POSTURAL CORRECTION THROUGH SENSOR DRIVEN FEEDBACK

Faraz Sadrzadeh-Afsharazar

Mohammadali Ahmadipour

Seyed-Youns Sadat-Nejad

April , 2017

Table of Contents

1.0 Abstract

In many sport activities and rehabilitation procedures, having correct posture is crucial to have beneficial and safe performance. Providing awareness to the performer is often done by a coach or a personal assistance. However, having third-party feedback from another person is not possible for everyone, in every moment. Hence, this project has focused on developing a device, in the form of a watch or wristband, which can measure and analyze the dynamic posture of its user and provide critical information to his/her postural quality, during the activity. The sporting activity that the group focused on is the bicep curl. This product can improve postural self-awareness and assist the user to correct their posture, by providing visual feedback. This feedback is provided by analyzing the data from an Inertial Measurement Unit (IMU). By observing the linear and rotational motion at the user's, his or her bicep curls can be analyzed and correction tips can be provided through a Graphical User Interface (GUI), on an integrated TFT screen. Thus, the feedback will be a fully visual (graphical and text based), with no audio. This device is designed to be compact, fully wireless, and battery powered. The wearable uses Wi-Fi to communicate with a nearby personal computer (PC), where the data is processed. The implemented device has high accuracy, low power consumption and is economically affordable. The product is very simple in design and its cost is approximately $150.00 CND.

Key words: Wearable Technology, Bio-Metric Devices, IMU, Posture Correction

2.0 Background

Between 8-10% of the annual referrals to the hospitals are due to sport injuries [9 ,6]. In addition, 30% of those injuries involve younger athletes between the ages 7-17 years [5]. These injuries have the potential to result in bodily growth deficiencies [5, 8]. Some reasons for these injuries include incorrect workout techniques, short recovery periods, incorrect rehabilitation procedures, and lifting heavy loads [5]. Hence, acquiring a correct technique and posture can result in the reduction of sport injuries and increase the quality of exercise. To achieve this goal, in the past few decades, bio-metric wearable technologies have played crucial roles both in professional and recreational sport activities.

A biometric system is a pattern recognition system that works by the means of acquisition of physiological inputs, from a user. The process involves extracting the signal of interest, and setting specific thresholds on certain aspects of the signal [7]. This threshold can be an amplitude triggered output of any modified version of the collected physiological signal. In sport activities, each move should be monitored, so that each is correctly executed, within the acceptable ranges. The muscles, joints, and ligaments of the body might get injured when too much mechanical stress in applied onto them. The defining threshold that distinguishes a safe and an unhealthy exercise is called the fatigue point. It is vital to exercise within these allowable static and dynamic ranges of motion. If not executed properly, not only the exercise would be inefficient, but also the probability of a sports injury would significantly increase [10].

2.1 Goals and objectives

With the progression of technologies that monitor human movements with accelerometers and gyroscopes, there has been several systems designed for assessing a person's posture and movement in training and rehabilitation activities. These activities may include learning how to perform a correct bicep curl. These devices are meant to present a performance summary to the user, indicating the accuracy and correctness of his/her movement.

This project has developed a wearable bio-metric device that notifies the user about incorrect movements and suggests corrections to avoid sports injuries, with the aid of biometrics and physiological signals. The implemented device will monitor and record the body's movements, during bicep curls. This system will be implemented in a watch-like contraption and it will be programmed to provide a visual feedback. The feedback will be provided through an LCD screen, mounted on the wearable. The device will be wirelessly communicating with a nearby computer, through Wi-Fi, to increase its computational power. The cross-platform commination will happen according to Figure 1.

