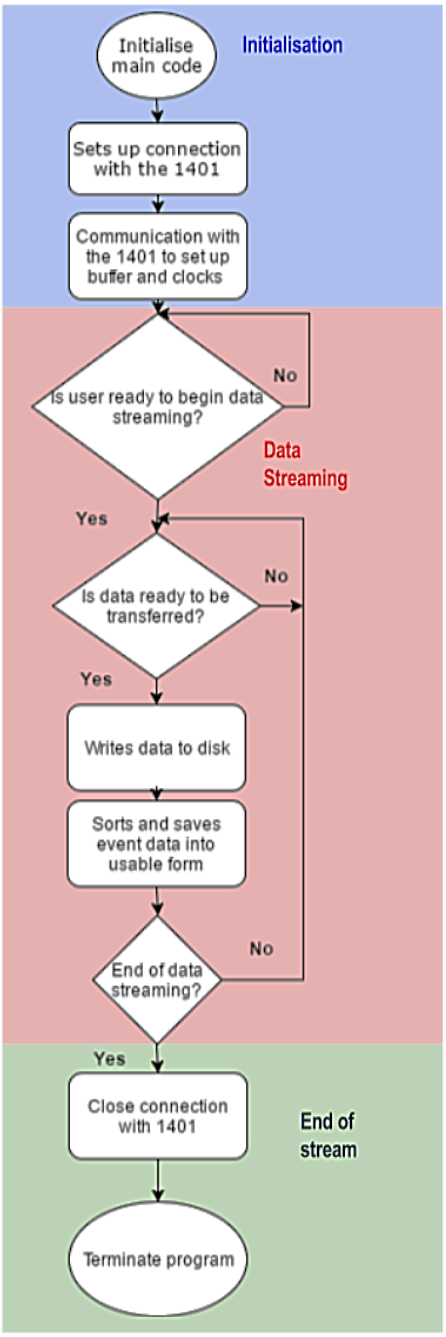
7.0 Data Streaming

The flow diagram in figure on the right represents how the data is transferred from the 1401. The code has a cyclomatic complexity of four meaning that the code is based off a few feedback conditions of waiting for the experiment to be terminated when the data has been retrieved sufficiently.

The most critical part of the data streaming code is to balance the buffer size within the 1401 and the data streaming capability. Due to the constraint of the experiment being a real-time systems setup, delays could not exceed 50ms for the data streaming and hence the amount of data transferred must be weighed against speed. This results in a lack of controllability of the buffer size and small buffer sizes will be inadequate as the neural data will be sampled too quickly and will overflow the buffer, hence missing neural data. Overall, with a sample speed of 31.25kHz per waveform channel (16 channels), each waveform data can have 1000-1200 data points (2 bytes each) which will achieve a data streaming speed of approximately 30ms.

Furthermore, the waveform data that is written to disk is a set of interleaved waveform data points. If the string command sent to the 1401 specifies that channels 0, 3, 1, 2 is to be recorded, the data retrieved from the buffer will be 0, 3, 1, 2 interleaved between each channel's data in that respective order. However, the data written to disk is in a simple int16 .bin format without any sorting to minimise any delays, which results in unsorted data in the .bin file which will be post-processed later to usable form. The event data is sorted according to the corresponding devices and is saved in a .mat format as a cell to allow for simple sorting. Further discussion can be found in *Section 3.4*.

Diagrammatic view of data streaming code



7.1 Data Streaming Functions

Real time data streaming with the recording hardware (Power1401) requires communication protocols between MATLAB and the Power1401. MATLAB was utilised for the data streaming as opposed to using CED's Spike2 software. Originally, tests were conducted to measure the real time capability of writing Spike2's data to disk (to allow for multi-processing for data analysis) and it was discovered that Spike2 was unable to record the data efficiently. This was tested with open source code found on the CED website, GitHub and other contributors from various websites. MATLAB was utilised instead as it is capable of performing real-time applications for analysis and data streaming.

The data streaming code written in MATLAB is divided into various functions: from initial connection with the Power1401, to initialisation of variables (for versatility) and the main data streaming code. A list of the functions can be found below:

Function definition	**Description**
Data_Stream	Contains all the function calls required to stream the data off the Power1401 MKII recorder. The transfer of the data is embedded in the main code to avoid function overhead between function calls
initialise_1401	Initialises connections and function commands with the 1401
file_setup	Sets up all the file names that will be used during each experiment
data_buffer_variables	Initialises all the waveform buffer variables
event_buffer_variables	Initialises all the event buffer variables
buffer_setup	Sets up the buffer within the 1401 with the defined variables
events_buffer_setup	Sets up the buffer within the 1401 with the defined variables
close_1401	Closes connection with the 1401

Figure 7B. List of the declared functions within Data_Stream.m

Full descriptions of each function can be found in the code itself.

7.2 Controllable Variables for Data Streaming

The main controllable variables for data streaming are listed below:

Variable	Declared In	Description
pre, cnt	*data_buffer_variables*	*pre* and *cnt* are divisors that control the sampling speed of the waveform channels. The fastest clock usable clock is 'H' which runs at 4MHz and is used to sample the waveform data on the rising edges of the clock. *pre* and *cnt* have a minimum value of 2 and hence the fastest sampling rate will be 4*10^6/(2*2) = 1MHz. This will also need to be divided into the number of channels (1MHz/16 channels = 62.5KHz per channel).
rpt	*data_buffer_variables*	This number specifies the number of repeats that the waveform data will be streamed in the circular buffer. A value of 0 indicates it will circle around the buffer with the maximum number of times (2^32 times).
channel_array	*data_buffer_variables*	The channel array indicates which waveform channels will be utilised during the experiment. Channels can range from 0 -> 15 and it is written as a normal MATLAB array. Conversion of the MATLAB array to a Power1401 string has been done in the Data_Stream.
single_data_length	*data_buffer_variables*	Controls the number of data points that will be set up in a section in the data buffer. This must be balanced with the sampling speed of each channel otherwise significant amount of data will be missed.
no_of_dig_inputs	*event_buffer_variables*	This allows expansion of the number of digital inputs (up to 8) for the Power1401 MKII. It cannot go beyond 10 due to the limitation of the Power1401 MKII. Code will be need to be written to allow 9 to 10 digital inputs.
time_length	*event_buffer_variables*	This indicates the sampling speed of the event channel which can be set from 65536 microseconds to 10 nanoseconds.
buffer_size	*event_buffer_variables*	Changes the size of the buffer holding all the events.

Figure 7C. Table of the controllable variables with descriptions in Data_Stream.m

'

Full descriptions of each variable can be found in the corresponding functions.

7.3 Waveform Data Streaming

The streaming of the waveform data is relatively simple compared to the event data streaming in the next section. Waveform data will be extracted from the Power1401 buffer as an int16 value (the number of points will be specified by the user) and will be stored in a MATLAB vector. The MATLAB vector will be saved into a binary file immediately as an int16 value to minimise any processing delays.

At the start of the trial on the footswitch press, all the waveform buffers will restart back to the start. This is to align the waveform data with the start of the trial and to minimise any data loss. The indication of a new trial is denoted by the reset of the time values in the timestamps file. This will be important for the accessing the data during online and offline experiments.

7.4 Event Data Streaming

The streaming of the event data is complicated due to the structure of the Power1401 command *AUDATM*. The sampling of the events does not give individual values for each event as there are debouncing errors when the events are being sampled (more information can be found in *Section 12.2.1 limitation number 3)*. Due to the multiple samples of the same event trigger, the code will sort out the code by finding the smallest value for that corresponding event in the buffer and will record that value into the .mat file.

Writing the event value to disk will only be triggered on a rising edge of the event (such as LEDs turning on, touch sensor being pressed) with exception for touch sensor as the released time is also recorded. Furthermore, the footswitch event is used to reset the event buffer which will result in all the recorded event times being relative to the footswitch event for that given trial. The reason for aligning all the timings to the footswitch is because the *AUDATM* command does not allow repeatable cycles for the digital events from the back of the Power1401 inputs, hence the recording system required a method to refresh the buffers.

Another limitation to the *AUDATM* command is the clock counter. The clock has a counter with maximum value of 65536 per sample. Thus, if the sample is 1 ms, the clock can only reach up to 65 seconds and reset to 0. This will cause problems if the trial exceeds 1 minute as the time measurement will be inaccurate. Therefore, the event buffer will reset if the trial exceeds the timeout value (30 seconds). Further limitations of the data streaming can be found in *Section 12.2.1*.

7.5 Data Formats

This section of the manual will cover the recorded formats of the data during the real-time streaming. This is important for any outside user to use the raw data that has been acquired through the Power1401 and MATLAB. The bare minimum of information has been recorded during the data streaming to ensure that minimise delay in the data streaming component of the experiment.

7.5.1 Data Formatting

When initializing the data streaming, 3 files will be created in the same directory; namely the waveform data, time-stamps, and event times. The reason for the 3 separate files is to ensure that the data sorting component is simplified while also minimising the information that is recorded for the different type of required information. This can be seen in the table below:

File/Format	Description of Format
Data (*.dat*)	The .dat file format was chosen because: - smaller data type as it is saved in flat binary (int16) which reduces data storage - the most compressed method of recording values through MATLAB (as opposed to .mat files as discussed later) as it is saved as a vector form This flat binary recording format is also a common way to record raw waveform data in the brain-machine interface field when acquiring data. The waveform data that is recorded will have interleaving data points between each channel when running the ADCMEM command in the Power1401 (*Section2.4.1*). The way the data is recorded is dependent on the channel buffer setup in *Data_Stream.m* with the variable **"channel_array"**. If the channel_array was [0 5 9 0 0], the data interleaved in the file will be in the same order (sampling channel 0 three times in each cycle): 0, 5, 9, 0, 0, 0, 5, 9, 0…… The waveform data is also set as int16 (2 bytes) as the digital inputs are capable of having 2 bytes comparison. Higher number of bytes for data recording will allow for more precision but it is not required as 2 bytes of floating point is sufficient. The values that are recorded will be the bit value instead of the voltage value. As the reference voltage of the Power1401 is 5V, the resolution of each bit is +/- 5/2^15 = +/- 32768. This has been scaled to the appropriate value in the *Data_Sorting.m* code. Increasing the number of bytes to 4 for the data streaming will result in a change of bit resolution.
Time-Stamps	The reason for using a .bin file is to reduce recording overhead. The formats .bin and

(.bin)	.dat are similar as there are no formatting capabilities. The reason for not using a .dat format with time-stamps is to allow future users to distinguish the three different files by the format type as opposed to the name. The timestamps are recorded in milliseconds in single format (4 bytes). The time resolution is set by the sampling frequency which is determined in *Data_Stream.m* by the variables "pre" and "cnt" (*Section 5.1.1)*. The timestamps are aligned to each set of data transfer of the waveform channels. For example: The PC is streaming 16 channels of 1000 points per channel, there will be 16,000 waveform data points for every 1000 points of time stamp points.

Event-times (.mat)	The format used for the event times is .mat due to the complexity of the information from the events. Due to the .mat format, roughly 3x the data required compared to a simple .dat and .bin file, hence this format has been used sparingly.
	
	The event-times file stores the data streaming information such as which channels are being streamed, sampling rates as well as the rising edges of the events during the experiment. These include foot-switches, green LEDs, robotic arm position etc. All of the information is stored under a *struct* format to simplify information access for users as seen below:
	
	`>> event = open('Test_28-Jul-2017_event_0001.mat')`
	
	`event =`
	
	`  struct with fields:`
	
	`     data_buffer_info: [1×1 struct]`
	`    event_buffer_info: [1×1 struct]`
	`        event_markers: {1×7 cell}`
	`           event_time: [7×34 double]`
	
	data_buffer_info stores information for the data buffer setup by the data streaming code:

```
>> event.data_buffer_info

ans =

  struct with fields:

                     Channel_Array: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]
                 Number_of_Channels: 16
              Number_of_Data_Buffers: 3
           Data_buffer_size_in_bytes: 96000
                  Single_Data_Length: 1000
    Sampling_Speed_per_Channel_in_Hz: 31250
              Time_Resolution_in_ms: 0.0313
       Time_out_for_trial_in_seconds: 30
```

event_buffer_info stores the event buffer setup by the data streaming code:

```
>> event.event_buffer_info

ans =

  struct with fields:

                 Sampling_Unit: 'U'
       Sampling_Rate_per_Unit: 1000
    Event_Buffer_Size_in_bytes: 400
```

event_markers stores information on how the event timings are stored in event_time:

```
>> event.event_markers

ans =

  1×7 cell array

  Columns 1 through 4

    '1 = Robot Position'    '2 = FootSwitch On'    '3 = Green LED on'    '4 = Red LED on'

  Columns 5 through 7

    '5 = TouchSensor On'    '6 = TouchSensor Off'    '7 = First Reward'
```

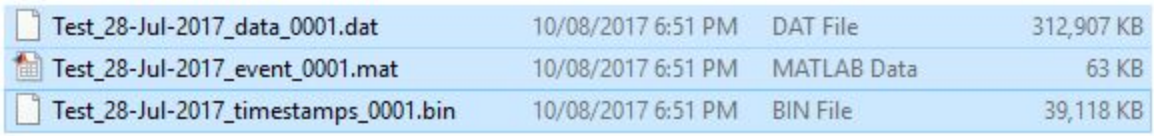
The overhead for utilising .mat files for the event information and timings is minimal as the events occur less frequently compared to recording timestamps and data.

Figure 7D. Table of descriptions of the data formats in each of the three files (.dat , .bin , .mat)

7.5.2 File Naming Conventions

The reason for using the different file formats can be seen in the previous section. Having unique file formats can act as a unique identifier between the different information the files contain. This allows for flexibility for future users to determine which files are required during analysis.

The following file naming conventions have been used:



File	File Format
Data	Name_Date_data_xxxx_.dat
Time-Stamps	Name_Date_timestamps_xxxx_.bin
Event-times	Name_Date_event_xxxx_.mat

Figure 7E. (Top) Snapshot of the naming conventions that have been adopted for the setup (Bottom) Table of the file name format

The xxxx is a four digit number (starting at 1) for each experiment that has been started on each day. The number will automatically increment by one each time the data streaming code restarts on the same day such as: 0001, 0002….9999.

To change the naming conventions, changes must be made in both *Data_Stream.m* and *Data_Sorting.m*. For *Data_Stream.m*, go to "file_setup" and change the names accordingly. The input variable "file_name" to *Data_sorting.m* must match *Data_Stream.m* in order for the the data sorting code to function properly.

7.5.3 Changing the Directory

Data_Stream.m will set the directory of where to place all the data files for the new experiments. To change the directory, open up the *Data_Stream.m* code and follow the steps outlined below:

1) Find the variable "Directory" in the code. It should look something similar to:
    a) Directory = ("C:\Users\admin\Desktop\Data\");
    b) Place your directory path into the quotation marks such as:
    Directory = (".....");

Ensure that the directory path is placed within **quotation marks** and has a **"\"** at the end. For example, if the \ is not there, it will place all the data files in the previous folder such as:

Directory = ("C:\Users\admin\Desktop\Data");

The data files will be placed within the "Desktop" folder instead of the "Data" folder.

7.6 Testing for Data Streaming

The code has been tested against two sinusoidal signals: one from the calibration signal of the AM systems 3600 Amplifier (1 kHz, 100mV pk-pk) and a second signal generator (2kHz, 100mV pk-pk). Consideration of the earth wire must be accounted for when testing with the second signal generator as earth looping may occur when both the signal generator and the Power1401 uses the same ground. Ways to bypass this includes having a separate reference point or low pass filtering the noise from the signal generator's sinusoidal wave.

To test all the possible robotic arm position, the gaze boundary from the EyeLink 1000 can be bypassed by changing the following variable in *RobotExperiment_22Feb2017.m*:

Variable: *cloudMat*

Old value:

[-500,1000;-500,1000;1500,3100;-3000,-700;-3000,-700;1500,3100;-500,1000;-3000,-700;-10000,10000];

New value:

[-8000,-8000;-8000,-8000;-8000,-8000;-8000,-8000;-8000,-8000;-8000,-8000;-8000,-8000;-8000,-8000;-8000,-8000;-8000,-8000];

This will allow the user to exceed the fixation condition of the experiment as the EyeLink 1000 has been setup to detect a macaque's eye distance instead of a human eye distance.

8.0 Data Sorting

The *Data_Sorting.m* function retrieves a file name input and will sort out the raw data which can easily be used by the user. The limitations for the *Data_Sorting.m* are outlined in *Section 12.2* and the code will output any errors for the file chosen if all 3 files does not exist with the same file number (see more in *Section 7.5.2)*. The output variables for *Data_Sorting.m* will have all the relevant information and data for the user to use (see more in *Section 8.4*).

The optimised data sorting speed comes from the data and time pointers that keeps track of where the last part of the data was read. For example, if it only analysed up to the 13th trial which is indicated by the 200,000 data point in timestamps, the data sorting code will record this number for future reference and will skip all the data points before the 13th trial. This system has sped up the data sorting code significantly.

It is also worth noting that *Data_Sorting.m* will be bypassed if less than 3 new trials have been recorded currently. Hence, the same PSTH and raster plots will be displayed on the *Main_Raster_PSTH_GUI_2.m* until the fourth trial has begun. For more information, refer to *min_trials* in *Section 8.2*.

8.1 Function Calls

The function call that was developed by the authors within *Data_Sorting.m* is:

Function definition	Description
event_sorter	Corrects some of the errors that occurs during data streaming. The errors include: robot position (correction of sequential robot positions) and touch sensor on/off (touch sensor off recorded as on). The robot position will not be necessary once the position is randomised. However, the touchsensor will be necessary.

Figure 8A. Function calls in Data_Sorting.m

As a side note, the event_sorter does not need to be called if all the suggested future fixes for hardware and software has been implemented. This is because there is a method of not refreshing the event buffer in every cycle of accessing the Power1401 but this will result in more event sorting within the *Data_Stream.m* code. Furthermore, the event_sorter is built **only** for robot positions that increments in sequential order.

8.2 Controllable Variables

The table below displays the controllable variables that exists within *Data_Sorting.m*.

