Wearable Robotics Workshop

IEEE UNSW RAS Student Branch 2020
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 1 2020 where one learns the different skills involved in making an exoskeleton arm (exoarm).

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 [3]. 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 the pin connections from the sensors to the microcontroller. Table II (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. Note that in an earlier version of this exoarm, we used an STM32 microcontroller (*i.e.,* blue pill). However, as the installation of drivers for STM32 is quite a tedious process, specially when running workshops (*e.g.,* attendees have to wait almost an hour just to install the necessary drivers), the author has opted for Arduino Nano which works pretty much out of the box with the default Arduino IDE.

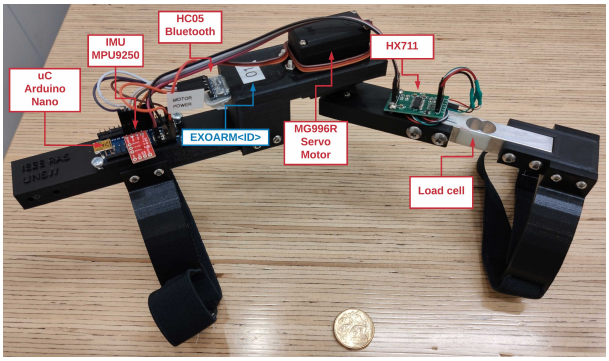


Fig. 2. Exo Arm Snapshot

TABLE I
PIN CONNECTABLE TABLE

Description	Pin
On Board LED	LED_BUILTIN
Load Cell (Force Sensor)	D_{out}: 5 SCK: 4
Motor	3
Bluetooth	RX: 6, TX: 7
IMU I2C	SDA: A4, SCL: A5 Power: A3, GND: A2

II. S01 - 3D CAD DESIGN

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 such as Autodesk and Solidworks may also be used.

TABLE II
BILL OF MATERIALS AS OF SEPT. 2019

Item	Unit	Price (AU$)
USB to Serial PL2303	1	$ 4.00
Arduino Nano	1	$ 9.00
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		$ 55.00

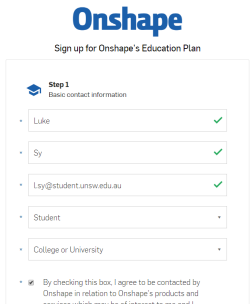
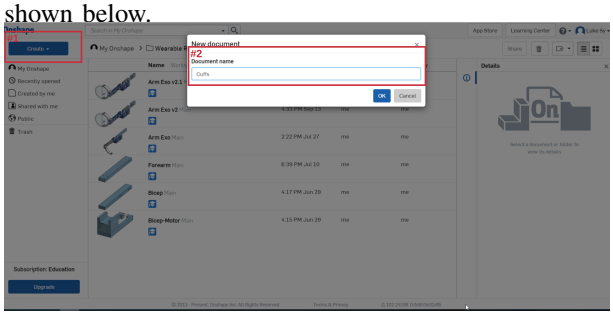


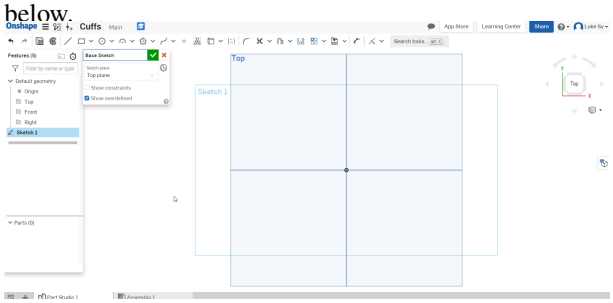
Fig. 3. Onshape Education Registration

B. Making the cuffs

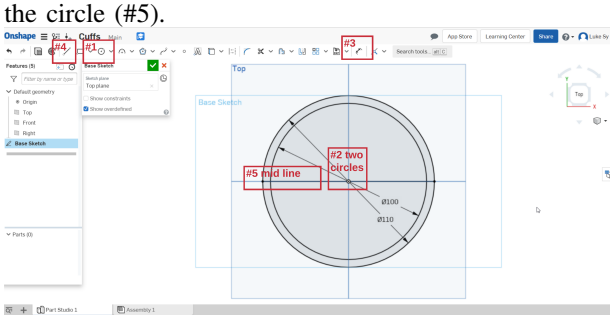
1) Create a document (#1) and set document name (#2) as shown below.



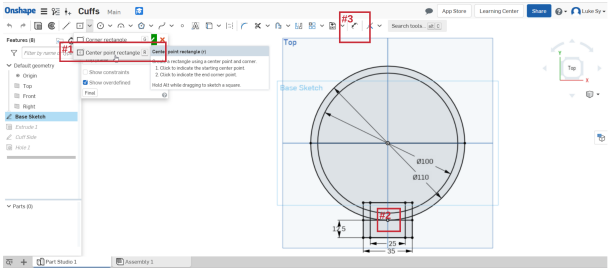
2) Create a new sketch and select the Top plane as shown below.



