Wearable Robotics Workshop

IEEE UNSW Student Branch 2019
Luke Wicent Sy

***Abstract*—This workshop aims to give students an introduction to wearable robotics.**

***Index Terms*—wearable robotics, exoskeleton**

CONTENTS

I. INTRODUCTION

This document accompanies the workshop held at IEEE University of New South Wales (UNSW) Sydney student branch during Term 3 2019 where one learns the different skills involved in making an exoskeleton arm.

Disclaimer: this project is not the first of its kind. In fact, this project took inspiration from Eduexo [1] and Exbow [2]. Nevertheless, as the motto of UNSW "Manu et Mente" says, we hope to encourage learning by hand and mind.

A. Why Wearable Robotics?

Wearable robotics can enable or enhance movement which can be used in performance enhancement, rehabilitation, and tele-operation applications.

(a) Military (b) Rehab (c) Teleoperation

Fig. 1. Sample Application

B. State of the art

For exoskeletons used in spinal cord injury (SCI) applications, several reports have demonstrated that these exoskeleton systems are safe. Exoskeletons require each patient to do some sort of calibration for 2 - 3 sessions of 10 - 30 minutes fitting plus at least 1 hour of safety procedures. The typical walking speed is 0.2 m/s (max of 0.7 m/s for some devices). In contrast, average walking speed is 1.4 m/s.

Some limitations are as follows:

- Designed for $<$ 100 kg patients making obese patients out of scope.
- Low metabolic cost during exoskeleton training can be bad for the patients health.
- Further limited by patient's limited range of motion, weak bone health, and prone to pressure injuries.
- Requires a well-trained caregiver
- Very prohibitive cost

C. Hardware Overview

Fig. 2 (below) shows a snapshot of the exoskeleton arm.

Table I (below) shows an overview of the bill of materials. If time and resource permits you, we encourage you to buy the corresponding parts and make your very own exoskeleton arm following this manual.

D. Why STM32 Microcontroller?

STMicroelectronics is one of the largest suppliers of microcontroller units (MCU) and their products are found in most embedded systems today. It would be beneficial for students to be exposed in MCUs used in the industry such as STM32 as opposed to using Arduino boards which are not meant to be used on commercial products but mainly used to

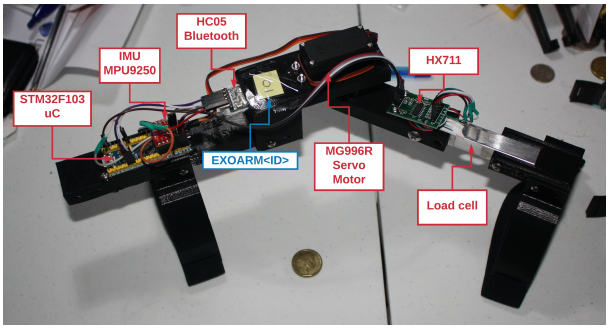


Fig. 2. Exo Arm Snapshot

TABLE I
BILL OF MATERIALS AS OF SEPT. 2019

Item	Unit	Price (AU$)
USB to Serial PL2303	1	$ 4.00
STM32F103 uC (AKA blue pill)	1	$ 4.00
HC-05 Bluetooth	1	$ 7.00
SparkFun IMU Breakout MPU-9250 - SEN-13762	1	$ 20.00
MG996R Servo Motor	1	$ 7.00
Load cell 5Kg + Hx711	1	$ 8.00
Total		$ 50.00

quickly implement or demonstrate simple projects. Moreover, the STM32F0 has superior hardware specifications compared to ATMEGA328 on Arduino boards.

STM has developed and continuously supporting its own HAL (Hardware Abstraction Library) which not only helps developers in quickly and easily setting up the microcontroller, but also provides basic libraries for the MCUs different systems such as communications (UART, SPI, I2C), memory management, USB, etc. STM HAL supports all families and every series of the STM MCUs, so it would be easy for developers to just use the library on other MCU which might be more suitable for the application. Arduino also has its own libraries, but are done in high level in which most hardware-software integration are obfuscated to the users.

See quora and udemy for further discussions on why STM32. In this workshop, we will be using an MCUs used in the industry with the Arduino library (i.e., STM32duino instead of STM HAL) as students are more likely to be familiar with the Arduino environment.

II. S01 - CAD MODEL

In this section, we will be teaching the basics of CAD model design with the intention of 3D printing the parts that will be created. As a tutorial, we will shown how to design the cuff part of the exoskeleton arm and leave the reader to designing the full exoskeleton arm.