Variable	Description
sp_wave_before	The number of points retrieved before the spike
sp_wave_after	The number of points retrieved after the spike
min_trials	The number of new trials required to be recorded before sorting the next batch

Figure 8B. List of the controllable variables in Data_Sorting.m

8.3 Input Variables for Data Sorting

The following table will outline the input variables for *Data_Sorting.m*. However, running the *Main_PSTH_Raster_GUI_2.m* will result in the user being able to selecting the directory and file names by simply browsing through the computer. The threshold value can also be set in the GUI.

Input Variable	Data Type	Description
threshold	Float	The threshold variable is for the user to determine what threshold voltage (in Volts) should be used to filter out the neural data into spikes. This will be dependent on the amplification used on the A-M Systems 3600. The default value for the gain is set to 10V/V so a threshold value of 0.8V has been set.
file_name	String	The file name chooses which data file will be analysed. Any of the 3 data files can be selected to begin analysis such as the waveform data, timestamps data or the event data. Any incorrect files that are chosen will return an error to the user.
directory	String	The directory variable is a string that chooses the directory of where all the data files are stored.

Figure 8C. List of the input variables to the Data_Sorting.m function

Full descriptions of each input variable can be found in the corresponding functions.

8.4 Output Variables for Data Sorting

The output variable from *Data_Sorting.m* is in a *struct* format which contains both spike data and information on the setup. The following screenshot of the structure highlights all the output information from Data:

```
Data =

  struct with fields:

      spike_waves: {16×93 cell}
      spike_times: {16×93 cell}
      event_times: [7×93 double]
    event_markers: {1×7 cell}
        data_info: [1×1 struct]
       event_info: [1×1 struct]
```

Figure 8E. Snapshot of the output struct from Data_Sorting.m

The following fields of Data has the following information:

Field	Description
spike_waves	This cell has each channel's spike waveforms arranged in a cell matrix (row = channel; column = trial number). Inside the cell matrix, each row of the channel's matrix contains a different spike which is arranged in ascending order (first row is the first spike). The first column corresponds to the time of the spike and the rest of the columns corresponds to the spike waveform. This data is all aligned to the relevant trial indicated by event_times.
spike_times	This cell has each channel's spike times. All the matrix sizes for different channels will most likely not be the same as it is dependent on the number of spikes that was recorded for the channel. Each column of the channel's vector contains a different spike time that occurred during the trial.
event_times	The event_times has the relevant event information that is aligned to spike_waves and spike_times. Each event is sorted by a different row while each column is a different trial with the first trial beginning in the first column.
event_markers	event_markers indicates which row corresponds to which event in the event_times matrix.
data_info	The data_info field contains information about the data streaming setup such as the sampling frequency, number of channels etc for the data channels.

	This will be identical to *Section 3.2.1* under the event_times format for data_buffer_info.
event_info	The event_info field contains information about the data streaming setup such as the sampling frequency, number of channels etc for the event channels. This will be identical to *Section 3.2.1* under the event_times format for event_buffer_info.

Figure 8F. A table of the struct fields from the output of Data_Sorting.m

8.4.1 Data Format Examples

8.4.1.1 *Data.spike_waves*

The figure below shows *Data.spike_waves* in MATLAB's variable window. Note that *Data.spike_waves* is a cell of matrices for spike waveforms. The following coloured boxes represent:

- Green box highlights which field is being displayed for the struct.
- Red box highlights the **channels** which are denoted by the row (16 channels in this case).
- Purple box displays the **spike waveforms** corresponding to each channel for each trial.
- For example, row 5, column 7 represents channel 5 in trial 7. It has 8667 spikes in that trial and has 51 values of voltage values and 1 timestamp value (52 in total).
- Indices with [] corresponds to the channel and trial with no spikes detected such as channel 11, trials 1 to 3.

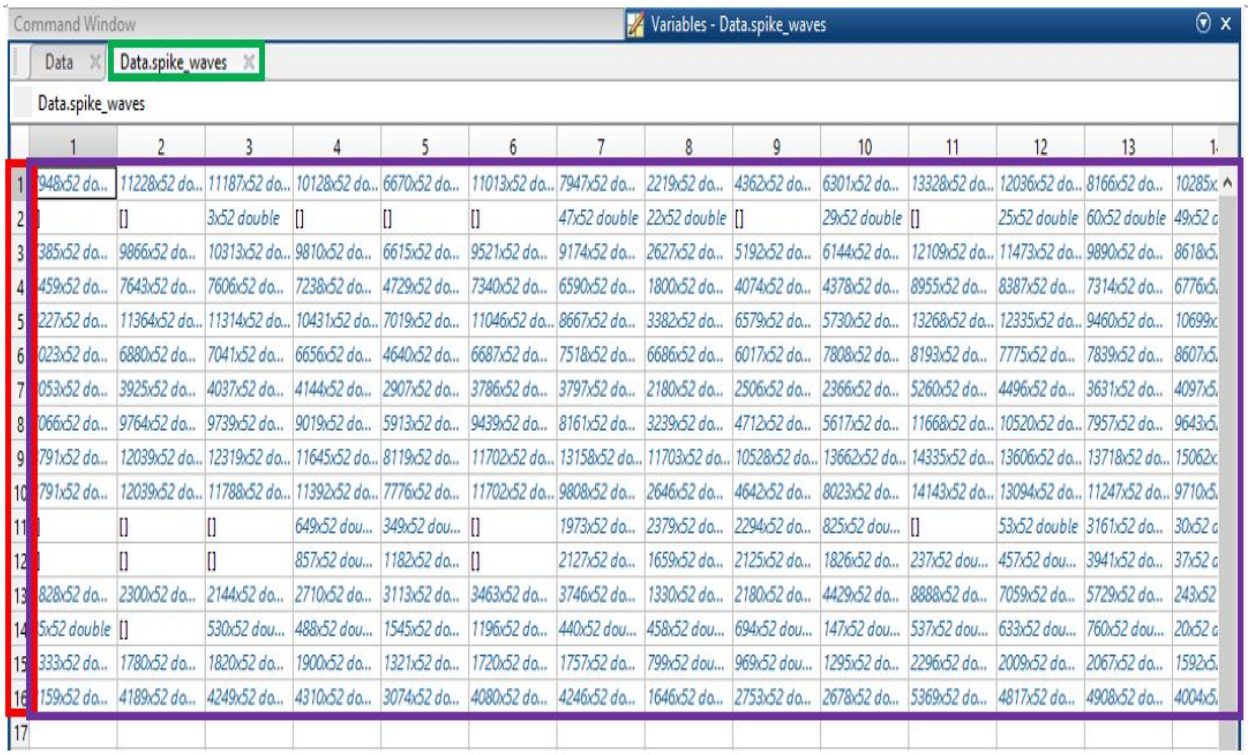


Figure 8G. Snapshot of the spike_waves field with the arrangement of trials and channels

Following the previous example, this is the matrix that existed in cell {5,7}. The coloured boxes represents the following:

- Green box highlights the cell index that has been accessed for *Data.spike_waves.*
- Red box highlights the **timestamps for each peak of the spike**.
- Purple box highlights the **voltage values** around the peak of the spike waveform.
- From the matrix, each row will represent a new spike waveform and column 1 represents the time of spike peak in milliseconds. From column 2 to the end, the values represent the voltages in V.
- Note that the **first number of every index** corresponds to the timestamp of the spike waveform and it **should not** be used as the plot for the spike waveform's voltage.

Command Window | Variables - Data.spike_waves{5, 7}

Data | Data.spike_waves | Data.spike_waves{5, 7}

Data.spike_waves{5, 7}

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0.1563	0.9464	1.0664	0.7744	0.3058	-0.0690	-0.2975	-0.4112	-0.4547	-0.4633	-0.4567	-0.4442	-0.4321
2	0.2500	0.3058	-0.0690	-0.2975	-0.4112	-0.4547	-0.4633	-0.4567	-0.4442	-0.4321	-0.4198	-0.4092	-0.3998
3	0.6875	-1.2318	-0.7941	0.0208	0.9587	1.7242	2.1921	2.2931	1.9218	1.1133	0.1674	-1.3774	-1.3640
4	1.2188	1.1090	1.8274	2.2406	2.2559	1.7981	0.9238	0.0081	-0.6313	-0.9863	-1.1462	-1.1943	-1.1899
5	1.8125	1.9701	2.3405	2.2533	1.6797	0.7512	-0.0761	-0.6099	-0.8911	-1.0074	-1.0358	-1.0245	-0.9995
6	2.3438	0.7910	1.4531	1.7877	1.6731	1.0980	0.3329	-0.2574	-0.6094	-0.7802	-0.8423	-0.8493	-0.8340
7	2.9375	1.4575	1.6573	1.4011	0.7498	0.1071	-0.3271	-0.5629	-0.6664	-0.6967	-0.6931	-0.6754	-0.6554
8	4	0.0459	0.5727	0.9598	1.0136	0.6712	0.2197	-0.1186	-0.3156	-0.4108	-0.4448	-0.4491	-0.4408
9	5.0938	-0.2672	0.0615	0.4845	0.7600	0.7036	0.3761	0.0565	-0.1593	-0.2768	-0.3305	-0.3479	-0.3493
10	5.1563	0.4845	0.7600	0.7036	0.3761	0.0565	-0.1593	-0.2768	-0.3305	-0.3479	-0.3493	-0.3436	-0.3378
11	6.7813	0.0278	0.5717	1.0361	1.2030	0.9505	0.4494	0.0206	-0.2531	-0.3992	-0.4655	-0.4883	-0.4927
12	6.8438	1.0361	1.2030	0.9505	0.4494	0.0206	-0.2531	-0.3992	-0.4655	-0.4883	-0.4927	-0.4904	-0.4866
13	7.3750	0.7257	1.2433	1.4357	1.1928	0.6154	0.0774	-0.2806	-0.4778	-0.5696	-0.6041	-0.6123	-0.6094
14	7.9375	0.9468	1.4108	1.5004	1.1386	0.4918	-0.0691	-0.4288	-0.6227	-0.7092	-0.7404	-0.7468	-0.7439
15	8.5000	1.4677	1.9777	2.0700	1.6827	0.8987	0.1149	-0.4236	-0.7262	-0.8694	-0.9245	-0.9377	-0.9348
16	9.0625	1.5276	1.9833	2.0076	1.5607	0.7372	-0.0626	-0.6064	-0.9091	-1.0518	-1.1043	-1.1159	-1.1104
17	9.6250	2.0828	2.4333	2.4480	1.9966	1.1052	0.1126	-0.6105	-1.0332	-1.2395	-1.3211	-1.3419	-1.3371
18	10.1875	2.0158	2.2983	2.3383	1.9618	1.1482	0.1276	-0.6754	-1.1642	-1.4130	-1.5163	-1.5443	-1.5410
19	10.7500	2.0787	2.2124	2.2244	1.9194	1.1430	0.0951	-0.7733	-1.3155	-1.5958	-1.7140	-1.7482	-1.7442
20	11.2813	1.7110	1.9598	2.0486	2.0538	1.9078	1.3249	0.3477	-0.6323	-1.3104	-1.6893	-1.8636	-1.9235
21	11.8438	1.6931	1.8472	1.8869	1.8716	1.7006	1.0841	0.0893	-0.8687	-1.5166	-1.8710	-2.0300	-2.0804
22	12.4063	1.6405	1.7361	1.7494	1.7258	1.6559	1.2331	0.3670	-0.6564	-1.4339	-1.8944	-2.1179	-2.2012
23	12.9375	1.3521	1.5533	1.6202	1.6212	1.5953	1.5155	1.0667	0.2063	-0.7953	-1.5555	-2.0059	-2.2208
24	13.5000	1.3913	1.5088	1.5341	1.5192	1.4903	1.4391	1.0725	0.2586	-0.7515	-1.5559	-2.0436	-2.2841
25	14.0625	1.3310	1.4421	1.4671	1.4528	1.4270	1.3766	1.0310	0.2643	-0.7069	-1.5273	-2.0424	-2.3030
26	14.6250	1.3560	1.4145	1.4178	1.3988	1.3716	1.3017	0.8655	0.0105	-0.9766	-1.7250	-2.1640	-2.3721
27	15.1563	1.1682	1.3509	1.4136	1.4182	1.4001	1.3762	1.3214	0.9315	0.1375	-0.8276	-1.6119	-2.0920

Figure 8H. Snapshot of a specific channel and trial in the spike_waves field

8.4.1.2 *Data.spike_times*

The figure below displays the format of *Data.spike_times*. The row values in *Data.spike_times* represents the peak of each spike for a given channel and trial. The data will ideally match the first column of *Data.spike_waves*. However, small discrepancies might occur if the peak of the spike times is at the start or end of the trial and there are not enough voltage values to form a spike waveform. This can occur because in order to acquire the spike waveform, the bin width might exceed the *Data.spike_times*'s index, hence causing a MATLAB error. Hence, some of the *Data.spike_times* value may be disregarded. However, *Data.spike_times* will always have equal or more rows than *Data.spike_waves*.

The coloured boxes below represents the following:

- Green box highlights the access of *Data.spike_times* for channel 5 (row 5) and trial 7 (column 7). This format is similar to *Data.spike_waves*
- Red box highlights the peak of each spike for the given channel and trial

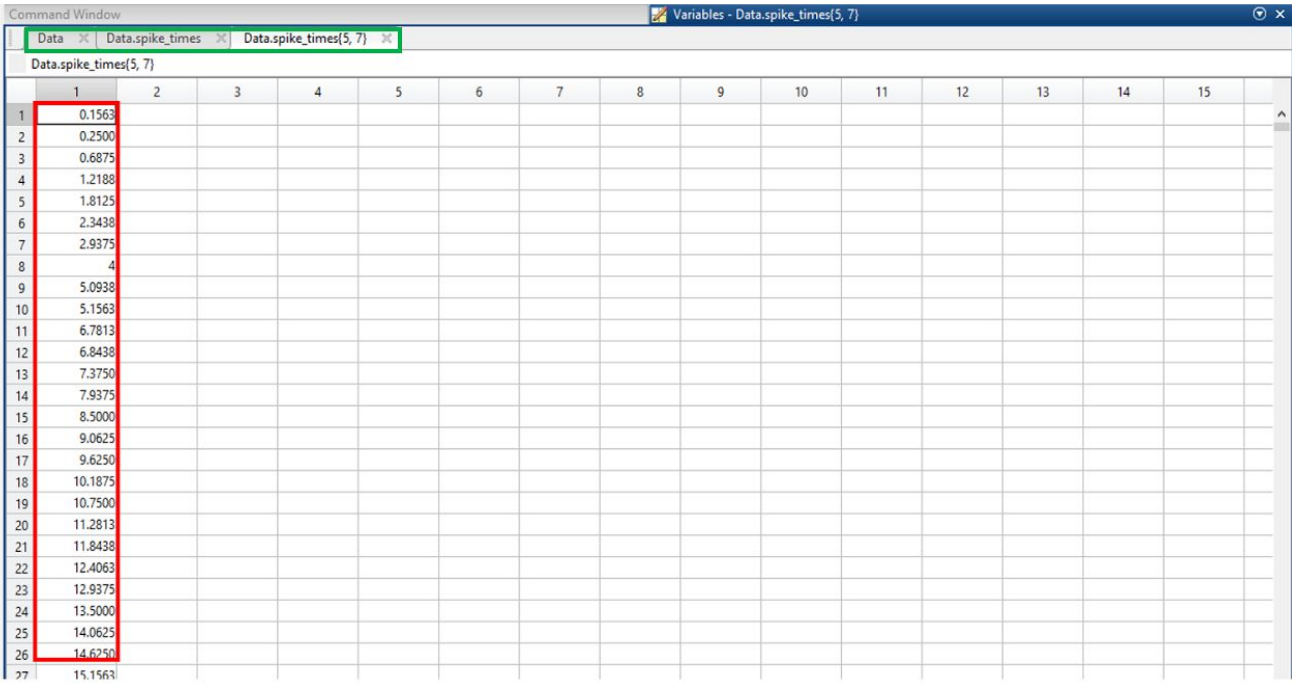
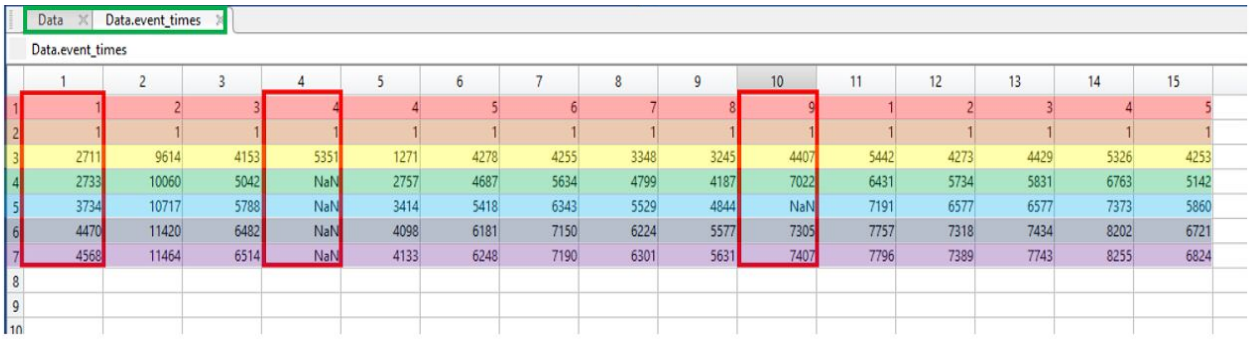


Figure 8I. Snapshot of a specific channel and trial in the spike_times field

8.4.1.3 *Data.event_times*

The figure below represents *Data.event_times* along with *Data.event_markers*. The event markers has the information of which event is represented in each row. The event times represents the occurrence of the specific event for the specific trial.