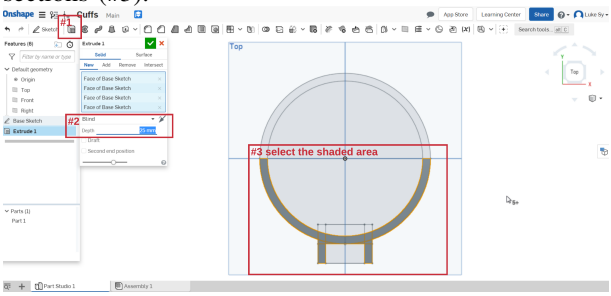
3) Draw two concentric circles let's say 100 mm and 110 mm. You can reduce these diameters depending on your wrist or arm diameter. To draw a circle click the circle icon (#1). For simplicity, set the center to the center of the top plane (#2) with any arbitrary size. The size can be adjusted using (#3). 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).



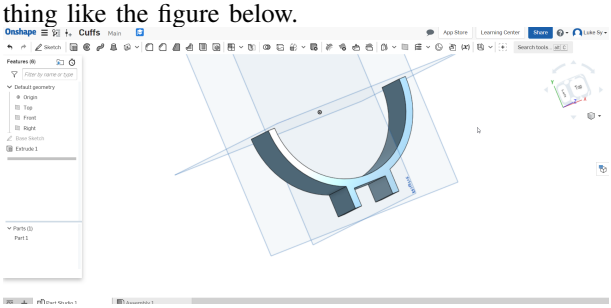
4) Next, we will make the rectangle that will connect the cuff to the exoskeleton arm. Draw a *Center point rectangle* by clicking (#1). Lay two rectangles on the lower intersect of the vertical line and the outer circle as shown in (#2). Set the width and length as shown in the figure below using (#3).



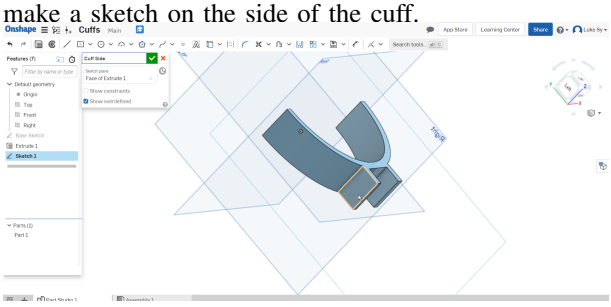
5) Extrude the relevant sections of the cuff sketch as shown in the figure below by clicking on the Extrude tool (#1). Set the depth to 25mm (#2) and select the appropriate sections (#3).



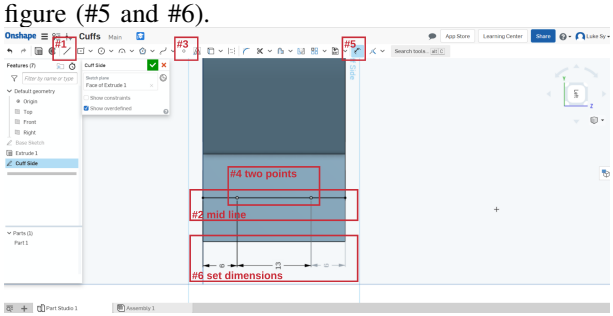
6) If viewing the model at 3D view, it should look something like the figure below.



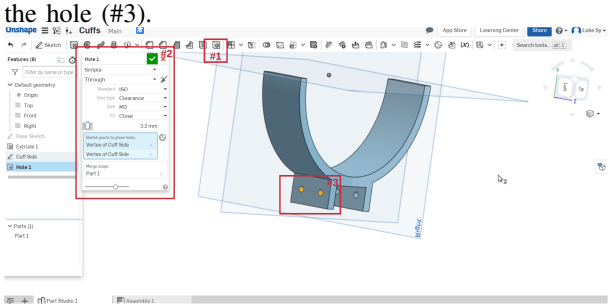
7) Next, holes will be added on the cuffs. One for the screw that will attach it to the exoskeleton arm, the other for the straps that will fasten the cuff to your arms. To begin, make a sketch on the side of the cuff.