A. Work environment setup

Register to onShape link as shown in Fig. 3 using your school (UNSW) email address and avail your free Education subscription. Similar CAD tools may also be used.

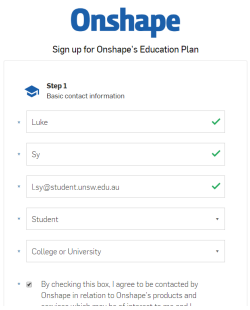
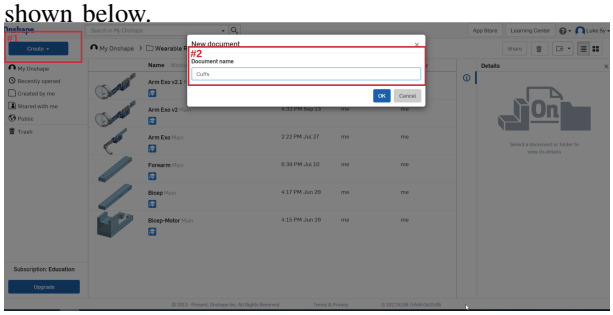


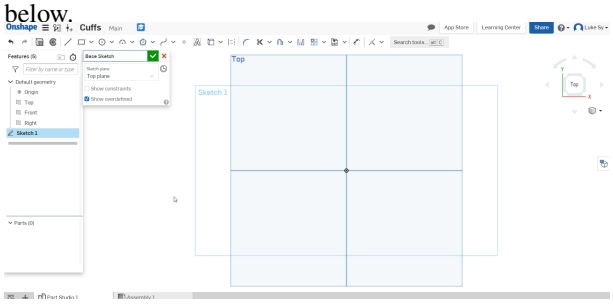
Fig. 3. Onshape Education Registration

B. Making the cuffs

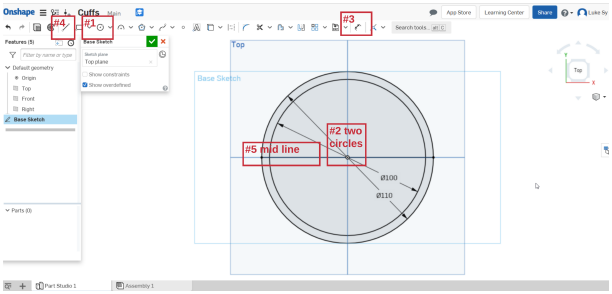
1) Create a document (#1) and set document name (#2) as shown below.



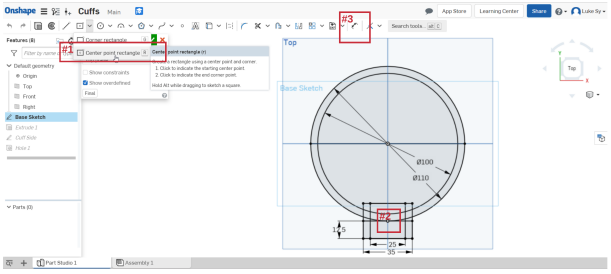
2) Create a new sketch and select the Top plane as shown below.



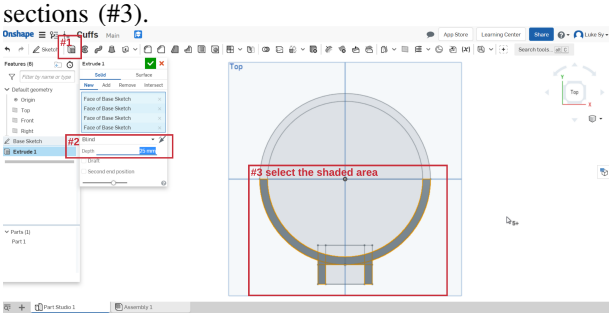
3) Draw two concentric circles let's say 100 mm and 110 mm. You can reduce these diameters depending on your wrist or arm diameter. To draw a circle click the circle icon (#1). For simplicity, set the center to the center of the top plane (#2) with any arbitrary size. The size can be adjusted using (#3). Click it and lay it on the circle created. Double click on the length and change to what you require. Lastly, add a mid-line using (#4). Clicking the line segment icon (#4) and lay it on the middle of the circle (#5).