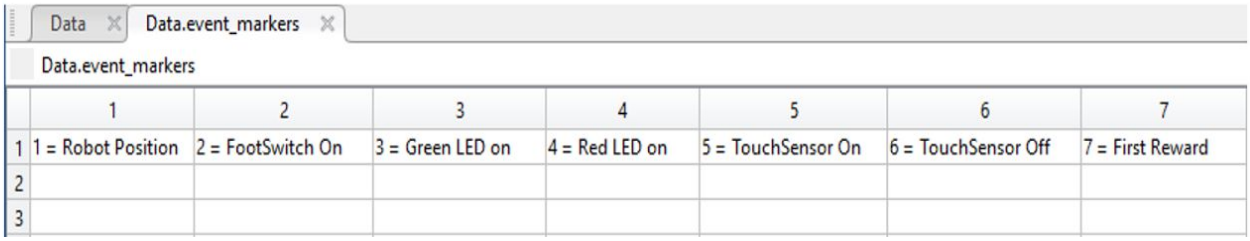


Figure 8J. (Top) Snapshot of the event_times field which displays the event and trials (Bottom) The event markers for each row in event_times

A quick recap on the events that each row represents:

- The **first row** represents the **robot position** (numbered from 1->9). The data below runs the robot positions in sequential mode (from 1->9 back to 1->9)
- The **second row** represents the **footswitch**. The "1" represents that the trial has been resetted by the footswitch and not by a timeout
- The **third row** represents the **green LED on** time. It is the most recent green LED on time for the given footswitch press (current trial in milliseconds).
- The **fourth row** represents the **red LED on** time. It is the most recent red LED on time for the given footswitch press (current trial in milliseconds).
- The **fifth row** represents the **touch sensor press**. There will only be one touch sensor value as the other arm will be released from the footswitch to press the touch sensor (in milliseconds).
- The **sixth row** represents the **touch sensor release** (in milliseconds).
- The **seventh row** represents the **first reward** that has been given for a successful trial. There will be several rewards that are given for a successful trial (based randomly) but only the first reward will be recorded in the trial. This represents the last event for a trial.

The following non-filled red boxes in the figures above represents the following:

- The first red box represents the first trial that has occurred for the experiment. For the first trial, the green LED on slot only recorded the green LED off (this can be seen because the time between green LED and red LED is minimal). This is one of the un-reliabilities of the code that has been discussed in the *current capability section* of the manual. However, the rest of the event time values are reasonably within range.
- The second red box represents the fourth trial corresponding to robot position 4. However, the subject released the footswitch after letting the green LED flash on and off several times, hence the rest of the event times have NaNs (from red LED to the reward). This is a valid unsuccessful trial.
- The third red box represents a NaN in the touchsensor on event (row 5). This is due to the fact that the touchsensor being pressed was not recorded as the streaming missed the event. Hence a NaN has been written into the touch sensor on instead.

8.5 Testing for Data Sorting

Data_Sorting.m testing can be done in offline or online mode as the function will sort out all the data in a chosen file.

For offline mode: *Data_Sorting.m* can be run by creating a MATLAB script and manually adding the three input variables as seen in *Section 5.2.1*. This will give users an output variable with all the relevant information.

For online mode: There are two .m files that calls *Data_Sorting.m*. For debugging purposes, it is recommended to use *My_Analysis.m* and loop the *Data_Sorting()* function call in order to simulate an online data streaming. The reason for not using the GUI to debug *Data_Sorting.m* is because MATLAB does not output the errors with relevant information in GUI mode and GUI has a timer which will make it hard to debug.

9.0 Data Processing

The data processing component encapsulates three parts; spike detection, post-stimulus time histogram (PSTH), and GUI for real time analysis with PSTH. Spike detection was simply done to find the maximum voltages (spikes) after crossing a specific voltage threshold. PSTH aligns the spikes to specific events (stimulus) that occurs during a trial and the GUI creates a visual representation of the performance of the macaque during the experiments.

Originally, several variations of spike detection codes were sourced from MATLAB's contributed software section. However, due to the incompatibility and inefficiency of the code, the authors of this manual wrote their own MATLAB spike detection code. This allowed us to have full controllability of the code and inefficiencies within the code was minimised. With a set of test data, the code was able to perform ~5x faster than the open source codes and will be used in the future robotic arm testing.

Similarly, different variations of PSTH codes were sourced online. The PSTH codes did not comply with the requirements of the GUI and hence the authors of this manual developed their own PSTH code to allow full controllability of the different variables which has resulted in the code performing ~3x faster than the open source codes. Variables have been fully commented in the code to allow for easy use and it allows for both analysis of spike density functions and PSTH figures. Although the PSTH code is slow, it is not a priority for further optimisation as the PSTH code is not required for the control of the robotic arm as it is only an analytical tool for neuron behaviours aligned to specific events.

Function definition	**Description**
spike_detection	This is a standalone function that detect the peak of all the spikes in a set of vector data with a given voltage threshold
psth_analysis	This is a standalone function that processes the spikes to align it around a specific stimulus. It has many input variables which allows for high controllability for the user such as plotting PSTH or spike density functions, skewing the epoch time etc.

Figure 9A. The functions that have been created for data processing.

9.1 Controllable variables for Data Processing Functions

The variables that can be controlled for *spike_detection* and *psth_analysis* can be seen below:

Variable	Function	Description
pre_epoch_time and *post_epoch_time*	*psth_analysis* *psth_analysis*	These define a window indicating the amount of time around an epoch (event). *pre_epoch_time* represents the negative time (before the event) whereas *post_epoch_time* represents the positive time (after the event).
pre_moving_bin and *post_moving_bin*	*psth_analysis*	These define the moving bin that calculates the number of spikes within the bin for each time interval.
bin_width	*psth_analysis*	This controls the gap between each time interval that will be recorded. Having a value of "1" will create a spike density function instead of a post-stimulus time histogram.

Figure 9B. List of controllable variables inside the data processing functions.

Full descriptions of each variable can be found in the corresponding functions.

9.2 Input Variables for Data Processing

A table of the input variables for the data processing functions can be seen below:

Variable	Function	Description
spike_data	*spike_detection*	A vector of waveform data that will be analysed for spikes
threshold	*spike_detection*	A static value that determines the threshold of where the spike will cross.
spike_times	*psth_analysis*	This is a cell containing channels in the row and trials in the column. In each cell is a vector containing all the spike times for the corresponding channel and trial.
event_time	*psth_analysis*	M*N matrix where M represents all the events and N is the number of trials. This will contain all the event times. The function has been built for the experimental setup.

Figure 9C. List of input variables for the data processing functions.

9.3 Output Variables for Data Processing

The following variable below is the output for the data processing functions:

Variable	Function	Description
spikes_index	*spike_detection*	Spike_index is a variable containing all the indices of spikes for the given neural data that corresponds to the time indices. This variable should be used in conjunction with the variable used to read the time data.

Figure 9D. Output variables from the data processing functions.

Note that there are no output variables for *psth_analysis.m* as the function will plot the histograms/spike density graphs automatically.

9.4 Testing for Data Processing

Both spike_detection and psth_analysis can be tested against simulated neural data created from a MATLAB script or by using the signal generators in the laboratory.

Spike_detection.m can be tested by simulating neural data by creating randomised variables with MATLAB functions. Plot the raw neural data and the corresponding indices from the output variable of *spike_detection.m* to validate the answers.

On the other hand, *psth_analysis.m* can only be tested by running the GUI while the experiment is running or on a pre-recorded set of data. The data must have at least one successful trial for each of the robot position in order for the GUI to run without crashing. The psth_analysis code will display graphs automatically corresponding to each event (from the footswitch to the reward).

10.0 GUI

The GUI is a graphical user interface for the user to see graphical outputs of the data in real time for this experimental setup. The GUI was developed by a collaborator of the authors in order for the physiology professor to analyse the neural data for his research. The GUI can expand to more features in the future.

10.1 Using the GUI

The GUI has some controllability in the code itself such as the refresh rate. Other controls can be found in the GUI itself as seen in the right panel. All the relevant information is displayed on the GUI for user convenience. The next sections will cover how to interpret the information on the displayed graphs and the variables that can be controlled.

10.1.1 The Interface

Firstly, open up the GUI by running *Main_PSTH_Raster_GUI_2.m* in a MATLAB window. The GUI will appear and as seen in the figure below, the GUI has a simple layout with outputs and controllable variables.

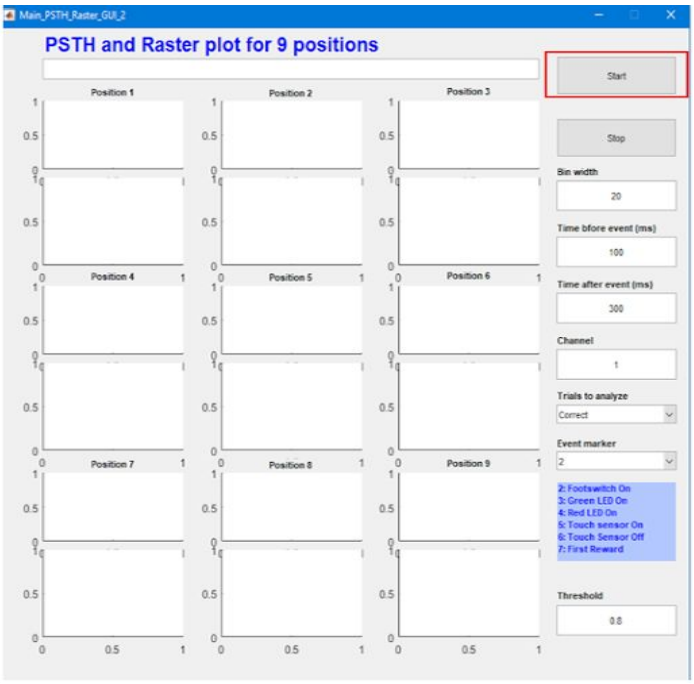


Figure 10A. Snapshot of the GUI before analysing any data

To start the GUI, press the Start button on the top right of the interface. A window will pop up for the user to select which data to analyse. Select the most recent file that has been created if the experiment is being analysed in real-time. Note that the required directory will be identical to the one that has been specified in *Data_Stream.m*. For more information on how to change the directories of the data storage, see *Section 7.5.3*

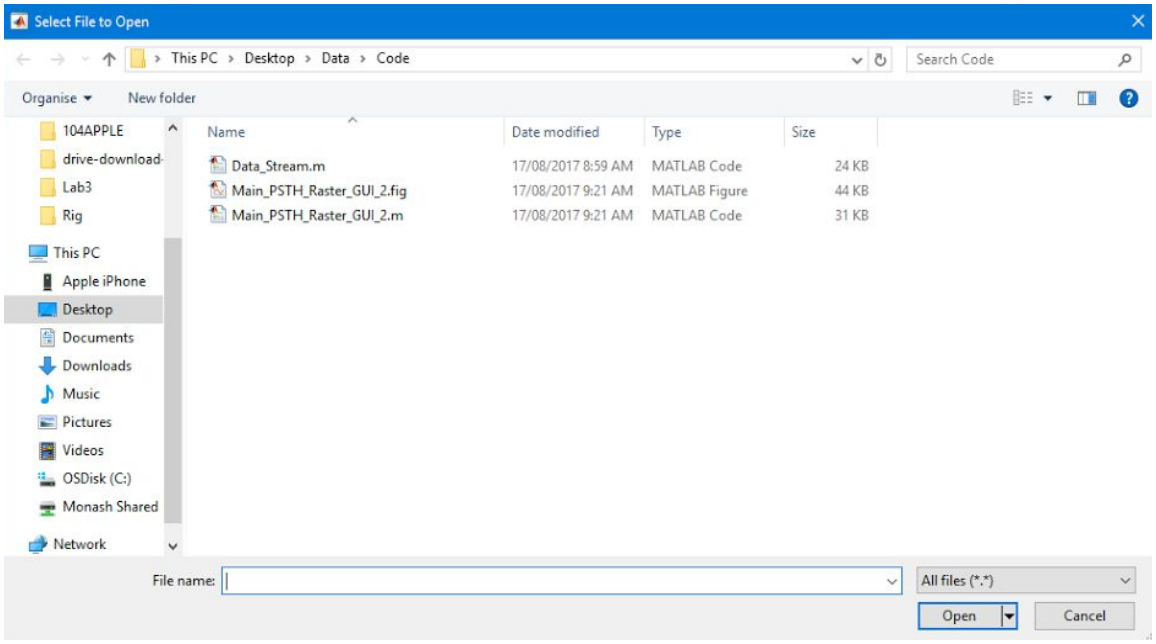


Figure 10B. Snapshot of the initial step of finding the file to be analysed

In this case, the directory of the data path is found to be:

\\ad.monash.edu\home\User031\jhcho33\Desktop\Data

The current set directory of the data path is:

C:\Users\admin\Desktop\Data

As seen in the figure below, the file path chosen in GUI matches the same directory as the *Data_Stream.m* code.

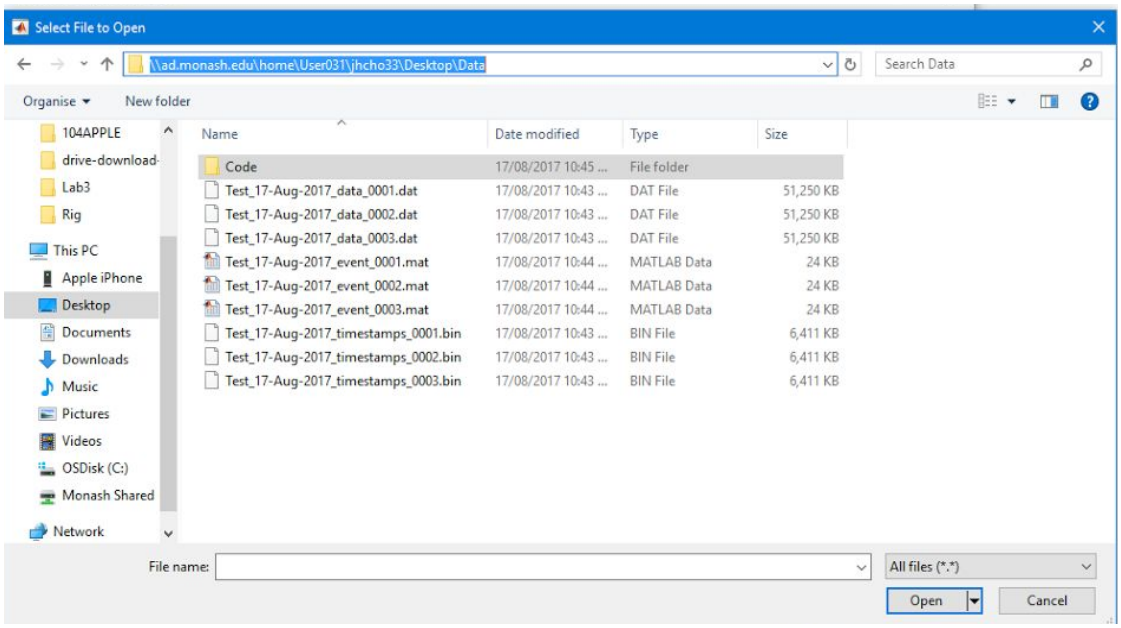


Figure 10C. Finding the directory of the data to be analysed

In the file path, there are several versions of the code that have appeared. If it is online data analysis, choose any of the three files with the largest number at the end of the file name. In the example below, the largest number is 0003. For offline analysis, pick any of the files that is required to be analysed.

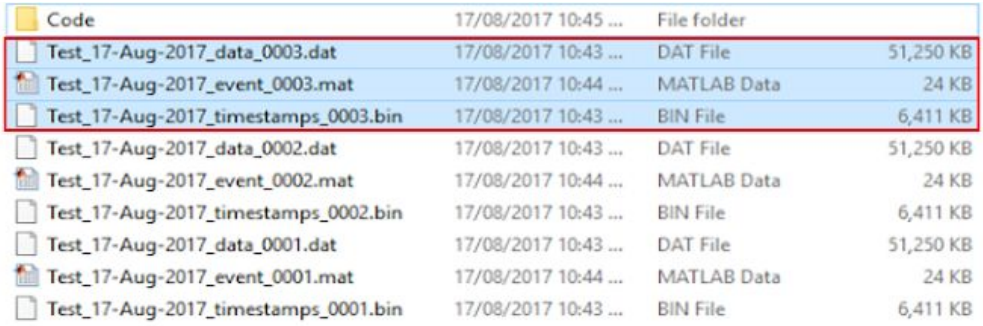


Figure 10D. Choosing the file to be analysed in the GUI

Note picking any of the three file names with the largest number is indifferent (as long as it holds the largest number). For example, the three different files are denoted by:

Name_Date_data_xxxx.dat
Name_Date_event_xxxx.mat
Name_Date_timestamps_xxxx.bin

In this case, the file chosen was the timestamps file as seen in the figure below:

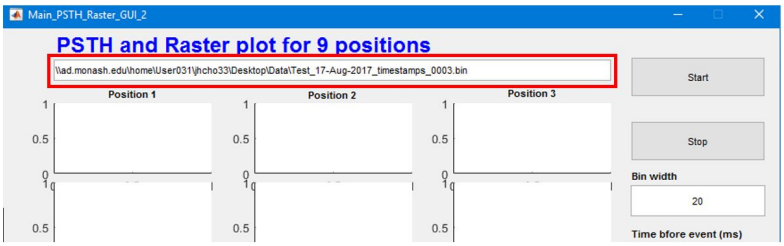


Figure 10E. The expected directory output

If the directory does not appear in the GUI, refer to the problems section of the manual found in *Section 11.1*.