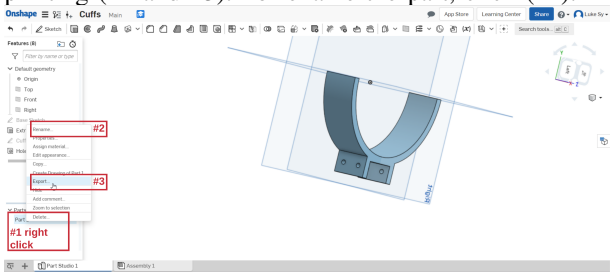
8) Add two points with the dimensions as shown in the figure below. The holes will be created from these points. First, add a line segment in the middle of the rectangle using (#1) as can seen from the side view (#2). Add two points along the line at arbitrary positions (#3 and #4). Set the distances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

III. S02 - uC Programming and Control

In this section, we will be teaching the basics of microcontroller (uC) programming and the intricacies related in controlling an exoskeleton device. Specifically in this tutorial, we will show you how to interact with each of the

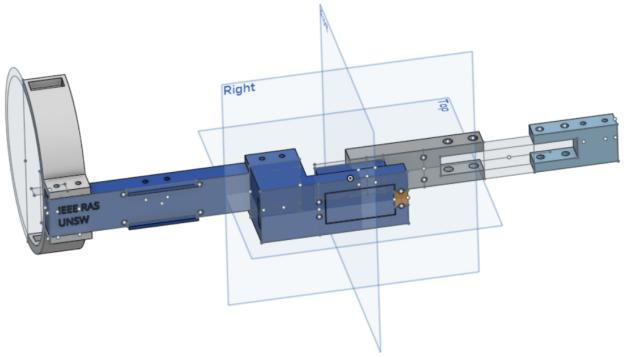
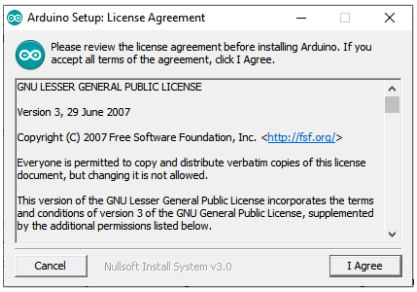


Fig. 4. Snapshot of our exoskeleton arm

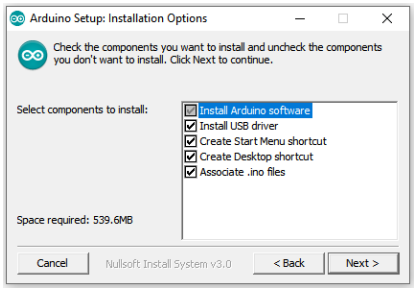
sensors and actuators embedded in the exoskeleton arm, and then finally combine all of them together to control the arm using a PID controller. This project will be using the https://store.arduino.cc/usa/arduino-nano.

A. *Work environment setup*

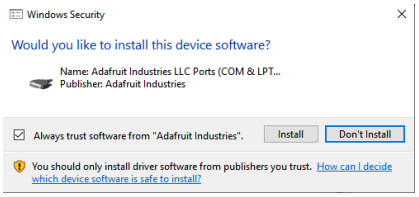
1) Installing the Arduino IDE.
   a) The easiest way to start programming the microcontroller is through the Arduino IDE. Note that the pictures below may be outdated, but the general procedure should be similar. First open the following website link. Then scroll down and download the appropriate installer 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).



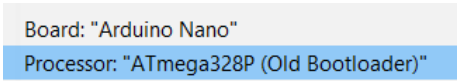
   b) Click 'I Agree'. In the next screen ensure that 'Install USB driver' and 'Associate .ino files' are marked and click 'Next'.



   c) 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. Select 'Install' on all prompts that appear.



2) Set the IDE board settings (under Tools tab) to the following.



3) Ensure to select the correct COM port (Tools → Port)

WARNING: EACH TIME BEFORE PROGRAMMING THE MICROCONTROLLER, PLEASE UNPLUG THE MOTOR POWER. IF THE MOTOR HAPPENS TO RUN (FROM SOME CODE DOWNLOADED PRIOR), IT MIGHT PULL A HIGH ENOUGH CURRENT TO DAMAGE YOUR LAPTOP. IF UNSURE, ASK THE DEMONSTRATOR BEFORE PLUGGING IN YOUR LAPTOP. My personal computer shut down a number of times due to this mishap. See Fig. 5.

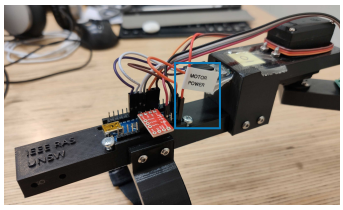


Fig. 5. WARNING: Unplug motor power when programming the microcontroller

B. *[Optional] Blink*

Nothing to fancy here but just a sanity check to make sure our microcontroller (uC) is working. Feel free to skip. See github code.

The aim of this task is to repeatedly blink the on-board LED (on for one second and off for one second). This feedback will allow you to determine if you have successfully set up the software and are able to communicate with the microprocessor.

1) Initialise the LED pin to act as an output. The function `void setup()` runs instructions at startup everytime the board is powered on or reset.

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}
```

2) Write your code on the function `void loop()` to indefinitely flash the LED. If it's your first time writing an Arduino code, see next item for detailed description.