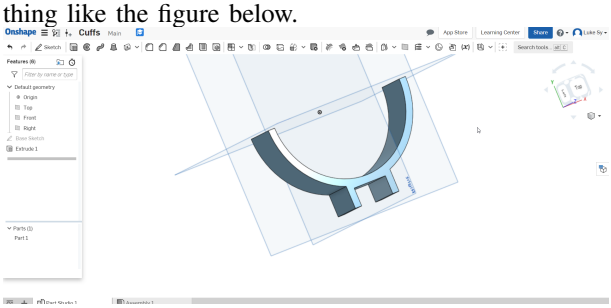
4) Next, we will make the rectangle that will connect the cuff to the exoskeleton arm. Draw a *Center point rectangle* by clicking (#1). Lay two rectangles on the lower intersect of the vertical line and the outer circle as shown in (#2). Set the width and length as shown in the figure below using (#3).



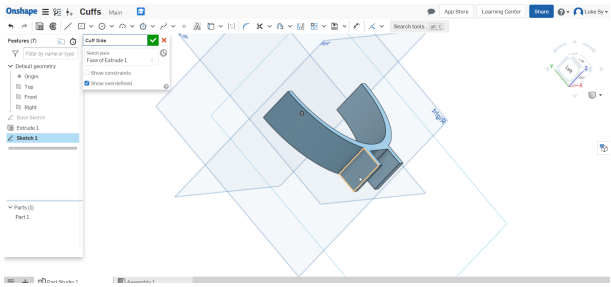
5) Extrude the relevant sections of the cuff sketch as shown in the figure below by clicking on the Extrude tool (#1). Set the depth to 25mm (#2) and select the appropriate sections (#3).



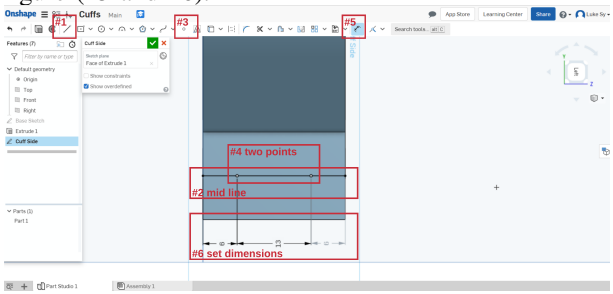
6) If viewing the model at 3D view, it should look something like the figure below.



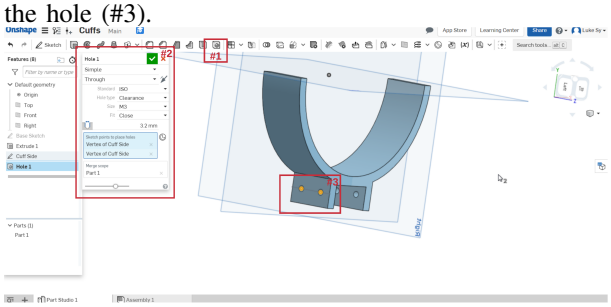
7) Next, holes will be added on the cuffs. One for the screw that will attach it to the exoskeleton arm, the other for the straps that will fasten the cuff to your arms. To begin, make a sketch on the side of the cuff.



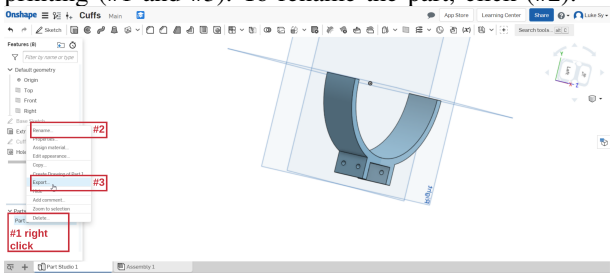
8) Add two points with the dimensions as shown in the figure below. The holes will be created from these points. First, add a line segment in the middle of the rectangle using (#1) as can seen from the side view (#2). Add two points along the line at arbitrary positions (#3 and #4). Set the distances between the points as specified in the figure (#5 and #6).



9) Add holes from the two points made in the prior step. Click the hole icon (#1), set the appropriate settings as shown in (#2), and select the corresponding points for the hole (#3).



10) Export the part in the desired format (e.g., STL) for 3D printing (#1 and #3). To rename the part, click (#2).



C. 3D Printing

If you are a UNSW student, you may want to explore the Maker space (link) and use their 3D printers. At the time of writing, they charge AU$3 per hour.