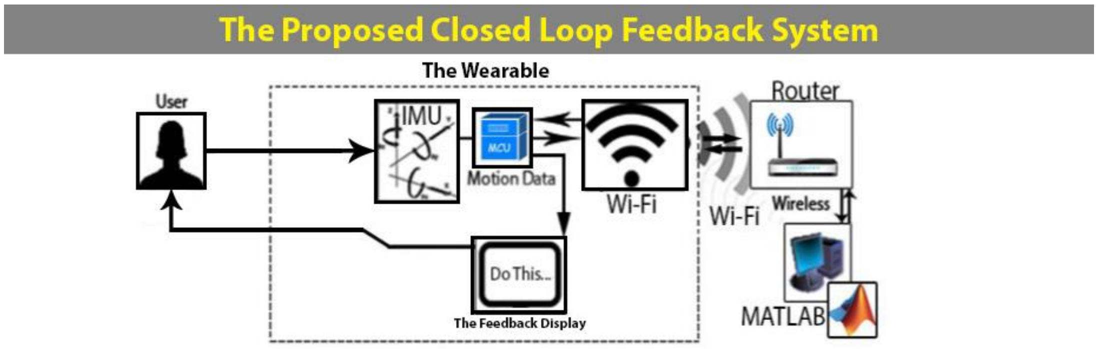


Figure 1. Block Diagram of the Components in the Working System

As seen in Figure 1, the device (wearable) collects the motion data from the user's wrist, where the wearable is mounted. It is evident that the device looks quite like a closed loop control system. Figure 2 represents the wearable-user interaction, using a closed loop system diagram.

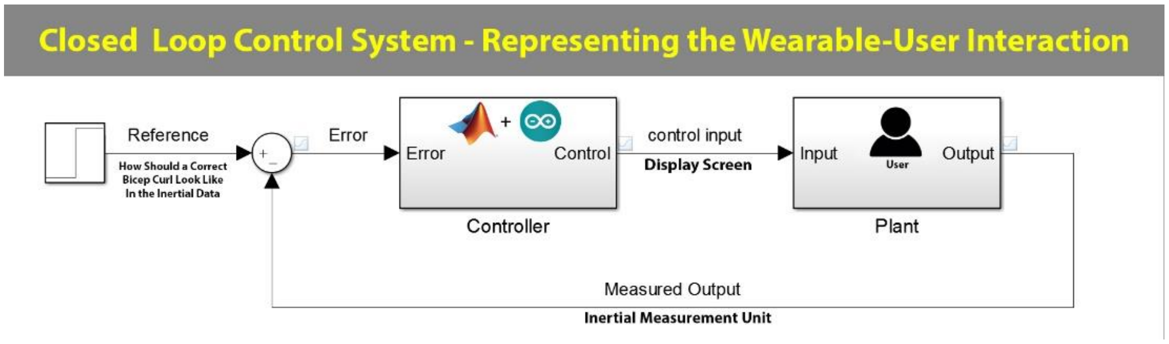

Closed Loop Control System - Representing the Wearable-User Interaction
Reference
How Should a Correct Bicep Curl Look Like In the Inertial Data
+
–
Error
Error
+
Control
Controller
control input
Display Screen
Input
User
Output
Plant
Measured Output
Inertial Measurement Unit

Figure 2. Closed Loop System Representation of the Wearable's Functionality

The motion data is collected using an IMU. An IMU is capable of measuring acceleration and angular velocity, in all three dimensions. The acceleration and angular velocity data can be processed, to obtain linear acceleration and absolute orientation. The raw acceleration data normally contains the Earth's gravitational field vector. This vector can affect the data processing procedure, making it more difficult to visualize the performed action, thus the vector is required to be subtracted away from the accelerometer data. In the IMU, the acceleration is measured by a unit called the accelerometer and the angular velocity is measured by another unit called the gyroscope. These data are then taken in by a Micro-Controller Unit (MCU). The MCU acts as the brain of the wearable device and oversees the streaming data and executes actions. As mentioned, the prototyped device is designed to be wireless, thus it includes a Wi-Fi module, on board. The MCU is supposed to be simultaneously communicating with the IMU, the Wi-Fi module, and the feedback display. After collecting the IMU data, the MCU will transmit the data to a local router. The data is then relayed and transmitted to a Windows PC, running the MATLAB software, via Wi-Fi. MATLAB can perform complex mathematical operations, in real-time, giving the designers robust tools to analyze the incoming data. Thereafter, the MATLAB software decides on whether the exercise (bicep curl) performance was satisfactory or not. This decision is then encoded in a wireless signal and is sent back to the wearable, which is directly delivered to the MCU. The MCU is already aware of all the possible decision outcomes of the MATLAB algorithm. Thus, the MCU will recognize the response and will then display a pre-recorded/written feedback message, in the form of a mute animated slide show, coupled with text. The feedback would either confirm and acknowledge the correctness of the users' form and posture, or it will display the fault in the user's posture, as well as displaying a slideshow of the correct method of doing bicep curls, so that the user can correct his/her posture.