```
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn the
    LED on
  delay(1000); // wait for a second
```

```
  digitalWrite(LED_BUILTIN, LOW);  // turn the
    LED off
  delay(1000); // wait for a second
}
```

3) `digitalWrite()` takes two arguments, the pin and the state. In the code above, we set pin PC13 to 'HIGH' or 'LOW'. Setting the output 'HIGH' will send a high voltage level (3.3V) to the pin and turn on the LED. Setting the output 'LOW' will turn off the LED.
4) `delay()` will cause the micro-controller to wait 1 second before executing the next command. For a 1 second delay, the value will be 1000.
5) Compile and upload the code to the uC.

C. Making the arm move (Motor test)

The aim of this task is to make the exoskeleton arm (motor at the elbow joint) move. See github code. The primary actuator we will be using is a servos motor. Servo motors have integrated gears and a shaft that can be precisely controlled. In this project, the servo motor MG996R (torque 9.4 kg/cm at 4.8V) can be controlled by the duty cycle of a pulse width signal (datasheet).

1) Install the *Servo library* by Michael Margolis through the Library Manager (Sketch → Include Library → Manage Libraries). This library will be used to send pulse width signals of different width to the servo motor.
2) Add the code below for initialisation, and initialise `Servo myservo` which is the (software) object that we will use to control the motor.

```
#include <Servo.h>

#define MOTOR_PIN 3 // D3
#define MOTOR_MAXPOS 100
#define MOTOR_MINPOS 0
#define DELAY 2000
Servo myservo;
```

3) Again, add the code below to `setup()` to configure `myservo` with the corresponding uC pin. Lastly, let us command the motor to go 0°.

```
void setup() {
  pinMode(MOTOR_PIN, OUTPUT);
  myservo.attach(MOTOR_PIN);
  myservo.write(0);
  delay(DELAY);
}
```

4) Compile and upload the code to the uC. After uploading, unplug the exoarm, reconnect the motor power, and then power the exoarm
5) Try moving the arm. You should feel that the motor is actually strong enough to resist normal movements.

Challenge: Modify the existing code to make it move through a range of motion, let's say from 0 to 90 degrees. Don't make the motor move past 120 degrees or you might damage the 3D printed body. We highly recommend you add a range check (see code snippet below) when commanding the servo motor. For the solution, see github motor rotate code.

```
if(pos < MOTOR_MINPOS) pos = MOTOR_MINPOS;
else if(pos > MOTOR_MAXPOS) pos = MOTOR_MAXPOS;
myservo.write(pos);
```

D. Force sensing

This section aims to introduce you on sensing modalities to understand the user's intention with regards to controlling an exoskeleton. In this tutorial, we will specifically sense from a load cell, a kind of 1D force sensor. For more details, see github code, datasheet, and similar projects. To be specific, the load cell's resistance changes depending on where force is applied. This sensor is similar to what weighing scales use. The change in resistance is typically small and additional circuitry is needed. In our case, a wheat stone bridge of resistors coupled with an amplifier is needed to translate the signal into something our uC can understand.

1) Install the *HX711 library* by Bogdan Necula, Andreas Motl (at the time or writing, v0.7.2) through the Library Manager (Sketch → Include Library → Manage Libraries). This library will be used to read the (uncalibrated) weight from the load cell.
2) Initialise `HX711 scale` with the code below which is the (software) object for communicating with the load cell. For debugging purposes, we also initialized `Serial`.

```
#include "HX711.h"
#define calibration_factor -7050.0
#define LOADCELL_DOUT_PIN 5 // D5
#define LOADCELL_SCK_PIN 4 // D4
HX711 scale;
void setup() {
  Serial.begin(9600);
  scale.begin(LOADCELL_DOUT_PIN,
    LOADCELL_SCK_PIN);
  scale.set_scale(calibration_factor);
  scale.tare(); // reset the scale to 0
}
```

3) The force is then read through `scale.get\_units()`. The value sensed by the scale can be in pounds (lbs) or kilograms depending on the calibration factor.

```
void loop() {
  Serial.print(scale.get_units(), 1);
  Serial.println(" lbs or kg(?)");
  delay(10);
}
```

4) Compile and upload the code to the uC. To test, run your favorite serial monitor (e.g., Teraterm or the Arduino IDE builtin serial monitor, Tools → Serial Monitor). See Fig. 6 for sample output.

E. Feedback control - PID

Now, it's time to bring it all together! Without a sensing and control loop, the exoarm will be stuck at a certain elbow angle incapable of knowing the user's intention. In this tutorial, we will sense from the load cell and command the servo motor to move accordingly using a proportional–integral–derivative (PID) controller. For more details on what a PID controller is, we encourage you to watch/read explanation at youtube and [4] (full systems and control book). See github code.

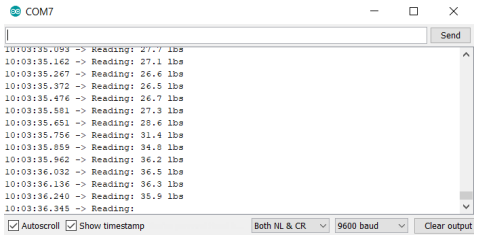


Fig. 6. Load Cell Sample Output

Note: Only the code related to the PID will defined in the tutorial below. The code for the motor and load cell was left for reader to figure one. Refer to the prior sections if needed.

1) Initialise the parameters and corresponding variables for the PID controller.