D. Full Exo Arm Model

After designing the cuffs (Sec. II-B), you should now have the basics of designing your own parts for 3D printing. **We challenge you to design your own exoskeleton arm model!** The exoskeleton arm will most likely consist of the following parts: i) biceps, ii) motor holder, iii) fore arm (two subparts), and iv) cuffs (now done). A sample exoskeleton arm CAD model is shown below (Fig. 4). The onShape project can also be viewed here.

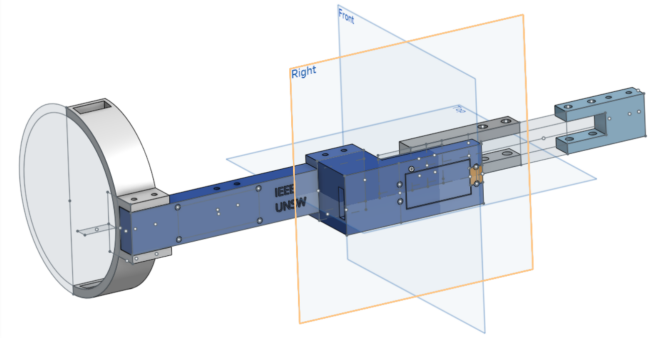


Fig. 4. Snapshot of our exoskeleton arm

III. Control

This project will be using STM32F103 boards, specifically the boards more commonly known as blue/black pill.

A. Work environment setup

1) Installing the Arduino IDE
The easiest way to start programming the microcontroller is through the Arduino IDE. First open the following website link. Then scroll down and download the appropriate installer for your operating system. Run the executable file you have just downloaded and you should be greeted with the following screen (may vary depending on your operating system).

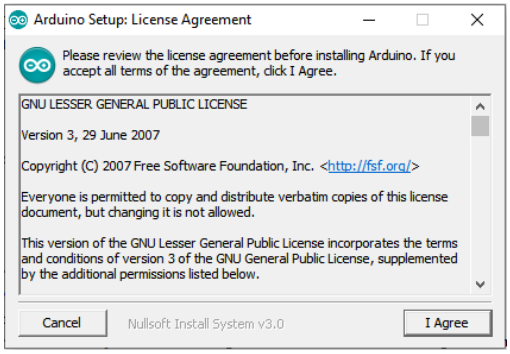


Fig. 5. Arduino IDE Installer: License Agreement

Click 'I Agree'. In the next screen (Fig. 6) ensure that 'Install USB driver' and 'Associate .ino files' are marked and click 'Next'.

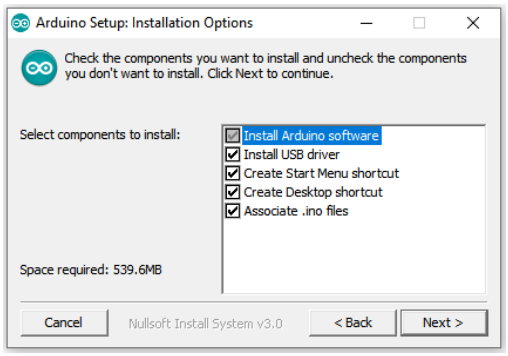


Fig. 6. Arduino IDE Installer: Installation Options

Then select the destination folder and hit 'Install'. Once completed you can close the current window. Some prompts to install some USB drivers should pop up (Fig. 6). Select 'Install' on all prompts that appear.

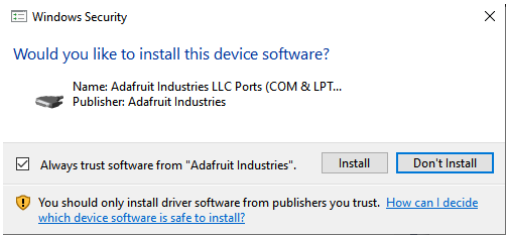


Fig. 7. USB Driver Prompt

2) Configuring the IDE [3]
Open the Arduino application and navigate to File → Preferences. Click on the button on the right of 'Additional Boards Manager URLs' (indicated in Fig. 8). Copy the following address into the textbox that appears and hit OK.
https://github.com/stm32duino/BoardManagerFiles/raw/master/STM32/package_stm_index.json

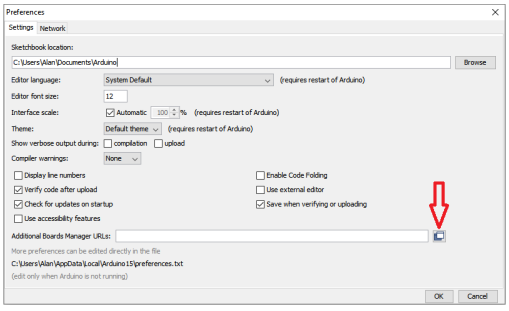


Fig. 8. Preferences Menu

Hit 'OK' again to exit the preferences menu and navigate to Tools → Board → Board Manager. Under 'Type' select 'Contributed'. Scroll down and install the latest version of 'STM32 Cores by STMicroelectronics'
This should take a while. Upon successful installation you may close the window.