If no errors have appeared in the MATLAB command window running the GUI, the user should expect some graphs appearing across the GUI as seen below:

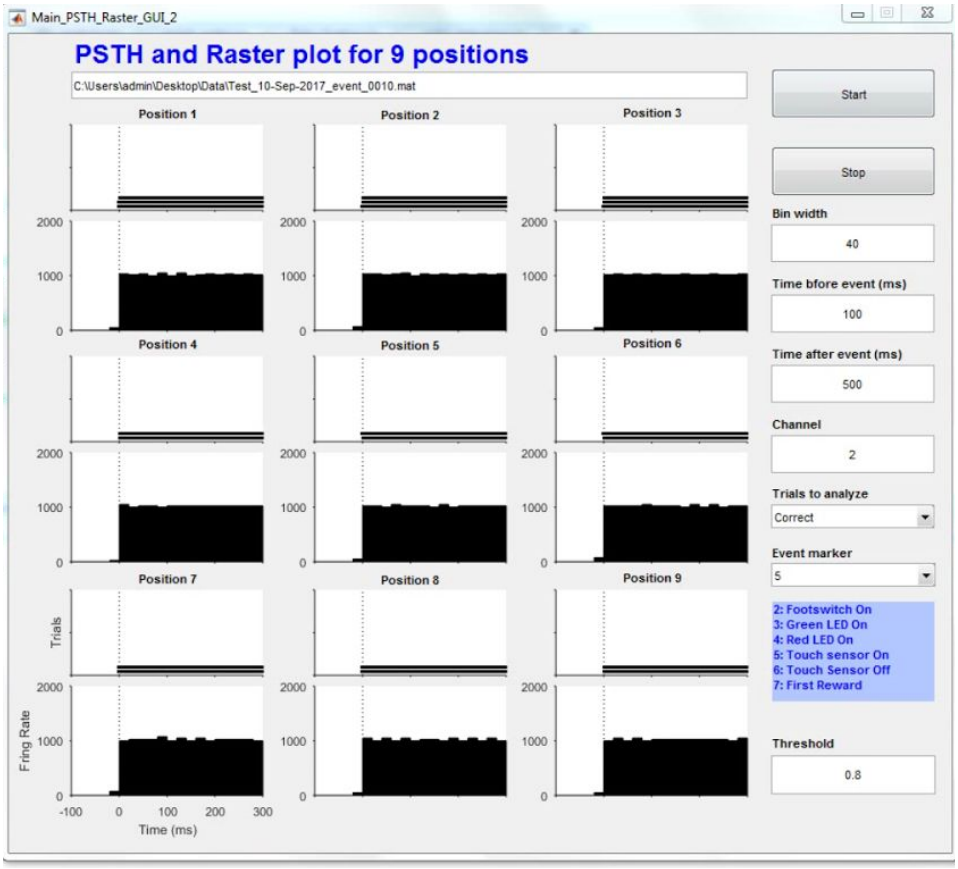


Figure 10F. Snapshot of the GUI analysing a 1kHz sine wave

Note that the data was acquired by using a signal generator to simulate neural spiking activity. As seen in the figure above, there are two plots for each position number. The plots on the top of each robot position displays the raster plot for all the trials that have occurred so far while the bottom plot displays the histograms (PSTH). The plot will automatically refresh if the "STOP" button is not pressed after a refresh time that can be set by the user as seen in the next section.

A second figure will also appear which will display the spike waveforms for each neural data channel which is plotted regardless of robotic arm position and event as seen below.

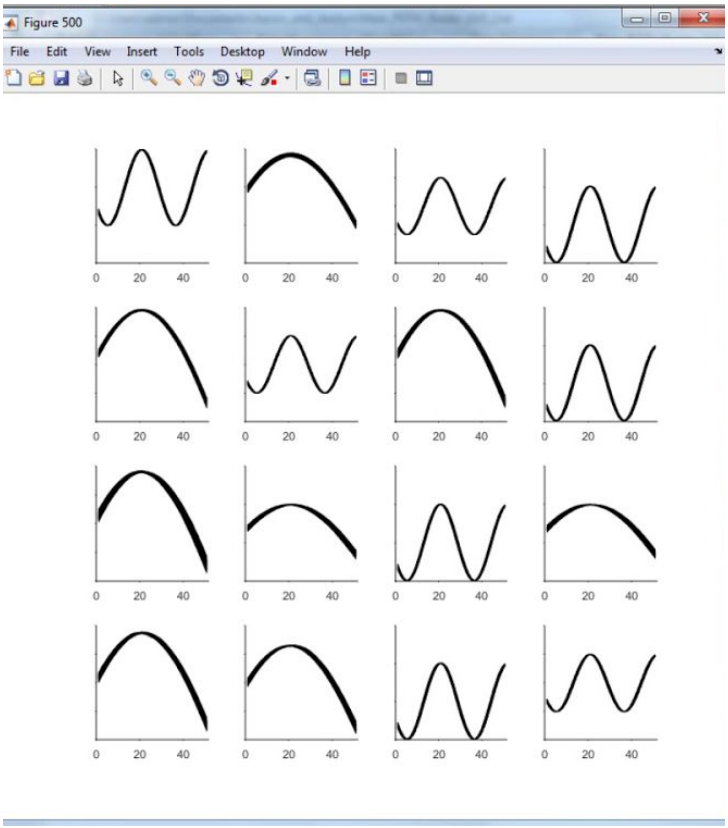


Figure 10G. Spike waveform plot of all 16 channels for the past 5 trials.

Note that the spike waveforms only plots a limited number of trials that can be set by the user. The figure above only displays the last 5 trials for each waveform channel (1-16).

10.1.2 Graphical Control for the GUI

The GUI allows for several variable controls of the PSTH and raster plots. The controls are highlighted within the red box below:

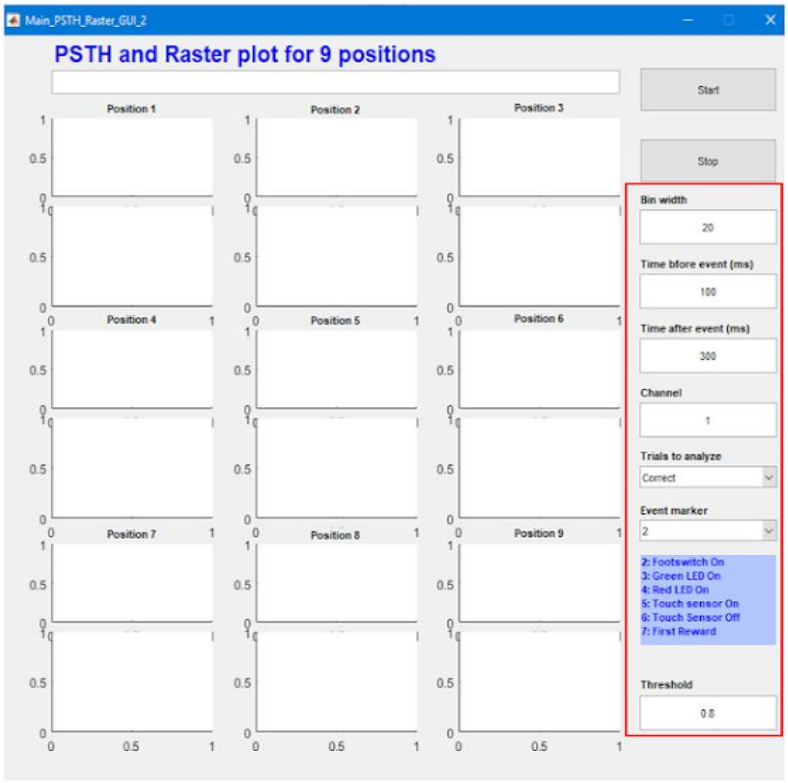


Figure 10H. Snapshot to display the adjustable variables

The table below will summarise all the settings that can be controlled by the user during the real time streaming of the neural data. Also note that the blue box in the above figure has to be changed according to the *Data_Stream* format of the event indices.

Variable	**Description**
Bin width	This controls the width of the PSTH bins when sweeping from minimum to maximum time relative to an event.
Time before event (ms)	This controls the minimum epoch time for the PSTH.
Time after event (ms)	This controls the maximum epoch time for the PSTH.
Channel	This controls the waveform channel that will be analysed.
Trials to analyse	Ability to choose to analyse successful, unsuccessful, or all trials.
Event marker	The event marker corresponds to the PSTH alignment to the event specified. The corresponding events can be seen in the blue box of the GUI.
Threshold (V)	The threshold corresponds to a static spike detection voltage threshold.

10.1.3 Controllable Variables for GUI

Within the *Main_PSTH_Raster_GUI_2.m* code, there is one variable that can be changed to suit the user and the experiment.

Variable: Period

The first variable is to set the refresh rate of the GUI. The default refresh rate of the GUI is set to 20 seconds. However, it can be increased over 20 or decreased down to 10 seconds for a faster refresh rate. The refresh rate cannot be too frequent otherwise the GUI will become unstable as MATLAB is unable to plot large amounts of data in a short period of time, hence 20 seconds has been found to be adequate for this setting.

The period can be changed by finding the following function call in the *Main_PSTH_Raster_GUI_2.m* code:

```
handles.timer = timer(...
    'ExecutionMode', 'fixedRate', ...       % Run timer repeatedly
    'Period', 20, ...                       % Initial period is 1 sec.
    'TimerFcn', {@update_display,hObject}); % Specify callback function
```

Next to 'Period' is a bolded number which indicates the refresh rate in seconds. The user can change that number before starting the GUI to control the refresh rate.

11.0 Common Problems

The following section encapsulates common problems that might be encountered when running the experiment. Limitations and potential fixes of the software and hardware will also be discussed.

11.1 Troubleshoot

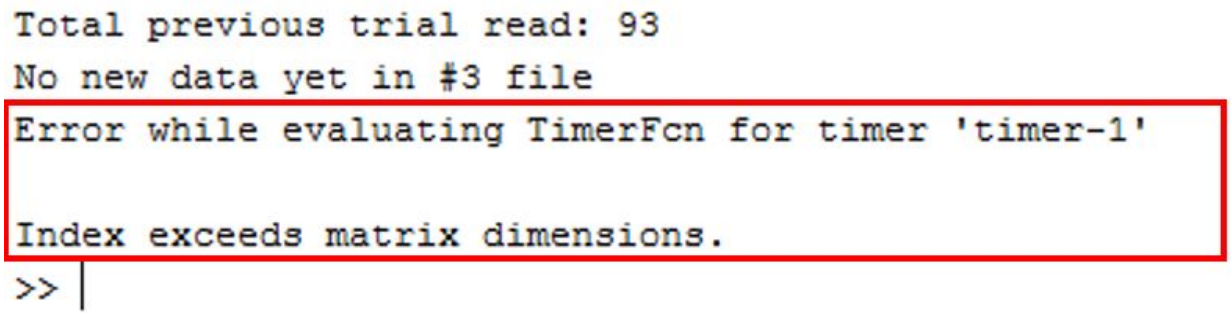
This section outlines all the small problems experienced by the authors during the experimental setup. Solutions and explanations of the problems will also be outlined with the problems.

Problem: Events 0 and 1 are not showing up (for the robotic arm's position)

Solution: Ensure MATLAB is not connected to the Power1401. Open up Spike2 and follow the steps:
Click: **EDIT -> EDIT PREFERENCES**
Select the following tab: **SAMPLING**
Tick the checkbox for: **Event ports 0 and 1 on rear digital input connector (micro and Power1401)**
Tick the checkbox for: **Sample and Play wave trigger on rear events (micro and Power1401)**

Explanation: Spike2 can setup the required digital event inputs from either the front inputs or the back inputs. By the default settings, Spike2 will only analyse the front 2 event inputs for EVENT 0 and EVENT 1, hence users must manually set the settings to the back digital inputs for EVENT 0 and EVENT 1.

Problem: The PSTH GUI is not producing any output and the MATLAB window is producing an error of:



Error while evaluating TimerFCN for timer 'timer-1'

And another error of:

Index exceeds matrix dimensions

OR

Dimensions of matrices being concatenated are not consistent.

**Note: The errors might not come out as red as MATLAB's GUI errors prints out the error in normal text colour

Solution 1: Delete the relevant sp.mat file (corresponding to the current online experiment). Wait until all the robot positions have been completed at least once (so 9 successful trials). Restart and run the GUI again.

Solution 2: If the problem still persists, restart all 3 codes (*Data_Stream.m, RobotExperiment_22Feb2017.m* & *Main_PSTH_Raster_GUI_2.m)*. Similar to solution 1, run the GUI once all the robot positions have been successful.

Explanation: The GUI code runs based on 9 robot positions for the plots, hence it is mandatory for the user to wait until 9 robot positions have been completed before running the GUI. Furthermore, MATLAB's online GUI function has defects so restarting all the codes will help fix the problem.

Problem: EyeLink Not Found (In *RobotExperiment_22Feb2017.m*)

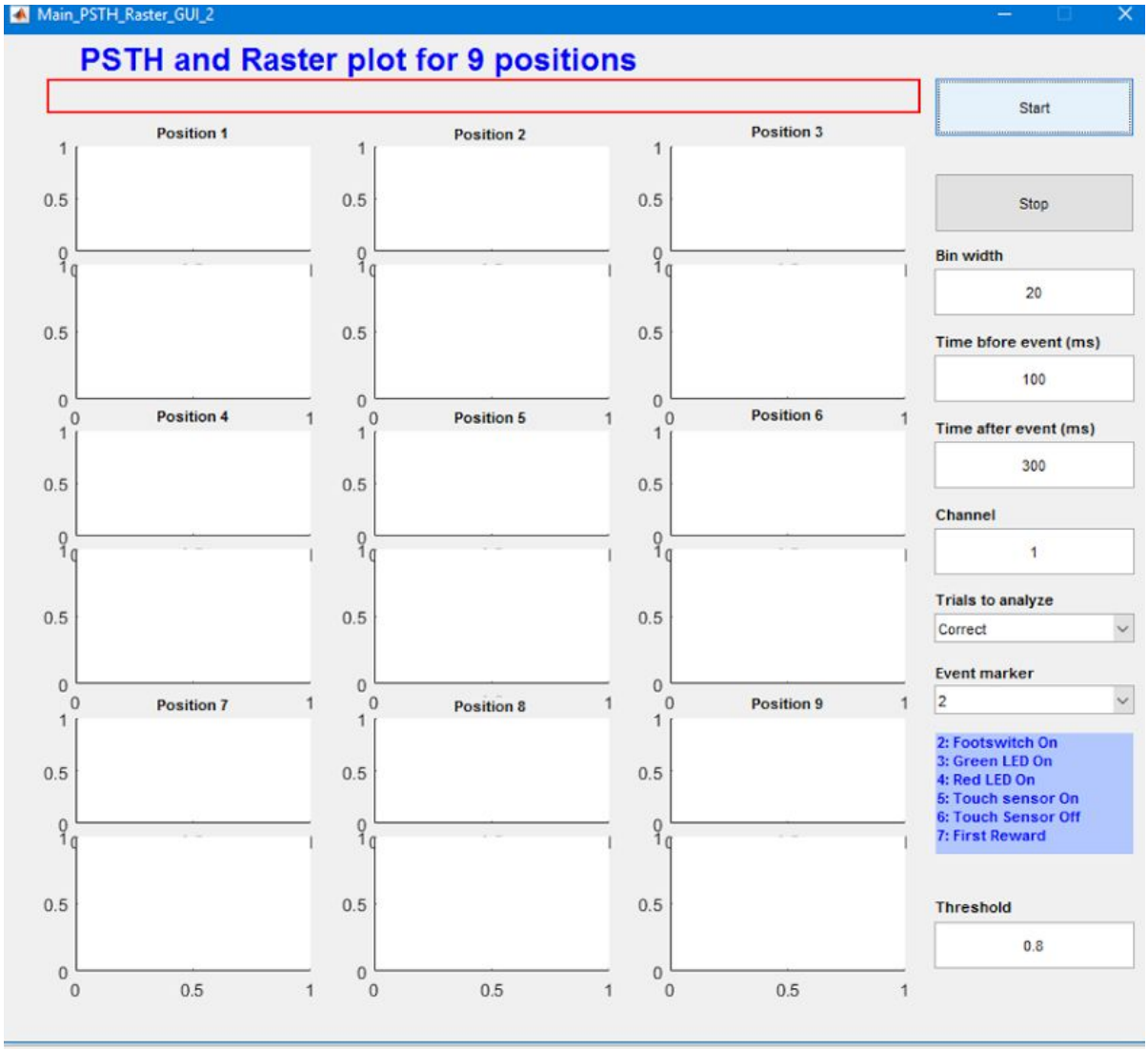
```
Error using RobotExperiment16Oct06
could not init connection to Eyelink
```

Solution: Ensure that the EyeLink PC is booted up after starting the EyeLink 1000 Camera behind the UR5 robotic arm. If the EyeLink PC was booted up first and the user is prompted to enter a command into the command window, type in "ELCL" and hit ENTER. Re-run the *RobotExperiment_22Feb2017.m* code once the EyeLink interface is displayed. Note the above error should display "RobotExperiment_22Feb2017" instead of "RobotExperiment16Oct06".

Explanation: In order for the EyeLink 1000 to boot up properly, the PC must detect an active camera before booting up. If the PC does not detect the camera, the boot up will be unsuccessful and the *RobotExperiment_22Feb2017.m* will also be unsuccessful when the code is executed.

Problem: The GUI Start Bar which displays the file path has disappeared

---



Solution: Restart the GUI by closing the GUI from the top right corner of the window. Run *Main_PSTH_Raster_GUI_2.m* again.

Explanation: The GUI code has no checking mechanism to see if a file name has been chosen, hence not choosing any directory after pressing "START" will result in the directory disappearing.

Problem: The error pops up in the GUI code: 'total_trial' does not exist OR Variable 'Outcome' not found

---

```
Warning: Variable 'Outcome' not found.
> In Data_Sorting (line 110)
  In Main_PSTH_Raster_GUI_2>update_display (line 190)
  In timer/timercb (line 30)
  In timercb (line 13)
  In timer/start (line 39)
  In Main_PSTH_Raster_GUI_2>Start_Timer_Callback (line 128)
  In gui_mainfcn (line 95)
  In Main_PSTH_Raster_GUI_2 (line 43)
  In matlab.graphics.internal.figfile.FigFile/read>@(hObject,eventdata)Main_PSTH_Raster_GUI_2('Start_Timer_Callback',hObject,eventdata,guidata(hObject))
Error while evaluating TimerFcn for timer 'timer-1'

'total_trial' does not exist
>>
```

Solution: Go to the directory of the data files and delete the file ending with sp_xxxx.mat where xxxx corresponds to the current experiment file that needs to be analysed.

Explanation: If the data sorting code has begun to sort the data, the code will create a file under the sp_xxxx.mat format. However, if the data sorting code is stopped halfway through the data sorting for the first time analysing that file (such as user ending the code), the next time the code runs will crash as one of the expected variables does not exist. In general, for any *Main_PSTH_Raster_GUI_2.m* error, try to use this solution as it may fix the error.

Problem: When turning on the Power1401, the Power1401's lights flashes infinitely

---

Solution: Close all the MATLAB windows on that PC that is connected to the Power1401. Next, restart the Power 1401 from the Power button. Wait until the Power1401's lights stops flashing before reopening MATLAB and restarting the experiment.