```
#define K_P 0.3
#define K_I 0.0
#define K_D 0.0
double error, new_error, int_error, diff_error;
void setup() {
  // ...
  error = 0.0;
  int_error = 0.0;
  diff_error = 0.0;
  // ...
}
```

2) The code below shows how each proportional, integral, derivative components are calculated. The sum of the error (deviation of actual) is fed back to `pos` and the servo motor.

```
void loop() {
  new_error = scale.get_units();
  int_error += error;
  diff_error = new_error - error;
  error = new_error;
  // proportional + integral + derivative
  pos += K_P*error + K_I*int_error + K_D*
    diff_error;
  if(pos < MOTOR_MINPOS) pos = MOTOR_MINPOS;
  else if(pos > MOTOR_MAXPOS) pos =
    MOTOR_MAXPOS;
  myservo.write(pos);
}
```

3) Compile and upload the code to the uC. After uploading, unplug the exoarm, reconnect the motor power, and then power the exoarm
4) Try moving the arm. You may feel that the exoarm is not reacting as fast as you hope it to be. It may even oscilate as certain times. This behavior is normal! Try tweaking the parameters for a faster reaction time and more stable control.

F. Challenge: Exoarm Teleoperation

We haven't talked about it much in this section but the exoarm also has a bluetooth attached. We challenge you to make two exoarms to talk with each other having the slave device imitate what the master device is doing. For more details, see github codes master pid, slave pid, or similar projects. See youtube video for sample demo.

1) You will need to setup the HC05 bluetooth module to connect with each other. You may want to use github code btpassthrough to configure the bluetooth module.
2) Enter bluetooth AT mode (configuration mode) by unplugging the exoarm, press and hold the bluetooth key button, and then plug the exoarm again. The bluetooth led should blink at 2 sec. interval when in AT mode.
3) Enter the following configuration for the slave device. Note that you will have to enter them one by one via serial and the bluetooth device should respond with 'OK' for each step. When changing roles, the bluetooth device might automatically reboot.

```
AT+RMAAD (To clear any paired devices)
AT+ROLE=0 (To set it as slave)
AT+ADDR (To get the address of this HC-05,
    remember to jot the address down as it will
     be used during master configuration)
AT+UART=38400,0,0 (To fix the baud rate at
    38400)
```

4) Code your own or upload this code to the slave exoarm. Note that by default, the uC module may buffer and process messages by batch (every 1 sec). To reduce this interval, use `Serial1.setTimeout(50);` at setup. Furthermore, the Arduino Nano uC only have one serial port and we'll have to implement the second serial communication using software GPIO. Add the following code:

```
#include <SoftwareSerial.h>
SoftwareSerial Serial1(6, 7); // RX (D6), TX (
    D7)
```

5) Enter the following configuration for the master device.