3.0 State of the Art

Using wearable bio-metric devices in rehabilitation procedures and sport activities is not a new concept. Many rehabilitation centres and professional sport leagues, like the English Premier League, Bundesliga, NBA, NHL, NFL, and MLB have been using biometric devices to track and help their athletes to have better performances, in their sport. Like the project presented here, much research has focused on the sensation of the posture, by IMUs and providing a vibrational haptic feedback [20]. *"Directional vs Non-Directional Modes of Vibrotactile Feedback for Arm Posture Replication"* [20], is one of the articles focused on this field, with the objective of improving arm posture, during upper limb rehabilitation. This article uses two IMUs attached on the upper and lower arm sections, along with vibrotactile feedback modules to correct the posture of the user, during his/her arm movement. The orientations of the wrist, elbow, and forearm were used to model the movement of the arm. The vibrotactile feedback is based on two approaches of directional and non-directional [20]. Figure 3 illustrates the mentioned device and how it is modeled.

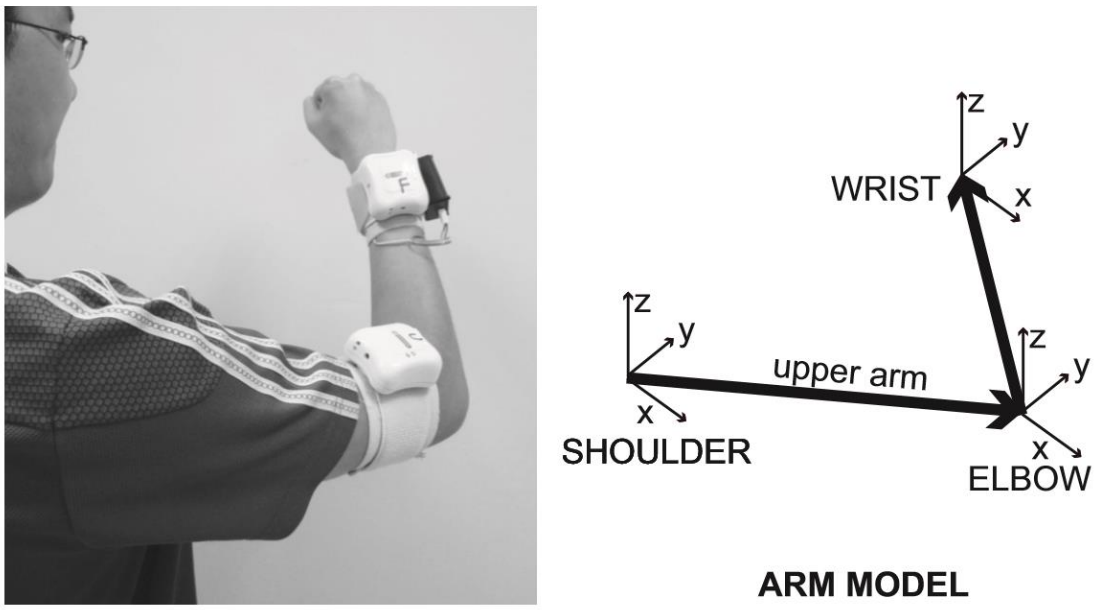


Figure 3. Picture of The Device and How It Is Modeled [20]

The sensory driven feedback devices have also been industrially commercialized to monitor the user's movements physically and dynamically and help users in performing various sports activates, in a correct manner. One the products that has the same approach as the project represented in here is called Moov [12]. Moov can monitor a wider range of sports, in contrast to our project, in which it can just monitor bicep curls. Moov consists an accelerometer, gyroscope, and a magnetometer. The device would use the inertial data and perform internal data processing. The device's algorithm has templates for each exercise, but it can also learn the habits and unique movements, in the user. The device would then decide on the correctness and the accuracy of the selected sports activity [12]. The device's multi-purpose, robust, and accurate functionality makes it an extraordinary design. The device can update the user, in real time, using an artificial speech based auditory feedback. The device can also pair up with one's smartphone and provide text and graphical based feedback [12]. This feedback will provide a performance report to the user, helping him/her to correct posture and exercise methodology. Figure 4 illustrates Moov in action.

