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 (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.

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

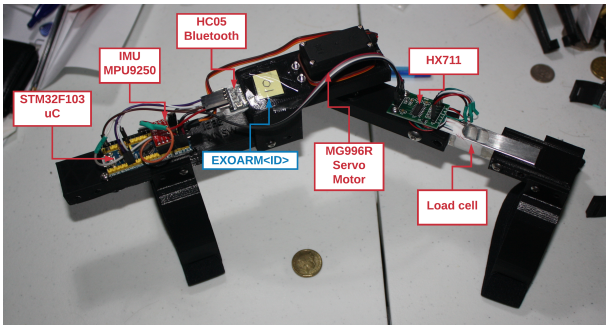


Fig. 2. Exo Arm Snapshot

TABLE I
PIN CONNECTABLE TABLE

Description	Pin
On Board LED	PB12
Load Cell (Force Sensor)	D_{out}: PC14 SCK: PC13
Motor	PB1
Bluetooth	PA9, PA10
IMU I2C	SDA: PB7, SCL: PB6 Power: PB8, GND: PB9

meant to be used on commercial products but mainly used to 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.

TABLE II
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

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 may also be used.

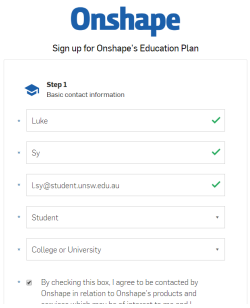
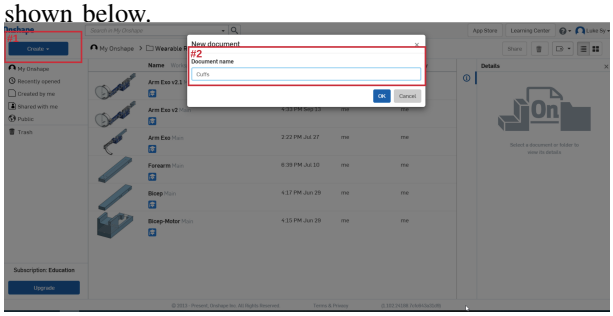


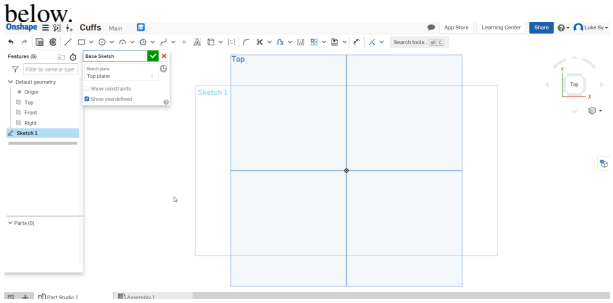
Fig. 3. Onshape Education Registration

B. Making the cuffs

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

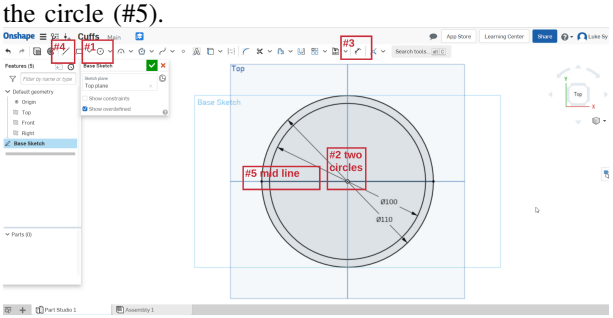


2) Create a new sketch and select the Top plane as shown below.