```
AT+RMAAD (To clear any paired devices)
AT+ROLE=1 (To set it as master)
AT+CMODE=0 (To connect the module to the
    specified Bluetooth address and this
    Bluetooth address can be specified by the
    binding command)
AT+BIND=xxxx,xx,xxxxxx (Note the commas instead
     of colons given by the slave module.
AT+UART=38400,0,0 (To fix the baud rate at
    38400)
```

6) Code your own or upload this code to the master exoarm.
7) Here's a debugging tool if things are not working as planned. github master rotate will use the master device to send messages to the slave device to rotate the arm from 0 to 125° and backwards.
8) Once done and you want to set the master device back to a slave device, use the commands listed in #3.

IV. S03 - Computer Interaction

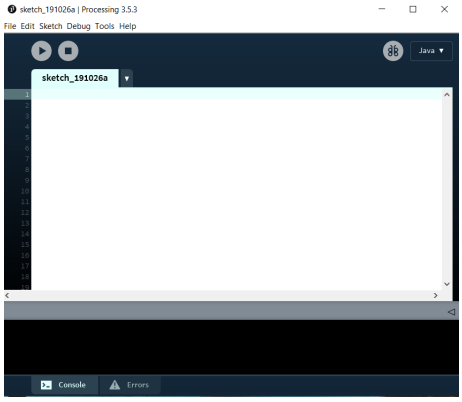
In this section, we will be teaching the basics of interfacing an exoarm and a computer using a bluetooth module. Specifically, we will model the orientation of the exoarm, and make a game out of our input interface. Note that these games can be used to gamify rehabilitation.

A. Install processing IDE

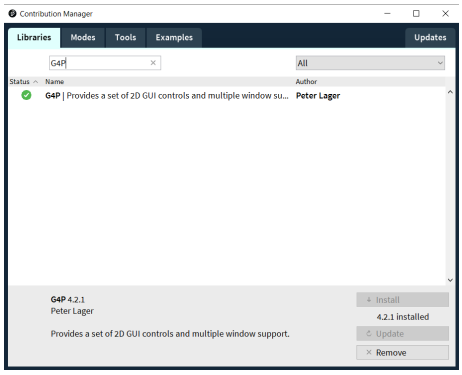
1) Download processing IDE from link. If using a Windows 10 64 bit machine, the downloaded file will be

something like `processing-3.5.3-windows64.zip`. See link for details.

2) Extract the zip file and run `processing.exe`.



3) Install G4P library via Sketch → Import Library → Add Library.



B. Orientation Estimation

This section downloads an orientation estimation algorithm, also known as Attitude and Heading Reference Systems (AHRS), into the uC. The exoarm will output the orientation state as serial data both via USB and bluetooth.

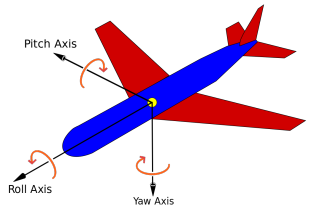
1) Load basicAHRSandFB code to the uC. Make sure `SerialDebug` is set to `true` and that the motor power is disconnected. Note that the code was based on Kris Winer's MPU9250 code.

2) Power the uC through your laptop and connect with your favorite serial monitor. You should see something like the figure below.

```
MPU9250 is online...
x-axis self test: acceleration trim within : 1.1% of factory value
y-axis self test: acceleration trim within : -0.1% of factory value
z-axis self test: acceleration trim within : 0.1% of factory value
x-axis self test: gyration trim within : -4.9% of factory value
y-axis self test: gyration trim within : -0.8% of factory value
z-axis self test: gyration trim within : 0.5% of factory value
MPU9250 initialized for active data mode....
AK8963 I AM 48 I should be 48
AK8963 initialized for active data mode....
X-Axis sensitivity adjustment value 1.23
Y-Axis sensitivity adjustment value 1.24
Z-Axis sensitivity adjustment value 1.19
ax = -94.97 ay = -97.23 az = -1096.25 mg
gx = 0.02 gy = 0.15 gz = 0.06 deg/s
mx = -355 my = 436 mz = 601 mG
q0 = 0.15 qx = -0.76 qy = 0.44 qz = 0.46
rate = 0.09 Hz
Orientation: -83.48, 55.80, 162.48
```

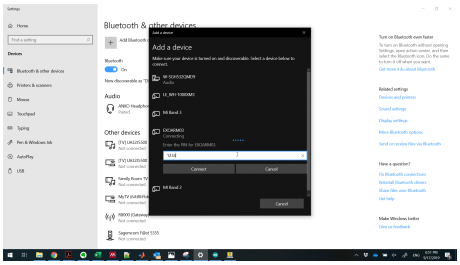
3) Specifically, the exoarm sends serial data `Orientation: yaw, pitch, roll, elbow angle`. Try moving around the device and observe how orientation changes with your movement.

4) Note that the code used Tait-Bryan angles to define the orientation of the bicep. Specifically, z-y'-x" (intrinsic rotations) also known as yaw, pitch and roll.



5) Now disconnect the device, connect the motor power and power the device through an external power source.

6) Connect your PC to the bluetooth module `EXOARM<ID>`. The default password is `1234`. ID is written on the exoarm label.



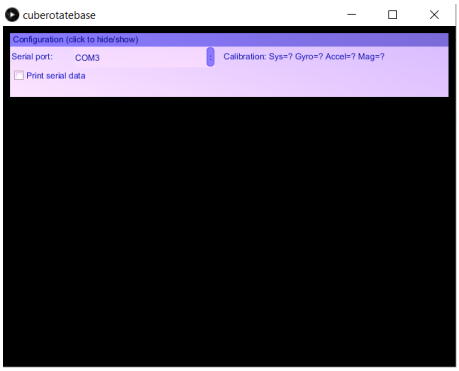
7) Set the COM port to the corresponding bluetooth port and read incoming data via your favorite serial monitor.

8) By default, the code runs `MadgwickQuaternionUpdate` [5]. Give `MahonyQuaternionUpdate` [6] a try as well (comment out the filter you're not using) and see if you can distinguish any difference.

C. Bicep 3D viewer

This section aims to receive the orientation serial data from the exoarm and model the bicep using the processing IDE.

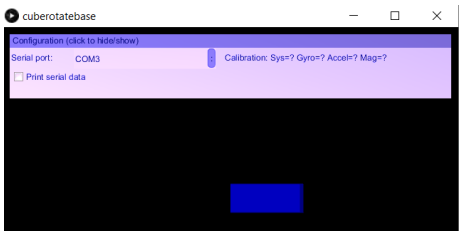
1) Load the cuberotatebase code in processing and run. You will see a blank box as shown below.



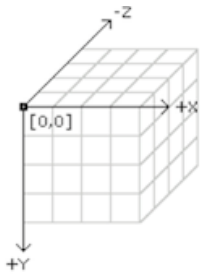
2) Select the corresponding serial port (whether cabled or via bluetooth), and check the `Print serial data` checkbox. If the serial communication is via bluetooth, the bluetooth LED on the exoarm will blink slowly (every 2 sec) if connected. You should be able to read serial data describing the exoarm bicep orientation.

3) Let us now draw a rectangle to model the exoarm biceps by adding the code below (see comment on where to add the code). If you re-run processing, it should look like the figure below.