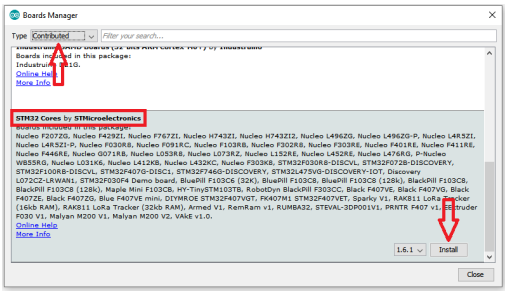


Fig. 9. Boards Manager

3) Set the IDE board settings (under Tools tab) to the ff.:

```
Board: "Generic STM32F103C series"
Optimize: "Smallest (default)"
Upload method: "STM32duino bootloader"
Variant: "STM32F103C8 (20k RAM. 64k Flash)"
CPU Speed(MHz): "72Mhz (Normal)"
```

Fig. 10. Arduino IDE board settings

4) Make sure you've selected the correct COM port (Tools → Port)
5) Not sure if additional driver installation is needed for Windows 10.
   Maybe see link.

B. Blink

1) The aim of this task is to repeatedly blink the on-board LED at a rate of on for one second and off for one second. This will allow you to determine if you have successfully set up the software and are able to communicate with the microprocessor. As indicated in the table, the LED pin for our system is attached to the digital pin 12 of the Black pill.

TABLE II
STM32 BOARD LED PINS

Board	LED pin
Black pill	PB12

2) The example public domain code developed by Scott Fitzgerald uses two functions. The first initialises the pin to act as an output and serves to set up the system when the board is first powered on or if it is reset. Setting pin 12 as an output allows us to write to it and thus control when to turn the LED on and off. PinMode takes two arguments separated by a comma. The first specifies which pin we want to set, in this case PB12. The second determines the state of the pin, in our case we wish it to be an output and hence write 'OUTPUT' to indicate this.

3) The second function provides a loop that indefinitely flashes the LED. The digitalWrite function takes two arguments, the pin and the state. We start by setting the pin high and achieve this by setting the first argument to 'PB12' and the second to 'HIGH'. This will send a high voltage level to the pin and thus turn it on.

4) Since we want the LED to be ON for 1 second, we add a delay loop after so that the micro-controller will wait 1 second before executing the next command. The delay function takes a single integer as an argument, the time in milliseconds the program has to wait until moving on to the next line of code. For a 1 second delay, the value will be 1000.
   To turn the LED OFF, we use the digitalWrite function again, but this time set the pin to be 'LOW'. This will ensure that the LED in turned off by making the voltage LOW. Again, we want to maintain the off state for 1 second and therefore we can reuse the delay instruction from before.

Make instruction for https://github.com/lsy3/exoarm/tree/master/s02-control/blink

C. Motor test

1) This next section will allow us to test that the motor is working as expected by moving the rotor through a range of motion. It is important to note that we will utilise the Servo library using the include at the top of the code. Servos have integrated gears and a shaft that can be precisely controlled. In our project we want to control the angle of the shaft between 0 to 125 degrees as specified by the define statements.
   Other variables that we have defined include the pin that controls the motor (PB1) and a delay variable set for 2 seconds (2000 milliseconds).

2) To allow us to easily control the servo, we create a servo object using 'Servo myservo;' Up to twelve servo objects can be created on most boards. We also set a integer variable to store the position of the servo and initialise this to be the minimum position of the motor, i.e 0 degrees.

3) The setup code for this task ensures that upon startup, the pins are properly configured for testing the motor. This involves setting the motor pin (PB1) to be an output and attaching the servo on pin 9 of the servo object. Then, we wish to initialise the servo object by writing a value to the servo and thus controlling the shaft accordingly. Since we want the shaft to be at 0 degrees when we start, we set this angle and the motor will move the shaft to that orientation.
   We finish the setup function using a delay loop for 2 seconds (2000 milliseconds) to allow the motor to move into the starting position of 0 degrees.