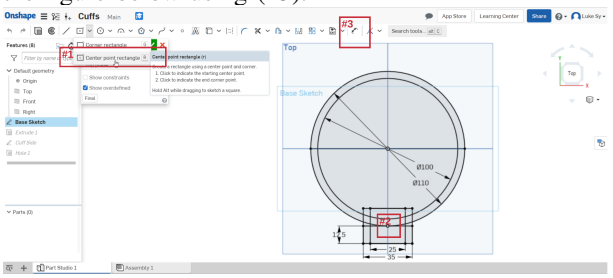


3) Draw two concentric circles let's say 100 mm and 110 mm. You can reduce these diameters depending on your wrist or arm diameter. To draw a circle click the circle

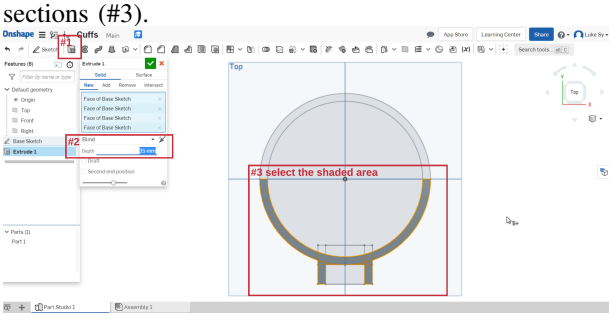
icon (#1). For simplicity, set the center to the center of the top plane (#2) with any arbitrary size. The size can be adjusted using (#3). Click it and lay it on the circle created. Double click on the length and change to what you require. Lastly, add a mid-line using (#4). Clicking the line segment icon (#4) and lay it on the middle of the circle (#5).



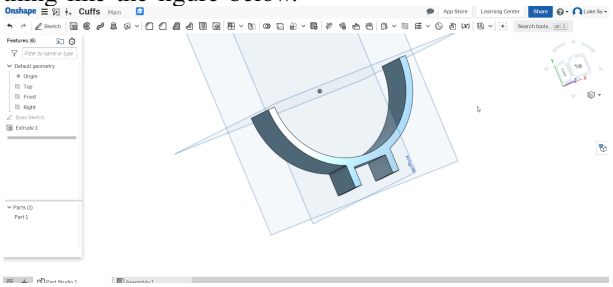
4) Next, we will make the rectangle that will connect the cuff to the exoskeleton arm. Draw a *Center point rectangle* by clicking (#1). Lay two rectangles on the lower intersect of the vertical line and the outer circle as shown in (#2). Set the width and length as shown in the figure below using (#3).



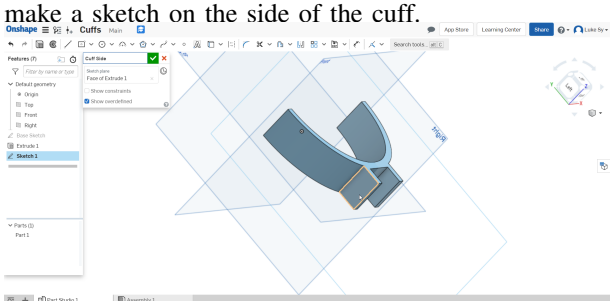
5) Extrude the relevant sections of the cuff sketch as shown in the figure below by clicking on the Extrude tool (#1). Set the depth to 25mm (#2) and select the appropriate sections (#3).



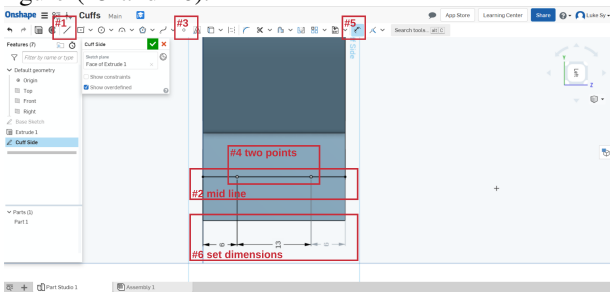
6) If viewing the model at 3D view, it should look something like the figure below.