```
// Add translation and drawing code here
translate(boxLength/2, 0, 0);
fill(0, 0, 255, 128); //blue
box(boxLength,2*boxWidth,boxWidth);
```

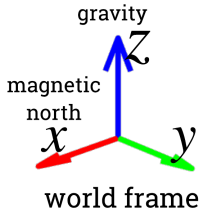


4) Now using the functions below, we will want to rotate the coordinate system of our 3D model such that it is aligned with the actual exoarm in the world frame. Do so by systematically trying different values (`yaw`, `pitch`, `roll`) as input to the `rotate` functions. The left handed coordinate system of the model (processing) is shown in the figure below.

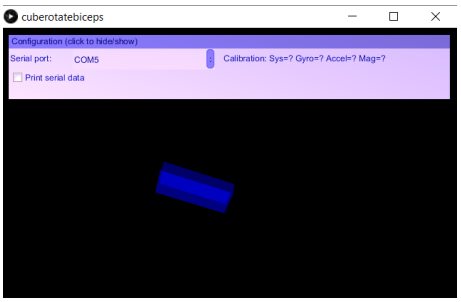



```
// Add rotation code here
rotateY(radians(?));
rotateZ(radians(?));
rotateX(radians(?)); // extrinsic rotation
```

5) In theory, the (actual) world frame is a right handed coordinate system as shown in the figure below. The conversion between left to right coordinate system involved multiplying a basis vector by -1, plus some fixed offset rotation between the sensor to exoarm frame. In this workshop, we suggested to do the trial and error approach for simplicity :p



6) If you have the correct `rotate` ordering, the orientation of the exoarm should move similarly to the exoarm model (e.g., if you rotate the bicep to the left, the box will also rotate to the left).

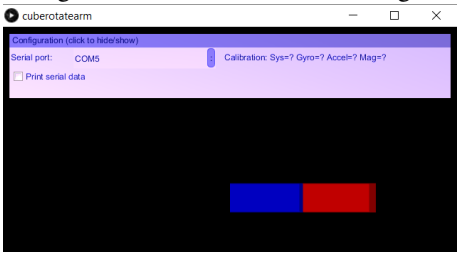


7) Sample solution can be found at cuberotatebiceps github.

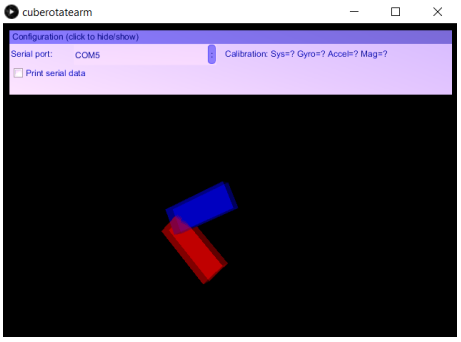
D. Arm 3D viewer

This section aims to extend the bicep 3D viewer by adding in the arm.

1) We will be extending our code from the prior section (Bicep 3D viewer). As the feature we want to achieve starts to be become more complex, let us first define some of the drawing functions below. See link for processing's 3D tutorial.
   a) `translate(x,y,z)` translates our model point with offset (x, y, z).
   b) `rotateX`, `rotateY`, `rotateZ` rotate the coordinate system along the X, Y, or Z axis.
   c) `box(x,y,z)` draws a box centered at the point where the model is currently at.
   d) `pushMatrix()` pushes the current transformation matrix onto the matrix stack. As we'll only have one object, unlikely to use it in this project.
2) Using a combination of `translate(x,y,z)`, the correct `rotate` function, and `box(x,y,z)`, draw the arm at the tip of the bicep. From better visualization, let's color the arm red using the code `fill(255, 0, 0, 128);`. You may also choose to inspect the model that it is correct before connecting with the exoarm via serial. The resulting model should look like the figure below.



3) Try different exoarm orientation specially the elbow bending and see if the model is representing it correctly.



4) Sample solution can be found at cuberotatearm github.

E. Challenge: Game interaction - Simple pong

In this section, we will be using the exoarm as a device controller for playing a simple pong game.

1) Load the simplepongbase code in processing and run. You will see an interface as shown below. Note that the bar on the right is controlled by the mouse y position.

2) Modify the code (take code snippets from Arm 3D viewer) such that the bar y position is controlled by the elbow angle. The modification will involve the following parts.
   a) UI control for selecting the serial port. Involves code at `void setup()`, and handlers such as `void handleDropListEvents(...)` and `void handleToggleControlEvents(...)`.
   b) `void setSerialPort(String portName)`
   c) `void serialEvent(Serial p)`
   d) Change `mouseY` into a normalised elbow position. `mouseY` ranges from $0 \rightarrow 480$ while elbow angle ranges from $0 \rightarrow 125$.
3) Sample solution can be found at simplepong github.

F. Challenge: Control Exoarm via GUI v1

Modify the Arm 3D viewer GUI such that a slider panel controls the the elbow angle of the exoarm. Note that for the first version, the model does not display a similar orientation as the exoarm.

1) Make a copy of your Arm 3D viewer code. We will be working on the this copy as our base.
2) Add a GSlider into the panel

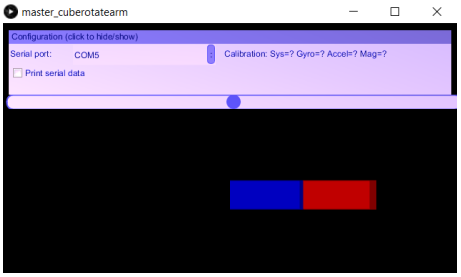
```
GSlider   sliderPanel;