4) The loop component of this function moves the servo from its starting position of 0 degrees to the maximum position of 125 degree as specified in the variable definition at the top of the code. To implement this motion, we use for loops as we know the breaking conditions for when we need to exit the loop.

5) The first for loop causes the servo to contract as it moves from 0 degrees to 125 degrees, one degree at a time. For each iteration, we tell the servo to move into the position specified by the variable 'pos' using the myservo.write function. Then, a 15 millisecond delay is added to allow the servo to reach its new position. This loop will continue till the servo exceeds the maximum position of 125 degrees. At this point, we include a 2 second delay to ensure sufficient time to pass before instructing the servo to move in the opposite direction.

6) The second for loop causes the servo to expand and return to the starting position of 0 degrees. It is similar to the first loop but works in reverse. We initialise the variable pos to be the maximum angle and decrement this variable by 1 degree till we exceed the minimum position. The breaking condition for the loop is for the position to exceed 0 degrees at which point we add a 2 second delay to provide sufficient time before repeating the function from the first for loop.

Make instruction for https://github.com/lsy3/exoarm/tree/master/s02-control/motor

D. Force sensing

1) This section relates to using the SparkFun HX711 breakout board with a scale for determining force. Before we begin, we need to obtain the HX711 library from here. This will allow us to use existing functions to easily interface the HX711 board and the scale for sensing force.

2) Some of the key variables that we define for this piece of code includes a calibration factor that is obtained from SparkFun's HX711 Calibration sketch to match our specific load cell setup. We also set the serial data out pin as PC14 and the serial clock pin PC13. We then load the scale as a HX711 object that we can easily manipulate.

3) The setup component of this activity sets the bit rate of the Arduino to be 9600 bits per second. This is more than adequate for our project. The println command prints data to the serial port as human-readable ASCII text, in this case informing the user that the scale is about to be tested.
The next three lines of code sets up the scale to accurately measure force. We start by passing two arguments, the data out pin and the clock pin into the begin function. This will tell the scale which pin to set the weight data that it senses and the rate at which to send this information as set it by the clock. The next line sets the calibration factor as determined by the Calibration sketch for our particular set up. Lastly the tare function resets the scale to 0, assuming there is no weight on the scale during start up. This provides the relative measurement for other weights to be compared to.

4) Before moving on to the loop function that repeatedly prints the weight sensed by the scale, we print an appropriate line in ASCII indicating that we will be displaying the readings from the scale.

5) The loop function begins by printing the ASCII text for the particular reading for this iteration. It achieves this by getting the value the scale measures using the set units function which returns a floating point value. The value sensed by the scale is in pounds(lbs) but this can be changed to kilograms if you refactor the calibration factor. Before the loop repeats, we print a new line and then execute a 10 millisecond delay before taking the next measurement of the scale.

Make instruction for https://github.com/lsy3/exoarm/tree/master/s02-control/load-cell

E. Feedback control - PID

Make this bit

IV. Computer interaction

A. Install processing IDE

Refer to https://processing.org/tutorials/gettingstarted/

B. Biceps

1) Orientation estimation: SerialDebug = true Based on link basicAHRS

2) 3D viewer: Based on link SerialDebug = false cuberotatebiceps

C. Arm

1) Orientation estimation: basicAHRSandFB

2) 3D viewer: cuberotatearm

D. Game interaction - Simple pong

https://www.openprocessing.org/sketch/47481/

V. Electromyography (EMG) control

Future Work

A. Reading signal

B. Feedback control

VI. Acknowledgements

The author would like to thank UNSW Arc (funding and venue) and the IEEE NSW section for supporting our student branch in this endeavour.

And the following people who helped with this workshop and documentation:

- Han Wen and Philip Byrnes-Preston for the idea and random Qs associated in making this workshop
- Alan Ngo for Sec. III-A Control Work Environment Setup
- Martin Lunel Agbayani for Sec. I-D

References

[1] *EduExo - The Robotic Exoskeleton Kit - EduExo*. [Online]. Available: https://www.eduexo.com/ (visited on 09/07/2019).

[2] *The UMass ExBow: An OpenSource Kit to Teach Wearable Robotics*. [Online]. Available: http://www.ecs.umass.edu/exbow/ (visited on 09/07/2019).

[3] *How to program a STM32 Blue Pill with Arduino - idyl.io*. [Online]. Available: https://idyl.io/arduino/how-to/program-stm32-blue-pill-stm32f103c8t6/ (visited on 09/08/2019).