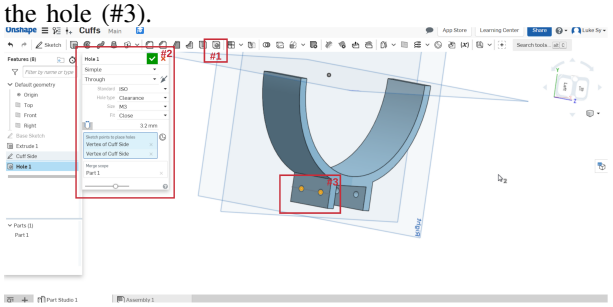
7) Next, holes will be added on the cuffs. One for the screw that will attach it to the exoskeleton arm, the other for the straps that will fasten the cuff to your arms. To begin, make a sketch on the side of the cuff.



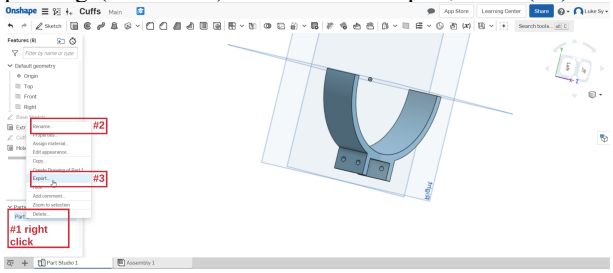
8) Add two points with the dimensions as shown in the figure below. The holes will be created from these points. First, add a line segment in the middle of the rectangle using (#1) as can seen from the side view (#2). Add two points along the line at arbitrary positions (#3 and #4). Set the distances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

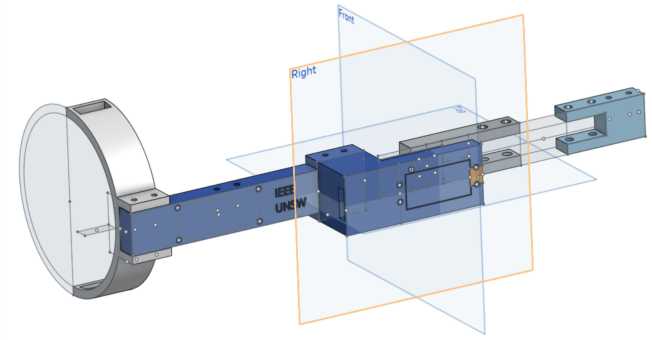


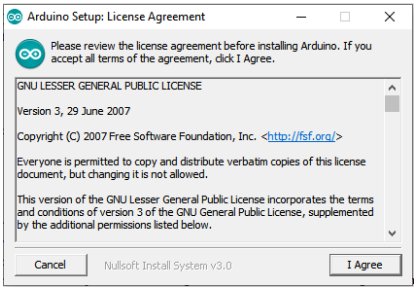
Fig. 4. Snapshot of our exoskeleton arm

III. S02 - uC Programming and Control

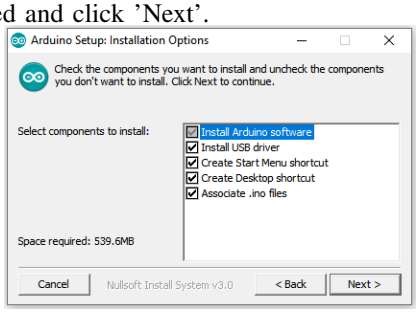
In this section, we will be teaching the basics of microcontroller (uC) programming and the intricacies related in controlling an exoskeleton device. Specifically in this tutorial, we will show you how to interact with each of the sensors and actuators embedded in the exoskeleton arm, and then finally combine all of them together to control the arm using a PID controller. This project will be using a STM32F103 board, specifically the board more commonly known as blue pill.

A. Work environment setup

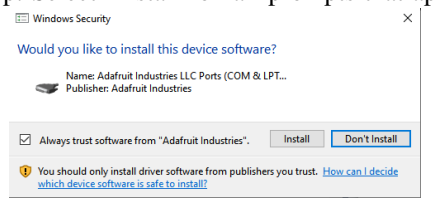
1) Installing the Arduino IDE.
   a) The easiest way to start programming the microcontroller is through the Arduino IDE. First open the following website link. Then scroll down and download the appropriate installer for your operating system. Run the executable file you have just downloaded and you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



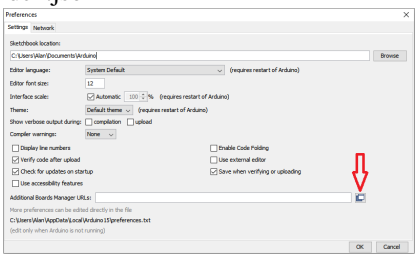
   b) Click 'I Agree'. In the next screen ensure that 'Install USB driver' and 'Associate .ino files' are marked and click 'Next'.