void setup() {
  ...
  sliderPanel = new GSlider(this, 5, 100, 640,
    20, 20);
  ... }
```

3) For simplicity, let us delete the contents of `void serialEvent(Serial p)`.
4) Define the GSlider handler

```
public void handleSliderEvents(GValueControl
    slider, GEvent event) {
  ...
  elbow = slider.getValueF() ...
  ...
}
```

5) Upload this code to the slave exoarm.
6) Select the corresponding serial port in the GUI to connect with the exoarm. Changing the slider panel should remotely change the elbow angle of the exoarm.

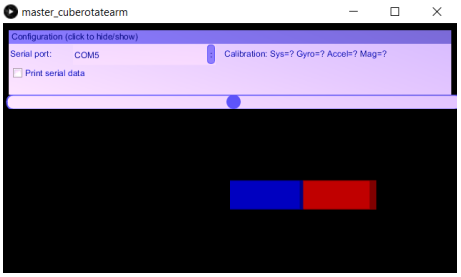


7) Sample solution can be found at master_cuberotatearm github.

G. Challenge: Control Exoarm via GUI v2

Modify the Arm 3D viewer GUI such that a slider panel controls the the elbow angle of the exoarm. Note that in this second version, we want the model to display a similar orientation as the exoarm.

1) Make a copy of your Control Exoarm via GUI v1 code. We will be working on the this copy as our base.
2) Insert back the deleted `void serialEvent(Serial p)` (refer to Arm 3D viewer).
3) Modify the basicAHRSandFB code such that the exoarm sends bicep orientation state but receives elbow state. Refer to slave pid code.
4) Check if the serial read and write events are interfering with one another. If they do, fix accordingly.
5) To test, select the corresponding serial port in the GUI to connect with the exoarm. Changing the slider panel should remotely change the elbow angle of the exoarm.



6) I still don't have a sample solution so if you successfully done this part. Please send me the solution `l.sy@unsw.edu.au`.

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, Philip Byrnes-Preston, and Ben Xia for the idea and random questions associated in making this workshop
- Alan Ngo for Sec. III-A.
- Logan Peters for initial versions of Sec. III-B, III-C, III-D.
- Martin Lunel Agbayani for the why STM32 (existed in older version).

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] A. S. Gorgey, "Robotic exoskeletons: The current pros and cons.," *World J. Orthop.*, vol. 9, no. 9, pp. 112–119, 2018. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/30254967%20http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC6153133.

[4] N. S. Nise, *CONTROL SYSTEMS ENGINEERING, (With CD)*. John Wiley & Sons, 2007.

[5] S. O. H. Madgwick, A. J. L. Harrison, and R. Vaidyanathan, "Estimation of IMU and MARG orientation using a gradient descent algorithm," in *IEEE Int. Conf. Rehabil. Robot.*, IEEE, 2011, pp. 1–7. [Online]. Available: http://ieeexplore.ieee.org/document/5975346/.

[6] R. Mahony, T. Hamel, P. Morin, and E. Malis, "Nonlinear complementary filters on the special linear group," *Int. J. Control*, vol. 85, no. 10, pp. 1557–1573, 2012.