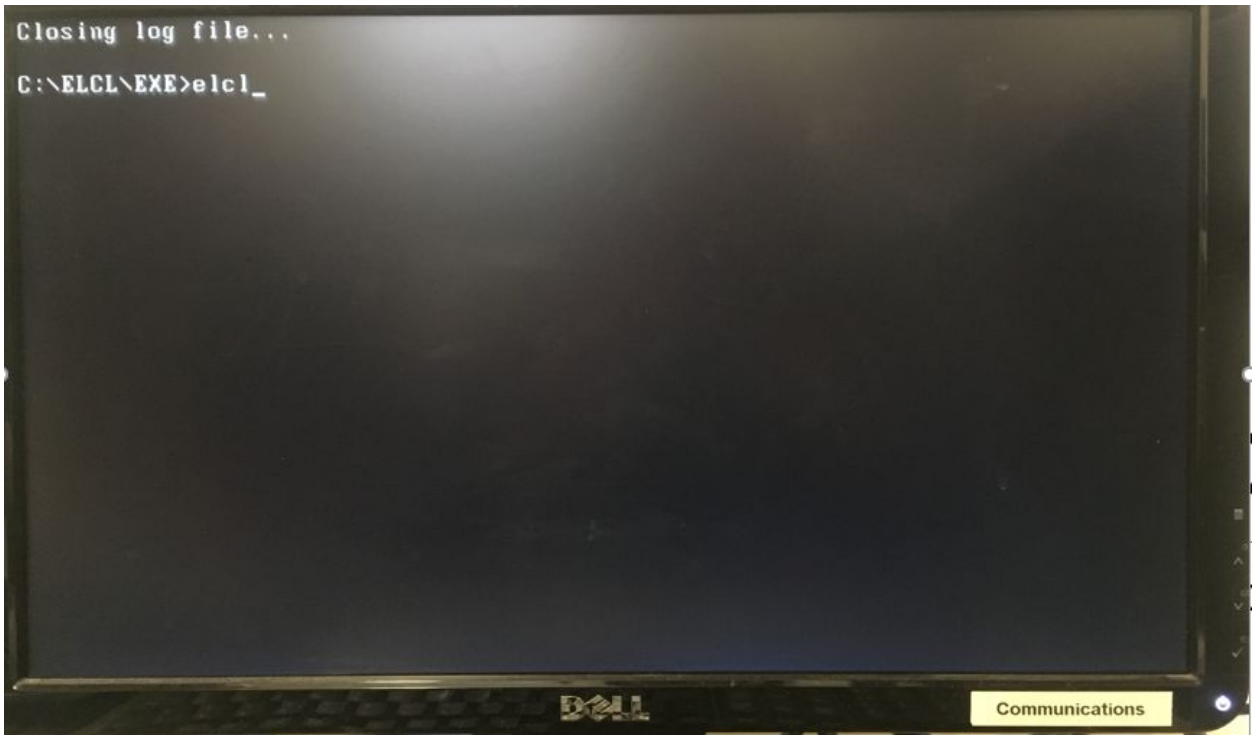
Explanation: The Power1401 has to be disconnected from the MATLAB code that runs *Data_Stream.m* by typing in:

res = matced64c('cedCloseX');

This will terminate the connection between the host PC and the Power1401, hence the Power1401 will not encounter any errors on the next bootup.

Problem: EyeLink is waiting for a command input at the bootup screen

---



Solution: Type in elcl in the command window at the bootup screen and hit ENTER.

Explanation: After an experiment ends, EyeLink will transfer all the data files into the other PC. After it finishes the transfer, the EyeLink PC will return to the boot-up mode for any further analysis if required. Hence, to restart the EyeLink, it requires a command input from the user.

12.0 Limitations

The following section will delve into the hardware and software limitations of the experimental setup. Some of the limitations are related to each other and will be linked within the explanations. Furthermore, some hardware limitations are solved by implementing software in order to create a more reliable system.

12.1 Hardware Limitations

The hardware limitations of the experimental setup is related to the software limitations as the software has been used to minimise the error that occurs from the hardware capability. The limitations that are inter-related will be referenced to each other in this section. This will give user a better understanding of the extent to how the hardware limitation was dealt with and the result.

12.1.1 Limitations for Power1401

The different models of the Power1401 has slightly different capabilities which can be found in the user manuals provided by *CED*. This results in some functions in the programming manual to not operate to the expected outcome. Hence the limited capability had to be resolved in software.

1) The 8 digital event ports at the back of the Power1401 cannot be setup with a circular timer. This has resulted in all the trials being aligned to the footswitch as opposed to the entire experiment as the reference point. The authors of this manual are unsure why this has been the case the for Power1401 but they have experienced this problem while setting up the event buffers for the data streaming.
2) Some of the commands provided in the programming manual (Progmanw) for the Powe1401 does not function to the required expectation due to different firmware versions. Hence, implementations in the *Data_Streaming.m* and *Data_Sorting.m* codes tackled this limitation.

12.2 Software Limitations

The following section will discuss all the limitations that has been encountered during the experimental setup. Most of the limitations will be due to incompatibility of MATLAB's direct communication with the Power1401. Other limitations include small glitches in the hardware of the experimental setup that was implemented before the authors of this manual started the data acquisition and processing phase.

12.2.1 Limitations for Data Streaming - *Data_Stream*

The limited accessibility to the some inbuilt functions for the Power1401 results in limitations of the MATLAB data streaming code. Other limitations arises due to the limited processing power of the PC while other limitations comes from file accessing speeds. Below are some of the limitations the authors of this manual came across and may not cover all the limitations that the data streaming code encapsulates.

1) To achieve real time streaming (under 50ms of data being streamed onto the PC), the sampling rate (currently set to 500kHz across 16 channels) must match the amount of data points to be sampled before passing on to the PC. The amount of data points is currently set to 1,000 points per channel so 16,000 for one single transfer which results in 0.032 (16,000/500,000) seconds to wait for data transfer. Increasing the amount of data points per transfer will result in two limitations:
    a) Slower real time streaming as there are more data points to wait for
    b) The amount of data streamed will be larger if the data points increases which might result in more data points missed as PC has to stream larger chunks of data from the Power1401 before being able to access the new data

   This has resulted in testing for an optimal point of data points to be set which was roughly around 800 to 1,200 data points per channel which was why 1,000 data points was chosen for this experimental setup.
2) As the sampling rate of the Power1401 cannot take floating numbers to calculate the sampling rate, the sampling frequency might exceed the amount required so the user has to ensure that there is not aliasing in the sampling when changing the number of channels required. In the data streaming code, the *data_buffer_variables* function will change the sampling frequency accordingly to keep it close to the required 30kHz for BMI applications. However, having non-integer numbers to: ***16/number_of_channels*** will result in sampling over the required frequency.

3) Due to the way the Power1401 samples the event channels by detecting a change in value (rising and falling edges), it is difficult to detect the current status of the devices such as footswitch and touch-sensor. The difficulty arises from:
    a) Power1401's event channels detecting changes in level only
    b) The Power1401 writing multiple similar values into the buffer which results in the buffer filling up quickly and the debouncing capability of the Power1401 hardware. MATLAB code has been written to detect the most correct event value that was recorded. An example of the event buffer that has been read could be:

        [444, 444, 445, 446, 447]

        Where each number corresponds to the event sampling frequency which could be seconds in this case. The above result only corresponds to one event although it seems like 5 events.
    c) To simplify the detection of the correct event, the data streaming code empties the event part of the buffer and will only read new values. This results in some cases of missed events. However, there is room for future improvements for this component of limitation which is outlined in *Section 13.0 Suggested Future Improvements*.

4) The micro-controller has its own limitations of setting multiple pins high at a time. During the testing phase of the robotic arm's position, it has been discovered that the digital input pins 4 (bit 1), 16 (bit 0) & 17 (bit 2) to the Power1401 received debouncing glitches from the micro-controller. No two pins can be set high or low simultaneously for those 3 pins connected to the micro-controller. However, all 3 pins or 1 pin can be set high or low simultaneously. This has resulted in the following problems:
    a) As the robotic arm has 9 positions for this setup, 3 bits is insufficient to cover all the positions without sending the signals twice (numbers 8 and 9). There are no more digital input ports that can be used to increase the number of bits.
    b) Further problem arises from switching on two pins simultaneously. This results in 3,5,6 being unable to be transmitted.

    The solution that has been implemented is as follows:
    a) For any possible single bit numbers such as 1,2,4, it will send the correct signal to a single pin.
    b) Number 7 can be transmitted by pulling all 3 pins high
    c) Numbers 5 and 6 will be transmitted in two parts. For example, for 5: bit 0 will be transmitted first and will be pulled low before transmitting bit 2. The two transmitted bits will be added together in the *Data_Stream.m* during real time streaming.

d) Numbers 8 and 9 will be transmitted in two parts. However, it is different to 5 and 6 as the micro-controller will pull all three pins high first (to send the number 7). The pins will be pulled low and will send 1 or 2 depending on number 8 and 9.
e) Any pins that are pulled low and are not detected will be fixed through *Data_Sorting.m* as it will ensure that any mismatch in robotic arm position will be resolved to the correct values. However, this will only work for sequential robotic arm position increments (such as 1,2,3,4...9,1,2…..9,1…). Randomisation of the robotic arm position will require future improvement of the data streaming code to mitigate incorrect robotic arm positions being read.

5) The *Data_Sorting.m* code is incapable of sorting out the interleaved data points if the initial settings for the data interleaved has the same channel initialised more than once. For example, if we set the following variable in *Data_Sorting.m* from:

channel_array = [0 1 2 3]; % Defines each individual channels

TO

channel_array = [0 1 0 0 2 0 0 3]; % Defines each individual channels

Now the channel array will set the Power1401 to sample channel 0 multiple times (5 times in one cycle). This is problematic as the code in *Data_Sorting.m* is not able to distinguish the channels. Hence it is recommended by the authors of this manual to restrict the channel setup to only once per channel for each sampling loop in the Power1401.

6) The frequency of the event that is streamed to the PC is dependent on the frequency of the data streaming. Hence, if the data streaming is not real-time, the event that is streamed will not function correctly as the event is only streamed once the data streaming has reached a certain point in the buffer.
For example: the code currently streams the data every ~30-35ms, hence the event will be streamed every ~30-35ms. Extending the data streaming to every 1 second will result in the event being streamed every 1 second. This is an undesired result as the event sorting in *Data_Stream.m* will not be able to separate multiples of one event that has occurred over the 1 second period (such as green LED turning on and off several times). Hence, to maintain the functionality of the code, it is suggested to not exceed data streaming every 50ms.

12.2.2 Limitations for Data Sorting - *Data_Sorting*

The main limitations in *Data_Sorting.m* is due to the data format as the data sorting must be able to decompose the data into useful information. Due to the structure of *Data_Streaming.m* writing three different files to disk (.bin, .dat., .mat), *Data_Sorting* must perform the data decomposition based on the .dat file (timestamps) because the time file has the data indices of when the trial resets. Hence, the limitations of *Data_Sorting.m* is as follows:

1) Must base the online trials on the timestamps data file as it has the trial reset values. This creates inflexibility in the code to analyse either the events or waveform data first instead of decoding the timestamps in order to get relevant data from the other two files
2) The function is not fully compatible with all changes to the *Data_Streaming.m* code as outlined above. Some limitations include: not being able to interleave the channels that are being sampled, changes in event.mat format etc.
3) By using the *event_sorting.m* function, *Data_Sorting* is able to correct the robot positions that have been recorded by *Data_Stream.m*. However, the correction only works with the robot position incrementing in sequential mode as opposed to randomised locations.

12.2.3 Limitations for Data Processing - *Spike_Detection & PSTH_analysis*

The main limitations that the authors encountered for the data processing functions is the non-randomised element for *Spike_Detection.m* as well as the speed for *PSTH_analysis*.

1) *Spike_Detection.m* has a static voltage threshold for detecting the spike instead of having a probabilistic model for the threshold. This might incur a higher amount of spike detection errors due to noise fluctuations in the data acquisition system.

2) To optimise *Spike_Detection.m*, the function only takes in one vector corresponding to the voltage values to a single channel and also the voltage threshold. Hence, the function only returns the index of the spike peaks instead of a time index value. This might restrict the user's understanding of the output variable of *Spike_Detection.m* as the user may not be able to utilise the function's output.

3) Due to the coding complexity of *PSTH_analysis.m,* the processing speed has been slow. Hence, other optimisation techniques can be added into the *PSTH_analysis.m* code to speed up the data processing function.

13.0 Suggested Future Improvements

This section will outline all the potential changes to the software and hardware setup to create a more robust experimental setup. The ideas below will be limited to the authors' ideas and may not be the best solutions and alternatives to the problems of the experimental setup. Changes in the hardware setup will result in changes in the software implementation, hence similar changes to both hardware and software will be outlined during the suggestions in the following sections.

13.1 Hardware Changes

Before implementing any hardware changes to the experimental setup, it is essential for the reader to follow the recommended precautions:

1) Ensure that the current users for the experimental setup have accepted the hardware changes before implementing the changes. Note that the users could be your current supervisor or other researchers utilising the same experimental setup
2) Ensure that the components that are changed or added are compatible with the power and current ratings of the existing electrical components
3) Ensure that an electrical technician has verified the additional components to minimise any damages to the existing setup

The precautions outlined above is to minimise any delays during experimental data collection for research as ordering of compatible components has a long procedure. Following the above steps will maximise the output of the project.

13.1.1 Power1401 Changes

The suggested Power1401 changes is to ease the data sorting in software by implementing hardware changes. This will allow more processing time for the data analysis code as the experimental setup will be more efficient.

1) The digital input events at the back of the Power1401 does not allow for circular buffers while the front two digital input events allow for circular buffers. Hence, it would be suggested to attach two digital events from the back of the digital input port to the front. The two suggested events would be the footswitch and the green LED. This is because those two events occur most frequently and hence has the highest chance of missing the event. However, implementing this hardware change will result in a code change in *Data_Stream.m* as the front two digital inputs has slightly different syntax to the back digital inputs.

13.2 Software Changes

The software changes below are limited to the authors' understanding of the hardware devices such as the Power1401 MKII, UR5 robotic arm, EyeLink 3600 etc. The primary reasons for the software suggestions is to minimise the amount of data missed during streaming from the authors' experiences. These suggested changes aims to tackle the limitations as discussed in the earlier sections of both the hardware and software capability of the experimental setup.

13.2.1 Data Stream Function

Optimising the *Data_Stream.m* function allows the user to minimise the loss rate in event and waveform data. Although it is currently running at a low loss rate, the loss rate can be further minimised by implementing the following pseudo-code suggested by the authors of this manual. Furthermore, testing of the following solutions can be done by following the instructions in *Section 7.6* of the manual

Solution: To implement a variable data buffer pointer

Code to change:

- All the buffer checks can be deleted within the *Data_Stream.m* main streaming loop.
- Retrieving the data waveform channels from the Power1401 must have a variable buffer size and a variable buffer pointer instead of the static ones.
- Set the time variable to initialise a sufficient size to fit each data point in the channel buffer.

Pseudo-code:

1) Initialise a sufficient data vector to fit the size of the whole waveform data buffer
2) Poll the buffer pointer to see where the data is filled up to and start the transfer from the previous point to the current point.
3) Buffer overflow of pointer must be accounted for within the code.
4) Add a variable buffer transfer with the MATLAB command:

   matced64c('cedToHost', ……..)

   Note, the syntax for cedToHost will be in the format specified by the Use1401. Refer to *Section 5.3.2 and Section 5.3.3* for more information.
5) Store the waveform data to disk.
6) Save the timestamp values according to how many waveform data points were stored.

Explanation:

The reason for applying this is to maximise the processing time allowed by the MATLAB. The current buffer setup limits the processing capability of MATLAB as MATLAB has to pend until the current buffer has filled up to the required amount. By using a variable buffer pointer to transfer the waveform data to disk, MATLAB will continuously process the data efficiently while also having more time to do the event time processing. This will further minimise the data loss rate in the streaming component.

Author's note:

The reason for not implementing the above steps is because there was insufficient time of testing due to the rushed software setup for the physiology researcher. Furthermore, the suggested code changes and pseudo code may not be exhaustive as unforeseen errors might occur. Therefore, it is imperative for the user of this manual to test the changes of the code by following the instruction in *Section 7.6*.

Solution: To not reset the events continuously

Code to change:

- To not reset the event buffer after every data transfer from the Power1401 to the PC by deleting this line of code after reading the event data from the buffer:

*matced64c('cedTo1401',no_of_dig_inputs*event_buffer_size,buffer_size+2,empty_event);*

Pseudo-code:

1) Transfer all the event data from all 8 event buffers (as implemented)
2) Sort each of the event data by dividing the data into 8 separate parts
3) Read if each of the event buffers have filled more by creating a separate pointer or variable that tracks the number of points filled for each event
4) Implement a software debounce of the event values
5) Ensure that all changes in events have been accounted for because each transfer takes ~30-35ms due to the data buffer. Hence, it is possible for some events to encounter two changes in events during that delay so it is imperative to extract all the changes of events from the event data
6) Reset the event buffer and event clocks at the start of the footswitch press (as implemented)

Explanation:

If the events are not emptied out at every data block transfer, that will result in all the event data being retained. This will result in better accuracy for the event times that occur during the experiment and will result in zero errors if implemented correctly.

Author's note:

The reason for not implementing the above steps is because there was insufficient time of testing due to the rushed software setup for the physiology researcher. Furthermore, the suggested code changes and pseudo code may not be exhaustive as unforeseen errors might occur. Therefore, it is imperative for the user of this manual to test the changes of the code by following the instruction in *Section 7.6*.

Solution: Adding EyeLink fixation condition as one of reset condition

---

Explanation:
If the events are not emptied out at every data block transfer, that will result in all the event data being retained. This will result in better accuracy for the event times that occur during the experiment and will result in zero errors if implemented correctly.

Author's note:
The current experimental setup did not require the EyeLink fixation as a priority addition to the code. However, for future similarly controlled experiments, users should add the EyeLink fixation as one of the reset conditions to the *Data_Streaming.m* code as it will improve the data streaming capability.