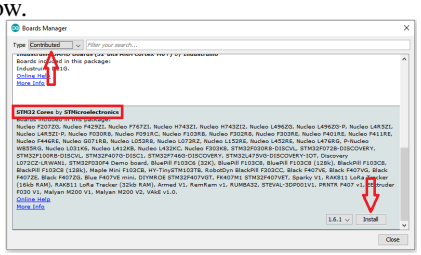
   c) Then select the destination folder and hit 'Install'. Once completed you can close the current window. Some prompts to install some USB drivers should pop up. Select 'Install' on all prompts that appear.



2) Configuring the IDE [4]
   a) Open the Arduino application and navigate to File → Preferences. Click on the button on the right of 'Additional Boards Manager URLs' (indicated in Fig. below). Copy the following address into the textbox that appears and hit OK.
   https://github.com/stm32duino/BoardManagerFiles/raw/master/STM32/package_stm_index.json



   b) 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.



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

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

4) Ensure to select the correct COM port (Tools → Port)
5) **Not sure if additional driver installation is needed for Windows 10.** If you encountered issues, see link.

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 (e.g., PB12) to act as an output. The function `void setup()` runs instructions when to set up the system when the board is first powered on or if it is reset.

```
void setup() {
  pinMode(PB12, 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(PB12, HIGH); // turn the LED on
  delay(1000);              // wait for a
    second
  digitalWrite(PB12, 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 PB12 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 STM32.

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).

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 computer shut down a number of times due to over current pull of the device. Remove the IMU (red board) and see Fig. 5.

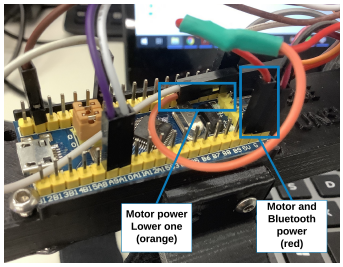


Fig. 5. WARNING: Unplug motor power when programming STM32

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 PB1
#define MOTOR_MAXPOS 125
#define MOTOR_MINPOS 0
#define DELAY 2000
Servo myservo;
```

3) Again, add the code below to `setup()` to configure `myservo` with the corresponding STM32 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 STM32. 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 microcontroller 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  PC14
#define LOADCELL_SCK_PIN  PC13
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 STM32. 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.

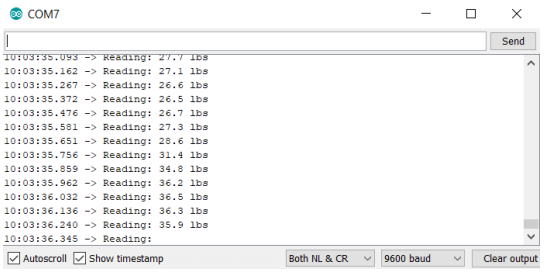


Fig. 6. Load Cell Sample Output

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 proportionalintegralderivative (PID) controller. For more details on what a PID controller is, we encourage you to watch/read https://youtu.be/UR0hOmjaHp0 and [5] (full systems and control book). See github code.

Note: Only the code related to the PID will defined in the tutorial below. We leave it to you to figure out the initialisation of the motor and load cell. Refer to the sections above if need to.

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 STM32. 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 by unplugging the exoarm, press and hold the bluetooth key button, and then plug the exoarm again.
3) Enter the following configuration for the slave device.

```
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 master pid to the slave exoarm.
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 master pid to the master exoarm.
7) Debugging tool 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.

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, 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 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] 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] *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).

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