13.2.3 Data Processing Function

The main changes that can be made to the data processing functions is to optimise the algorithm to speed up processing. Below are some changes that can be made:

Solution: To not analysis each spike index for every bin (PSTH_analysis.m)

Code to change: The for-loop for the cumulation of the spikes can be changed to a while-loop instead to decrease the amount of values that needs to be checked as opposed to checking every spike value.

Pseudo code:

1) Instead of the for-loop, have a nested loop conditions that have the following:
    a) Not the last spike value
    b) Based on the previous bin, set the starting range from the next spike index value
    c) Analyse for any spikes within the current range and exit once the spikes exceed the bin range.
    d) Repeat steps b and c until the all the bins have been processed.

Explanation:

The code is not required to sift through every spike that has occurred in the waveform channel. Instead, the code can analyse from the previous bin until the next bin in order to optimise the data processing speed.

Author's note:

The main reason for not implementing this algorithm for *PSTH_analysis.m* is due to the fact that other functions have a higher priority to be optimized first. This is because *PSTH_analysis.m* does not have to be a real time data processing function, hence the current version of *PSTH_analysis.m* is adequate for data processing.

14.0 References

14.1 Table of Important Links

The table below outlines the important links that were used during the construction of the experimental setup. Other links includes open-source functions that were used and tutorials to give users a better understanding of the manual.

Important Links	
A-M Systems 3600	http://www.a-msystems.com/p-256-model-3600-16-channel-extracellular-amplifier.aspx
Eye-Link 1000	http://www.sr-research.com/index.html
MATLAB psych toolbox	https://github.com/Psychtoolbox-3/Psychtoolbox-3/tree/master/Psychtoolbox/PsychBasic MATLAB functions that are required to run *RobotExperiment_22Feb2017.m*
MEX tutorial (highly recommended)	https://www.google.com.au/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0ahUKEwib-7W68N3VAhWIwbwKHWWKDKUQFggmMAA&url=https%3A%2F%2Fwww.researchgate.net%2Ffile.PostFileLoader.html%3Fid%3D551a2e59cf57d7620c8b463b%26assetKey%3DAS%253A273746584113162%25401442277679173&usg=AFQjCNFiENIZ48feYSQ90BltfAiPpqv27g Automatic pdf download link
Micro-controller manual	http://www.mccdaq.com/pdfs/manuals/USB-1208FS.pdf
Power1401 MKII	http://ced.co.uk/img/power2.pdf
Spike2 (GUI)	http://ced.co.uk/img/Spike7.pdf
Spike2 (Language)	http://ced.co.uk/img/Script6.pdf
Spike2 (Open source)	http://ced.co.uk/downloads/contributed
Spike2 (Training)	http://ced.co.uk/img/TrainDay.pdf
Spike2 (Videos)	https://www.youtube.com/playlist?list=PLSMI_zyqtBlw_2pK2VE2aFpHOO3CQLHZp

Use1401 and Progmanw manuals	http://ced.co.uk/downloads/1401downloads Install 1401 Programming Support for the manuals
UR5 robotic arm	https://www.universal-robots.com/products/ur5-robot/

15.0 Glossary

15.1 Table of Definitions

The table below will outline technical terms that have been utilised throughout the manual. The glossary will cover both engineering and physiology terminologies.

Term	Definition
BMI (Brain Machine Interface)	A direct communication between a brain and an external device such as a prosthetic arm.
NaN	Not a Number. This is written in the MATLAB code to represent some of the event data to highlight that some values were not recorded during the experiment.
Neural data	Data that is acquired from the brain.
PolyScope	Graphical controller for the UR5 robotic arm.
Spike	An electrical signal that carries information which is sent by a neuron to another neuron.
Raster plot	A plot that contains spikes that are aligned with a specific external stimulus which indicates when a spike has occurred.
PSTH (Peri-Stimulus Timing Histogram)	A built up of raster plots to create a histogram of spiking histories that are aligned with an external stimulus.

15.2 Table of MATLAB functions

The table below describes the functionality of each MATLAB function briefly. For a comprehensive description of the functions, refer to the software section of the manual.

Function	Description
Data_Stream.m	To stream data off the Power1401 in real time
Data_Sorting.m	To sort the data that has been recorded and provide users with a usable set of data as the output
event_sorter.m	Sorts out any errors that have occurred during the event data streaming
spike_detection.m	To detect the peak of spikes
psth_analysis.m	Builds PSTH based on the spikes detected relative to an epoch

16.0 Appendix

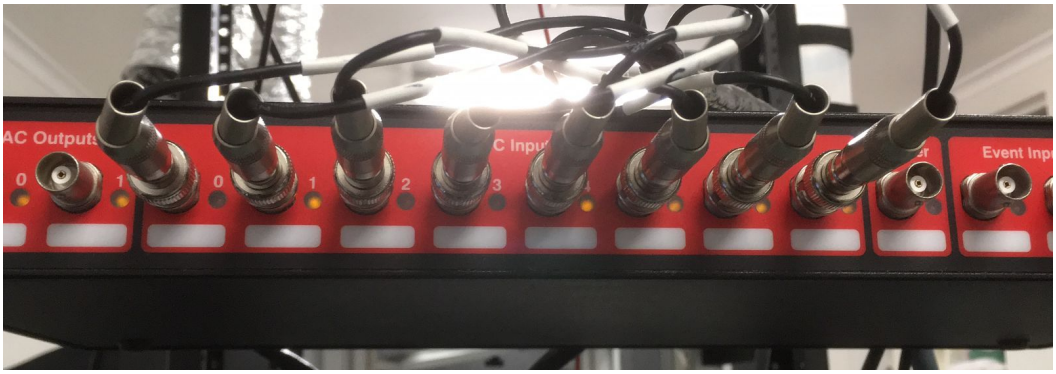
16.1 Setting up the Power1401

Turn on the Power1401 by pressing the power button as seen in the red box.

The lights will eventually turn on for the ADC inputs as seen below.

More lights….

And more lights…



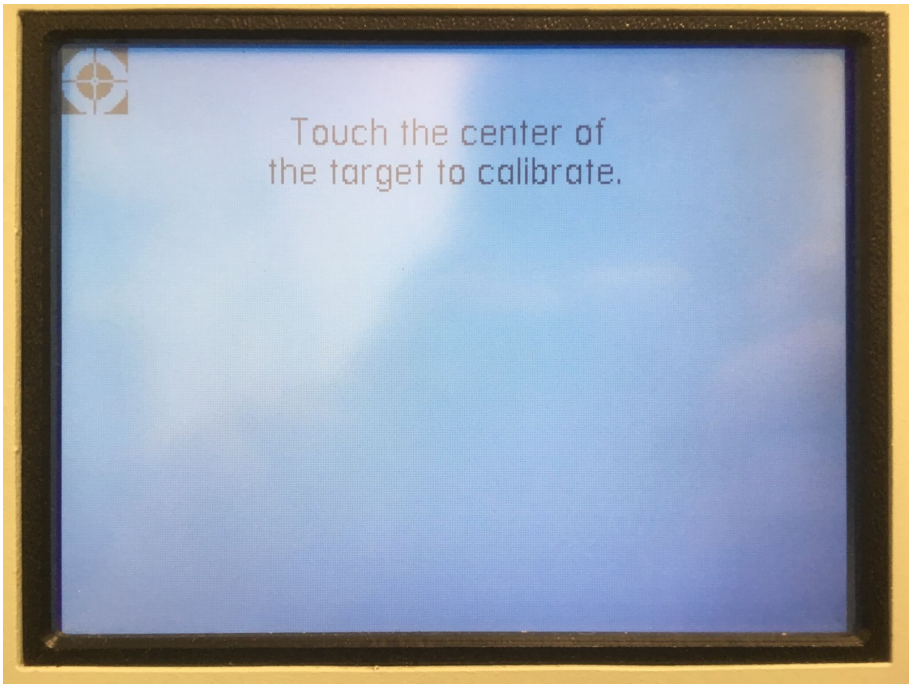
The lights will turn off and the power button should be green as seen below.

If the Power1401's power button remains red and the lights flash continuously, refer to the *troubleshoot* section of the manual to fix the problem.

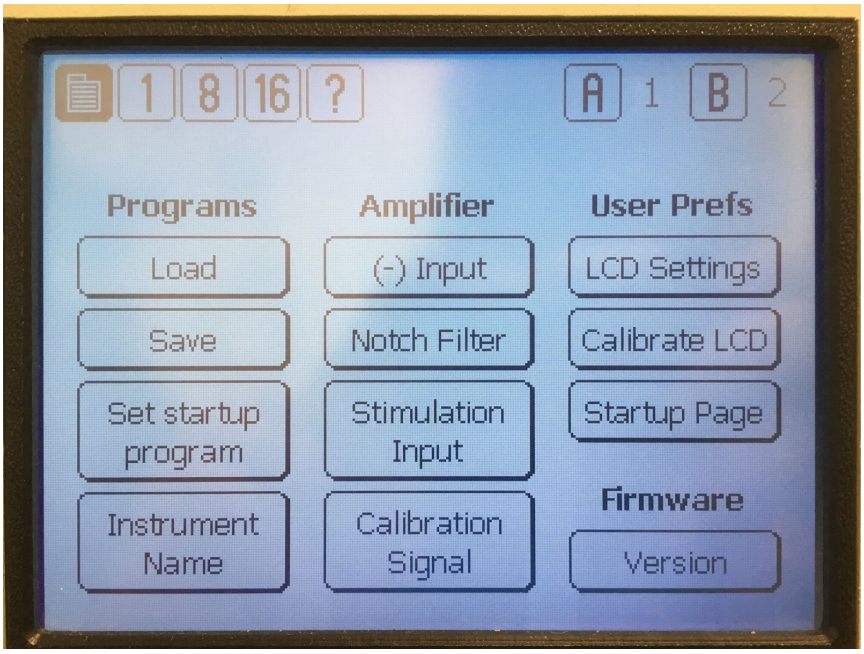
16.2 Setting up the A-M Systems 3600

Turn on the A-M System 3600 by pressing the power button as shown in the red box below.

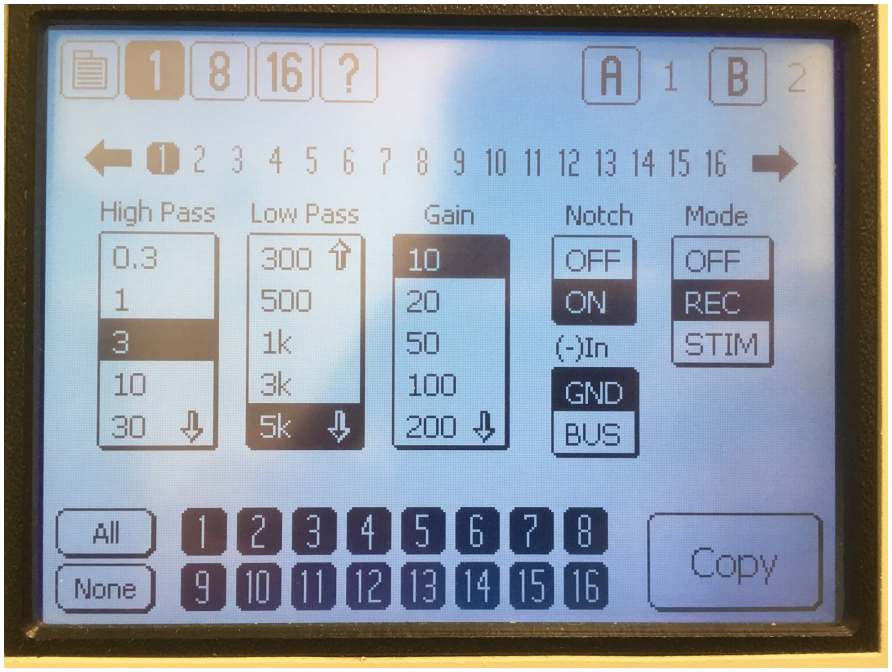
The screen will turn on and will prompt the user to follow the instructions to calibrate the touch screen.



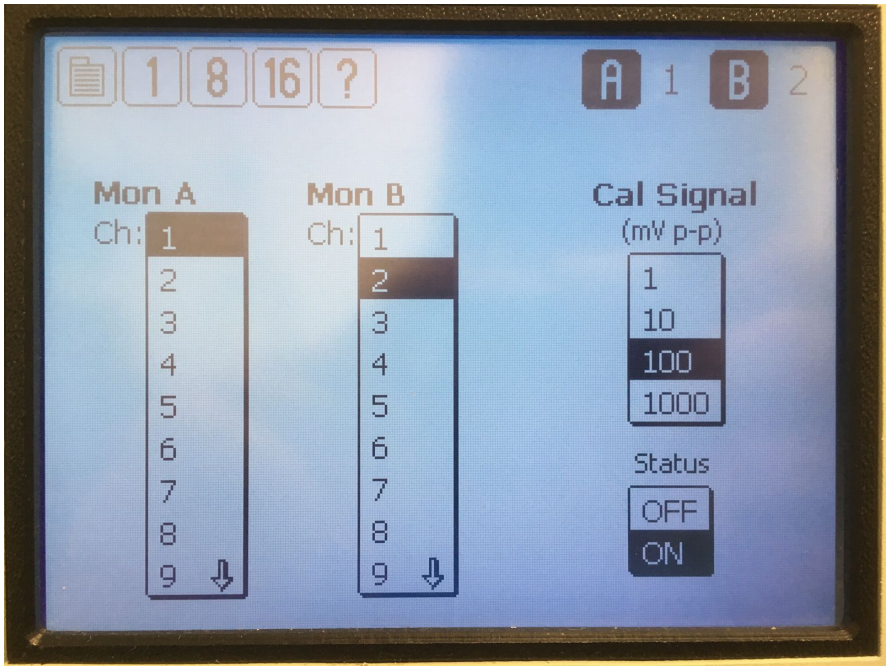
The home screen will appear after a successful callibration.



Select “1” on the home screen and the image below will be the next screen. Choose the Low Pass Filter (recommendation of 3kHz). Turn Notch “ON” and turn Mode to “REC”. Press “COPY” to copy the settings of the current channel to all the other channels



OPTIONAL: For testing purposes, press the button “A” on the top right corner of the screen.



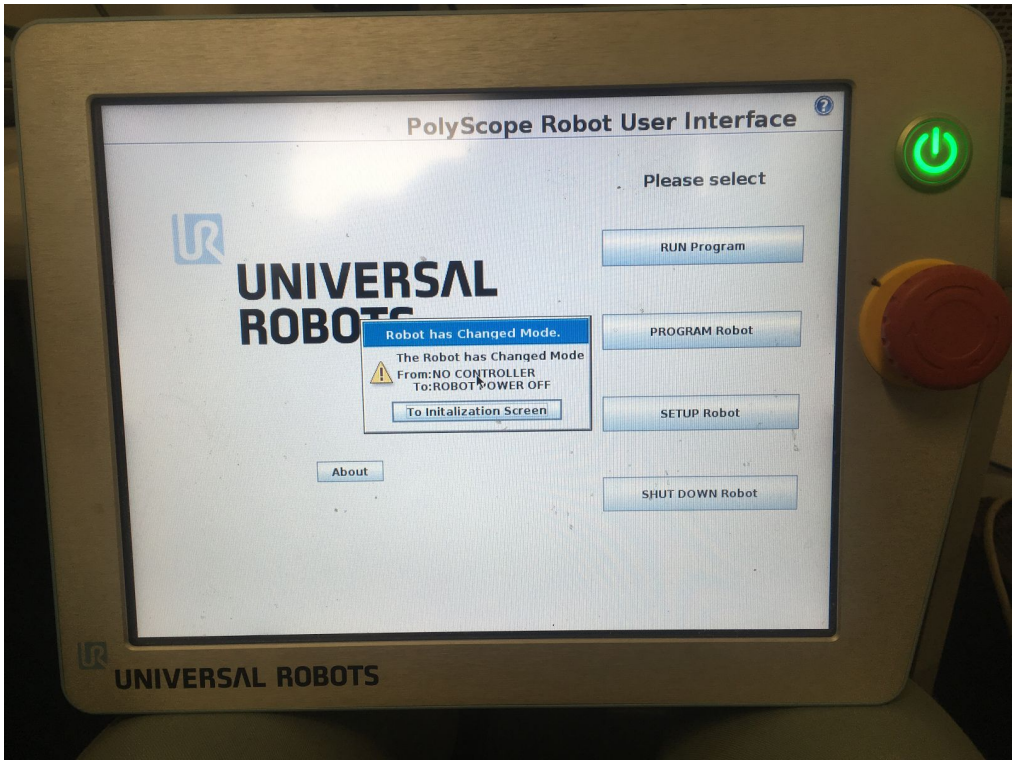
16.3 Setting up the reward systems

Turn on the reward systems by flipping the switch in the red box.

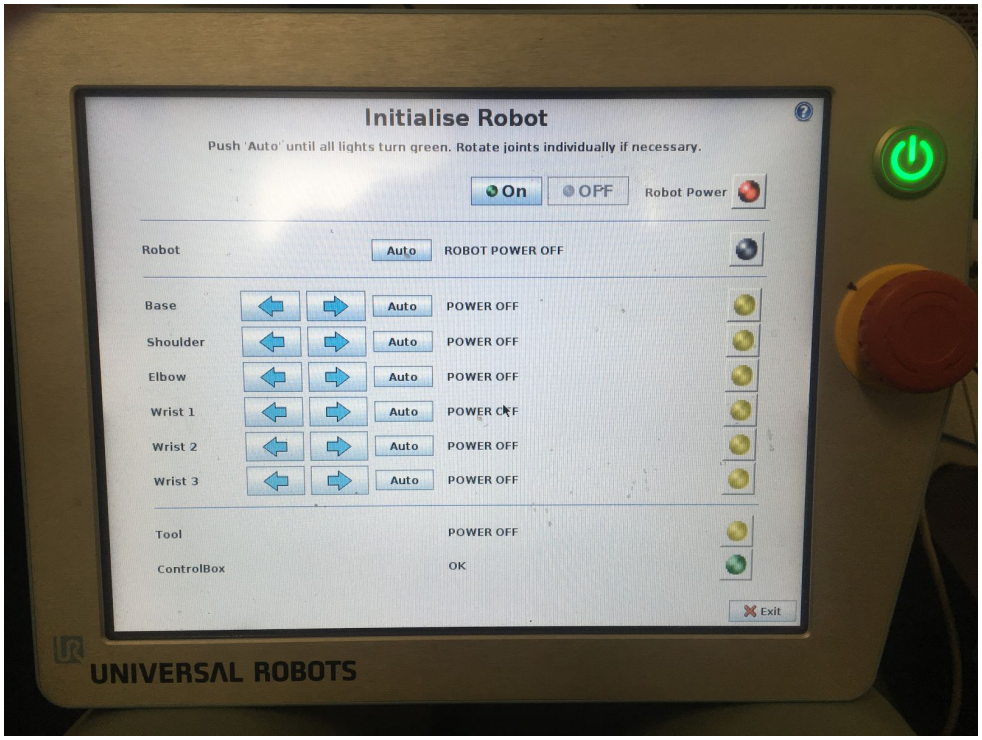
16.4 Setting up the UR5 Robotic Arm

Turn on the controller (PolyScope) that is used for the UR5. It should be located under the table of the EyeLink PC monitor.

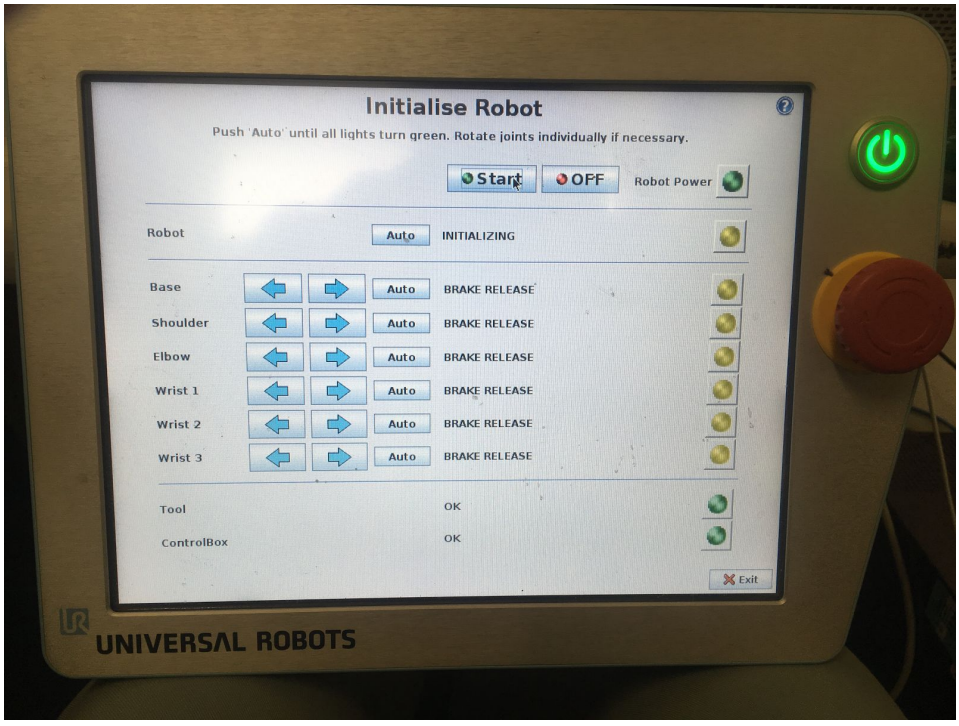
The following screen will appear:



Press “To Initialization Screen” and the next screen will appear:

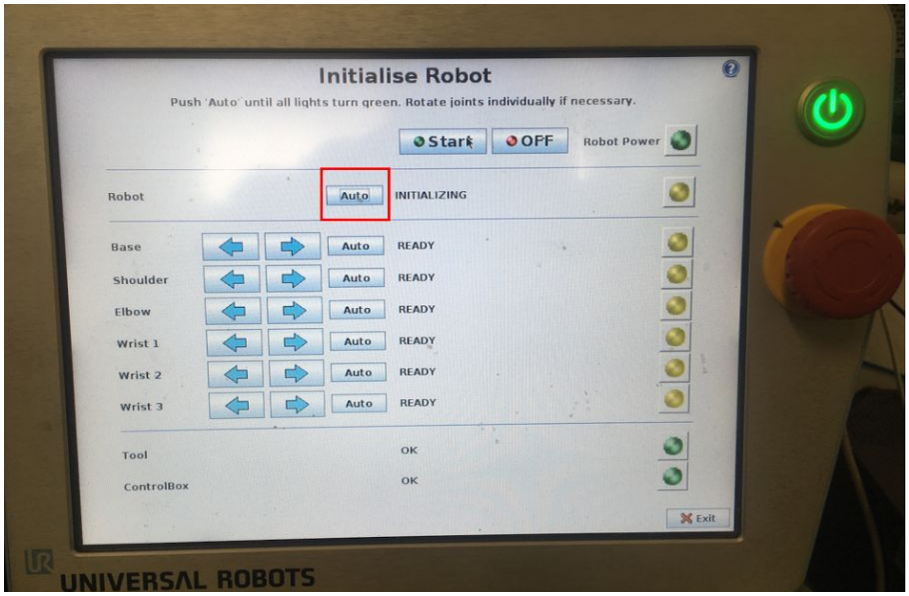


Press “On” to start the initialisation of the robot. A “Start” button will appear (in the place of the “ON” button) as seen below. The sounds of the robot joints will be audible which indicates that the UR5 robotic arm is ready to be calibrated for each joint once the brakes have been released.

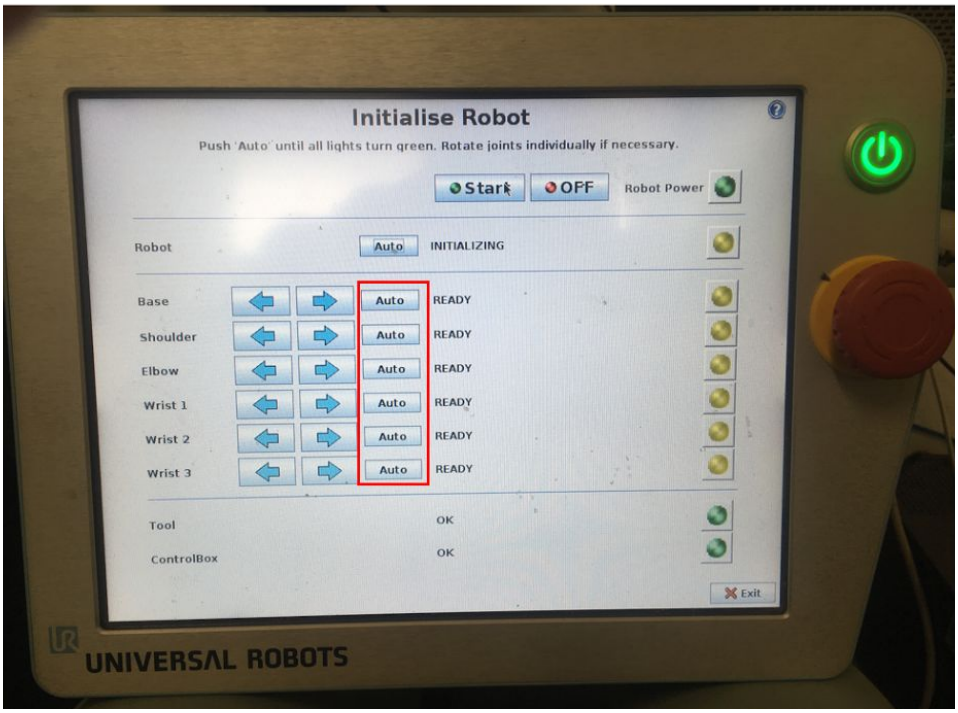


Before you proceed please read the following:

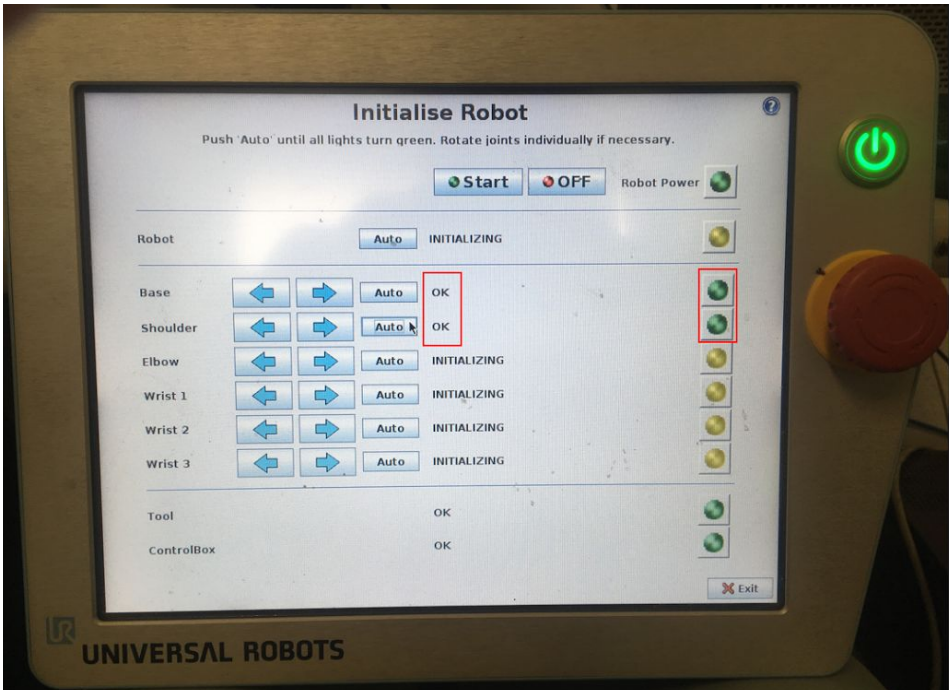
Because there are other devices that are around the UR5 robotic arm, it is imperative that the user ensures that the calibration of the arm does not hit any of the devices in the vicinity. Furthermore, be ready to press the emergency red button if the robotic arm seems to be out of control. Lastly, **DO NOT** press the "Auto" button that is next to the "Robot" text. This can be seen below:



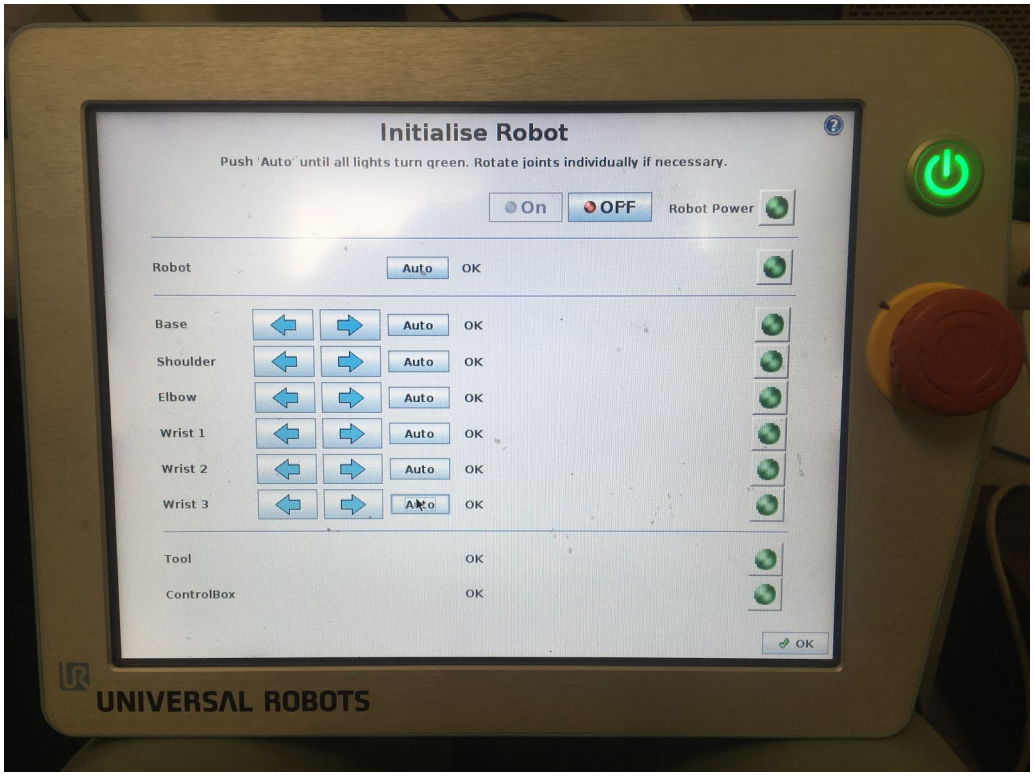
The following steps below involves calibrating each joint. This can be done by holding the "Auto" button for each joint separately. If the arm is moving towards the direction of an object, release "AUTO" and press "AUTO" again to move the arm towards the opposite direction.



After each joint has been calibrated, the a green circle will appear next to the joint with an “OK” text. Ensure someone is keeping track of the robot’s movement during calibration.

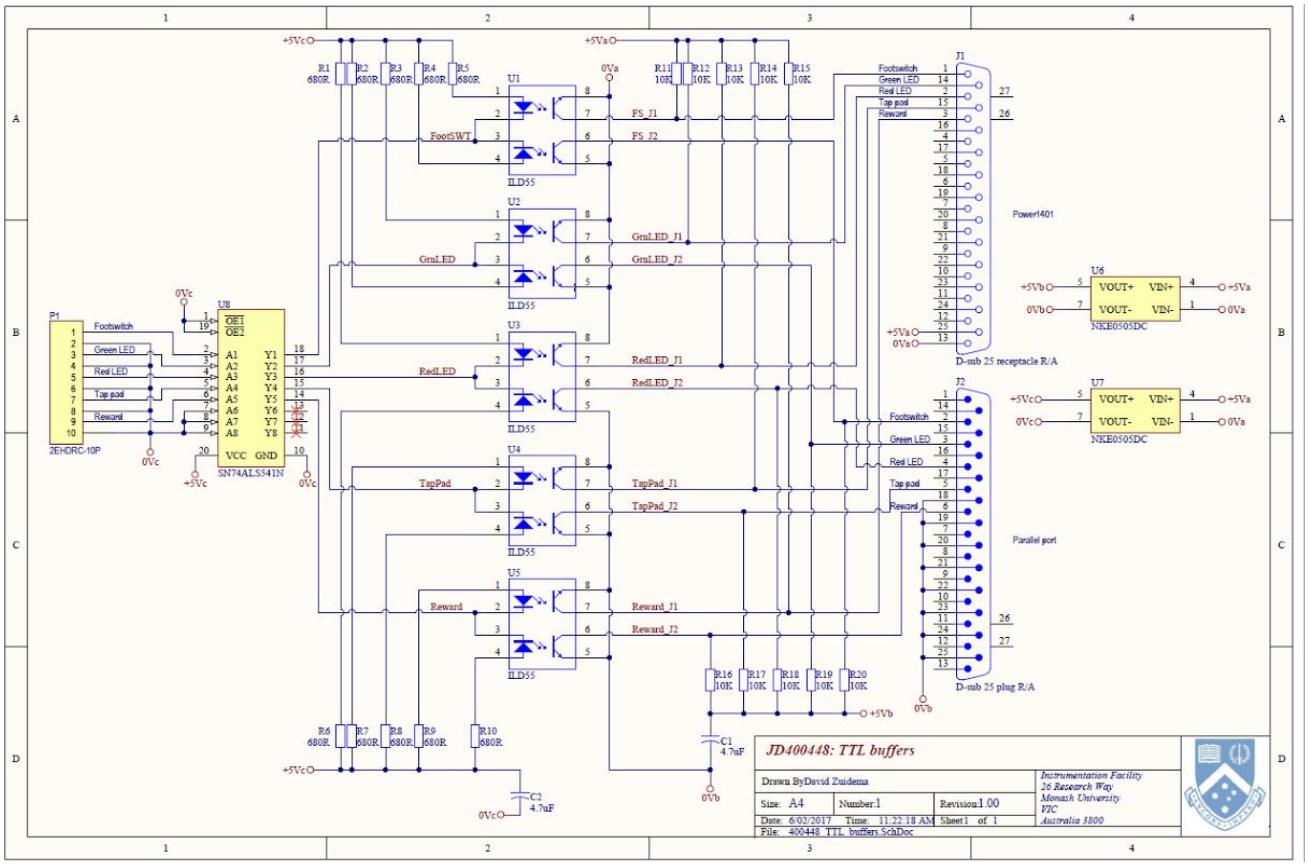


Once all the joints have been calibrated, the screen should have all green displays for the circles.



16.5 Circuit Diagrams

The whole experimental setup with the wiring to each component except for the UR5 and EyeLink connections.



The figure below shows the Power1401 digital input pins. This links to the circuit diagram above.

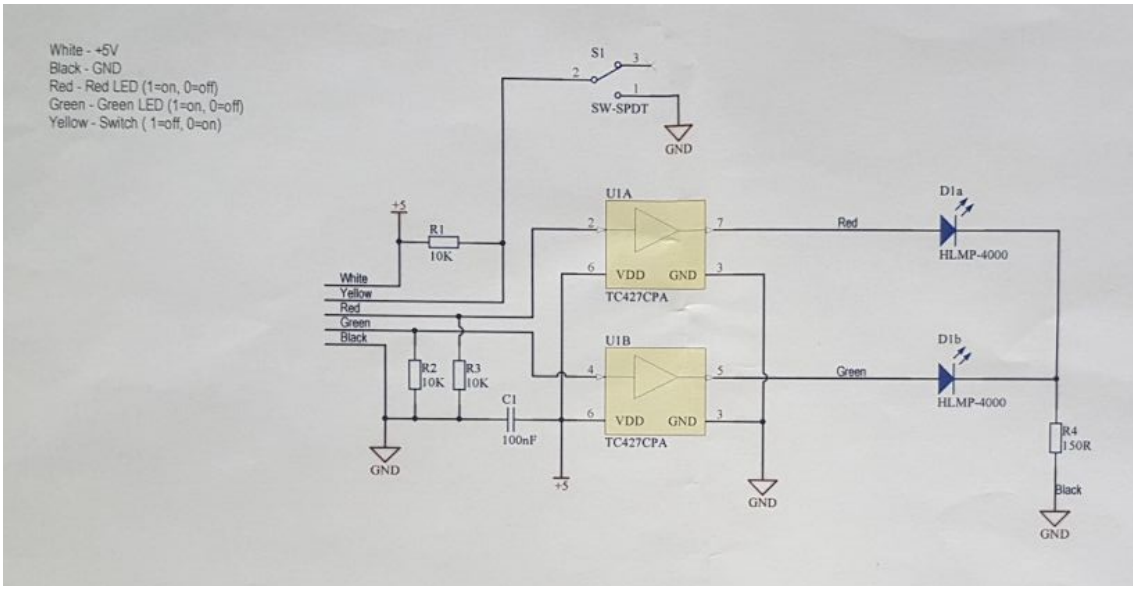
Digital I/O connectors

Digital Input plug

Digital Output socket

Pin	Output function		Pin	Input function	
	High byte out	Word out		High byte in	Word in
1	7	15	1	7	15
14	6	14	14	6	14
2	5	13	2	5	13
15	4	12	15	4	12
3	3	11	3	3	11
16	2	10	16	2	10
4	1	9	4	1	9
17	0	8	17	0	8
	Low byte out	Word out		Low byte in	Word in
5	7	7	5	7	7
18	6	6	18	6	6
6	5	5	6	5	5
19	4	4	19	4	4
7	3	3	7	3	3
20	2	2	20	2	2
8	1	1	8	1	1
21	0	0	21	0	0
9	DRH Data received 8-15	i/p	9	DTH Data transmitted 8-15	o/p
22	User i/p (buffered, reserved)		22	1-wire port	i/o
10	User o/p (buffered, reserved)		10	Not connected	
23	NDRL New data ready 0-7	o/p	23	DAL Data available 0-7	i/p
11	Output disable	i/p	11	Not connected	
24	DRL Data received 0-7	i/p	24	DTL Data transmitted 0-7	o/p
12	NDRH New data ready 8-15	o/p	12	DAH Data available 8-15	i/p
25	+5V (200mA maximum)		25	+5V (200mA maximum)	
13	Ground		13	Ground	
Shell	Mains earth to cable screen		Shell	Mains earth to cable screen	

The wiring diagram for the touch sensor, green LED, and red LED.



16.6 Micro-Control Centre Pins

The figure below depicts the pins on the micro-controller that acts as the MCC. The manual can be found *here*.

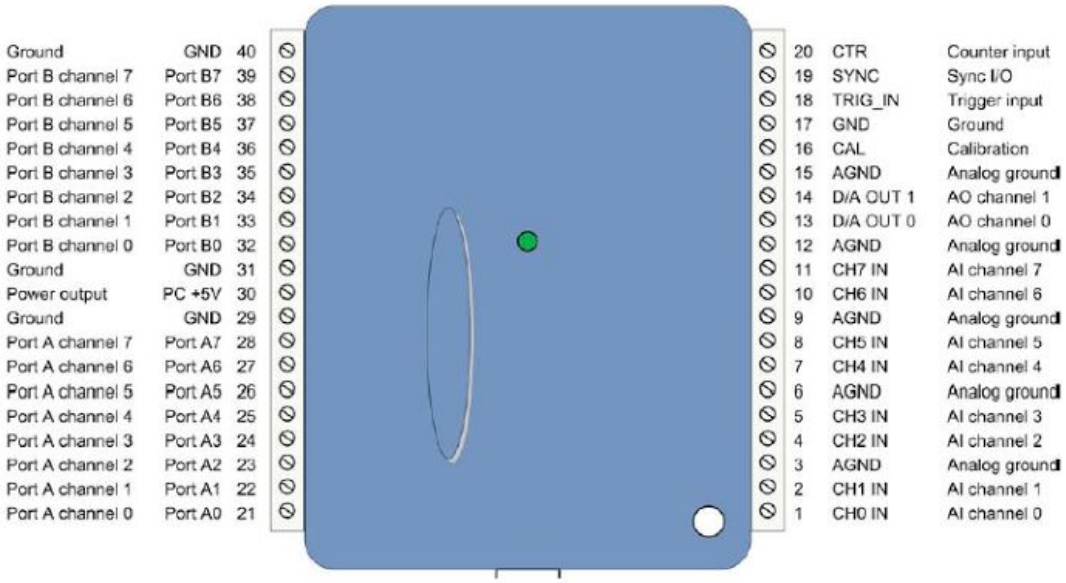


Figure 3. Single-ended mode pinout

The differential mode pinout is shown in Figure 4.

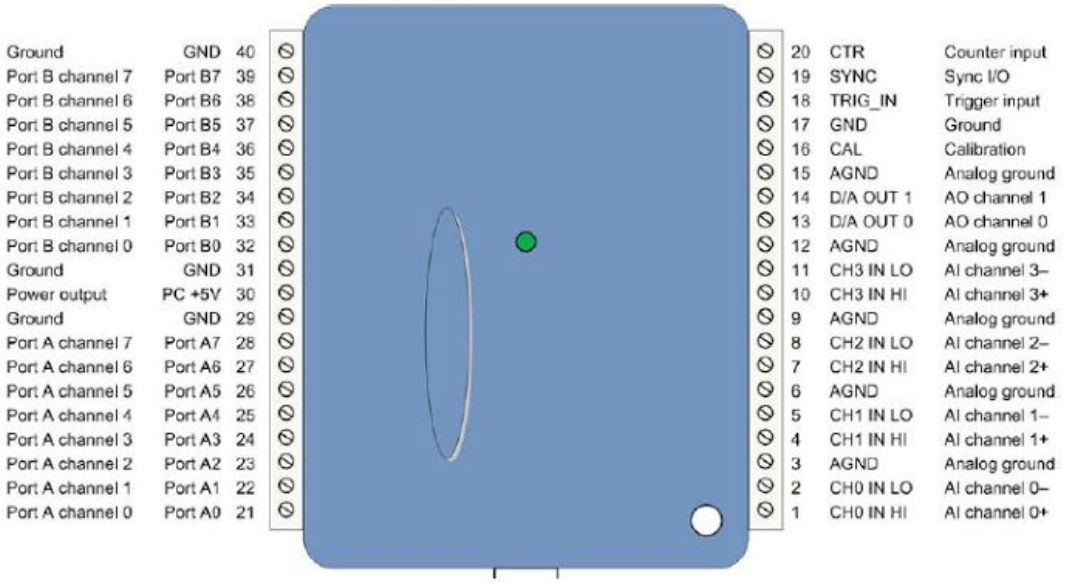
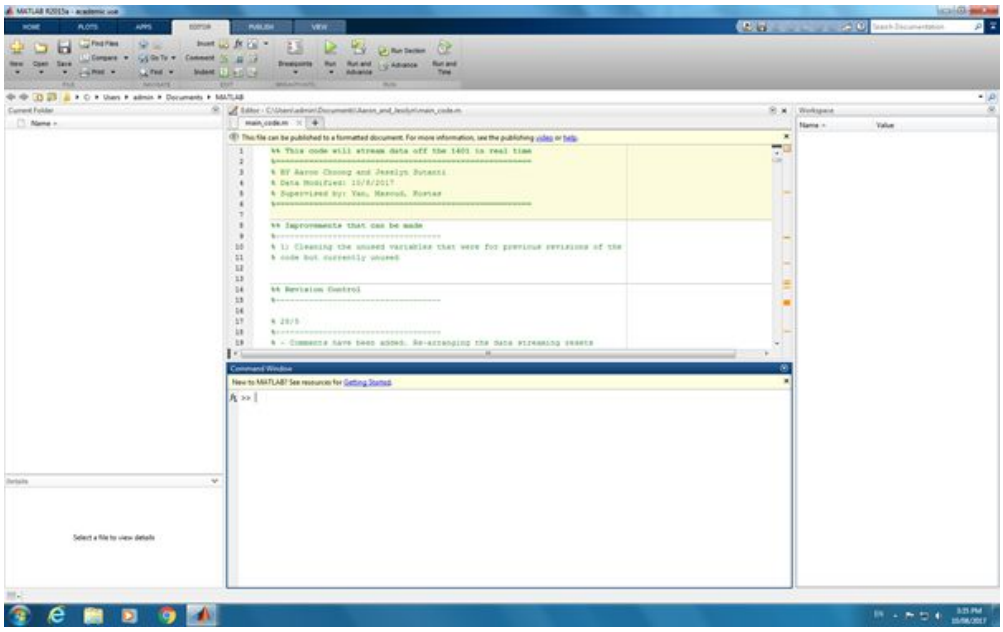


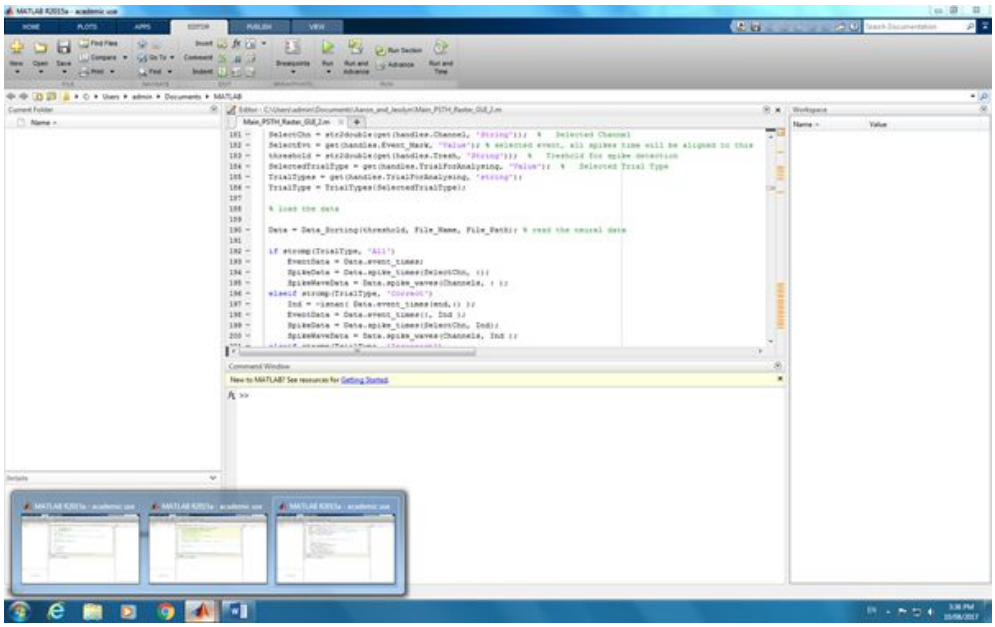
Figure 4. Differential mode pinout

16.7 Setting Applications as high-priority

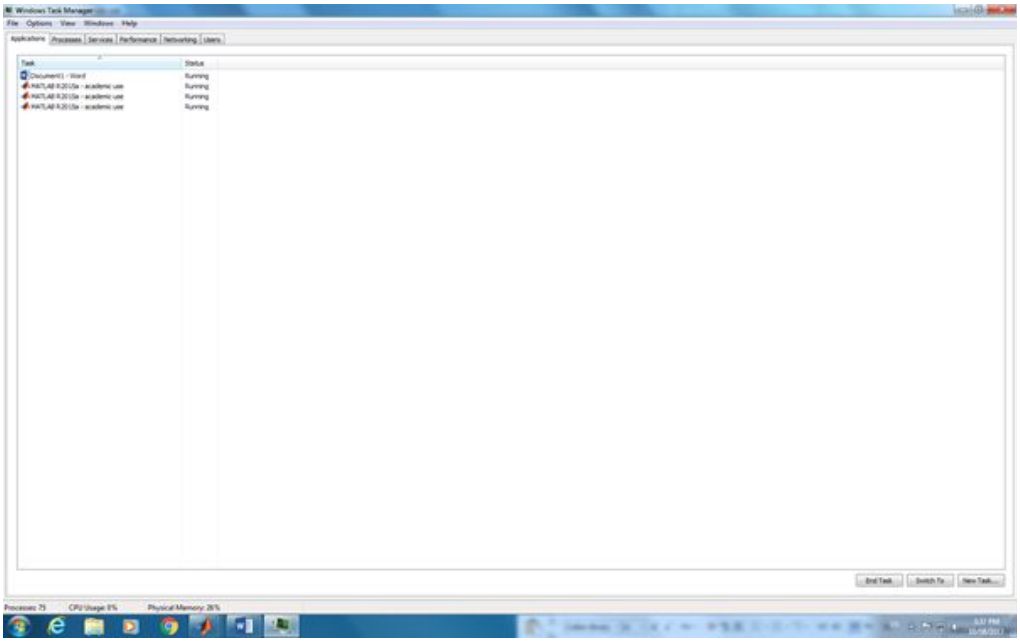
Open up all the MATLAB applications required to run the experiment before proceeding to the next step.



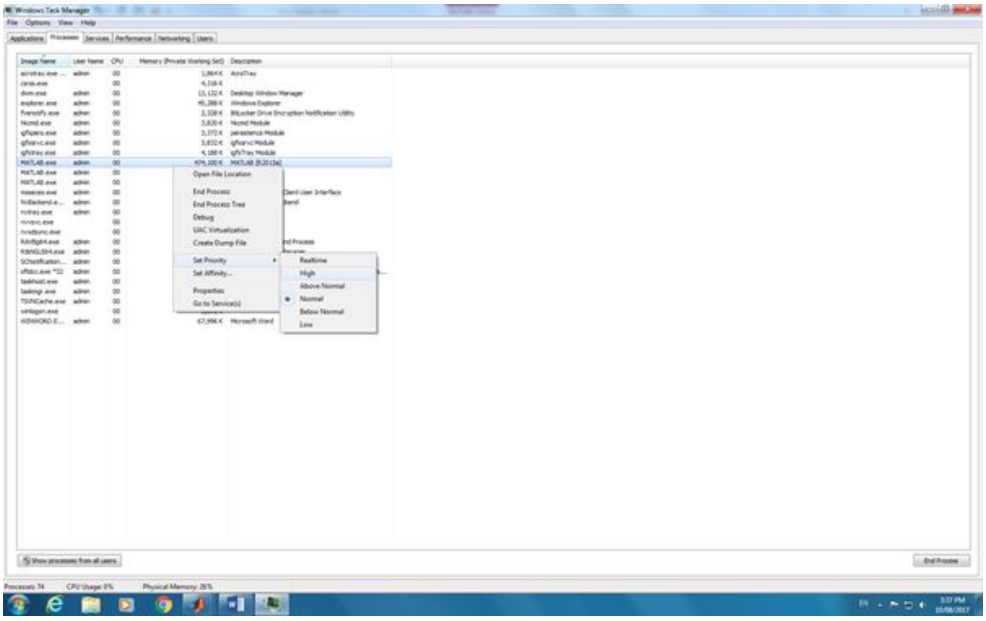
As seen in the figure below, all 3 MATLABs have been opened. For the purpose of this demonstration, the MATLABs have been opened under the one PC.



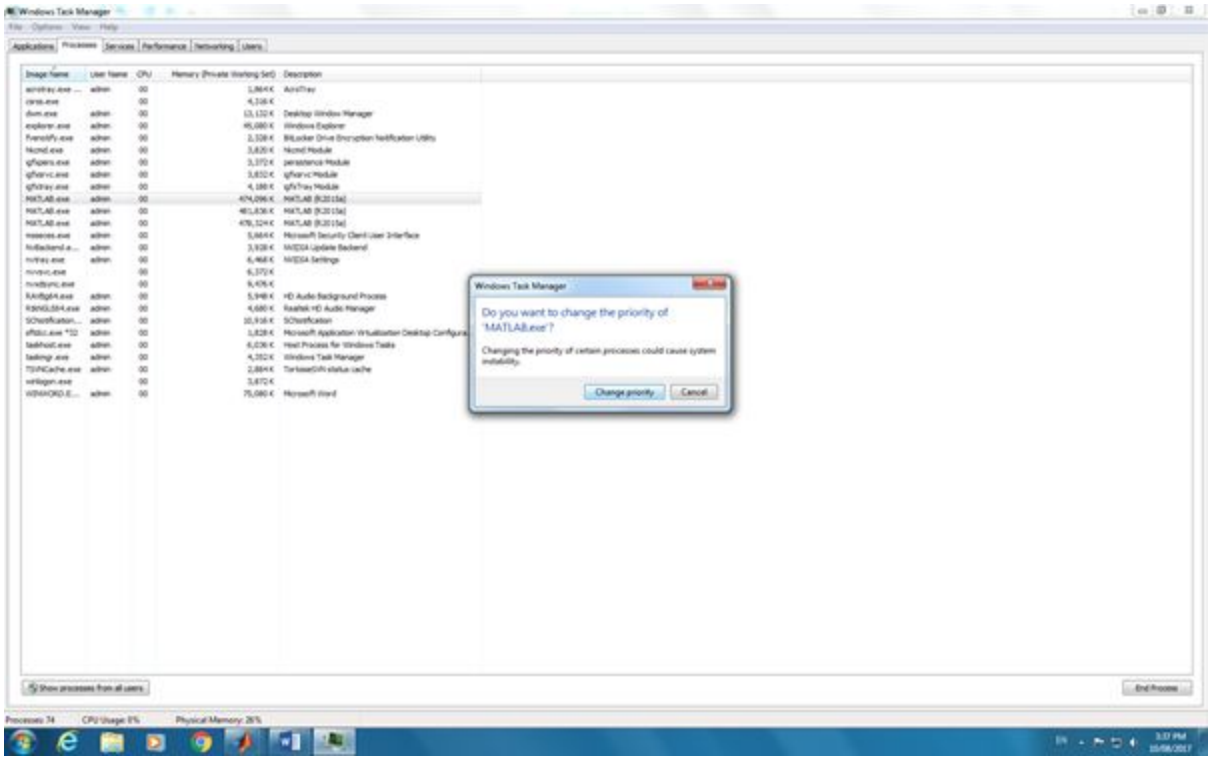
Open up Task Manager by pressing Ctrl + Alt + Delete and selecting “Task Manager”. The following window will pop up.



Open up the “Processes” tab and right-click on the MATLAB application. Select the “Set Priority” option and click “High”.



Accept the change in priority for the MATLAB application.



Set the processing priorities high for the rest of the MATLAB applications by repeating the above steps. Once that is done, close Task Manager and start the experiment.