Figure 4 Moov Device Used in Running [12]

4.0 Methodology

The proposed method in this project uses observed data from a IMU module, to detect the postural orientation and kinematics. The selected IMU module can provide various inertial data such as linear acceleration with gravitational vector included, linear acceleration with gravitational vector excluded, angular velocity, relative Euler orientation, absolute Euler orientation, and Quaternion orientation [23]. Using a microcontroller, the gravity-excluded linear acceleration, Euler absolute angular orientation, and magnetometer data are extracted from the IMU, using the I^2C protocol. The collected data is then wirelessly sent to a nearby computer, using Wi-Fi. The data is received and processed in the MATLAB environment. The data analysis consists of a series of modifications to the input raw data. The modified data is then thresholded with respect to global pre-set values. The generated results have a deterministic nature, meaning that they only indicate the absence or the presence a certain feature (a True or a False). MATLAB will then send the results back to the microcontroller. The microcontroller receives the data, which will then indirectly display the visual feedback, on the respective TFT screen. By indirectly, it is meant that the main micro-controller is not in direct communication with the TFT. The communication between the two device is facilitated, using an intermediate secondary micro-controller, acting as a Graphical Processing Unit (GPU), responsible for innervating the pixels, on the TFT screen. The respective TFT screen will then illustrate the output, through a user-friendly interface. The following sub-sections will further explain the hardware and software design approaches of the prototyped wearable.

4.1 Data Design

In this section, the overall structure of the data will be explained. The software design can be divided into three components: input, processing, and output. Input sections illustrate the observed data from microcontroller and the properties of the data. Processing parts demonstrate the process, in which experimenters went through to achieve the objective of the project using the raw input data. In the output section, the software-based demonstration of the results on the designed GUI is discussed.

4.1.1 Input

The input data arrangement takes place on the main MCU. The main MCU is in direct communication with the IMU, using the I^2C protocol. This protocol includes the main MCU sending a “read” byte to the IMU. This byte specifies what kind of information the MCU is interested in, by sending the right register address, that needs to be read. Consequently, the IMU will then write the requested information into the main MCU. The IMU can offer fifteen individual sets of data that can be read. There also exists a register, in which the calibration status of the IMU is updated. The MCU can also collect this information to allow the user to efficiently calibrate the IMU [23,1].

The main MCU is Arduino based, thus requiring the Arduino IDE software, to be programmed. An Arduino code is implemented to allow the collection of the data from IMU to the MCU. In order to receive each sample of the desired data, the MCU needs to send the read byte to all of data’s respective registers. The Arduino Code receives each sample and places the data in a data structure. Each component of the data can then be called, using the dot operator. In the Arduino code, the gravity-excluded accelerometer data, Euler absolute angular orientation data, and the magnetic vector magnetometer data are extracted from the sampled data structure. The extractions were then converted to a ‘string’ data type, stored in a buffer register.

The Arduino code also needs to establish a connection with a local Wi-Fi network. The selected MCU has a built-in Wi-Fi module, in which the two communicate, using an internal SPI connection. This establishing code starts with input spaces for putting the SSID and password of the local Wi-Fi. When the code is run, the MCU’s USB serial connection will output the IP and

Port information of the Wi-Fi connection. This information is later needed, for the establishment of the PC-MCU connection. The code utilizes the UDP Wi-Fi protocol which is a wireless communication protocol that does not require the acknowledgement of the listening device (in this case MATLAB running on the PC device). In addition, the protocol does not perform any bit-checking. Bit-checking is a wireless protocol, in which the correct transmission of the wireless bits is ensured. This is relatively a tedious process, that would increase the latency of the data transmission. Due to this, UDP is a much quicker way of transmitting the IMU data [27]. When the code is run, the Arduino MCU will connect to the local router, using its SSID and password. Upon this establishment, the string-casted buffered IMU data will be sent one by one. The Arduino code exists in a state of a semi-infinite while loop, in which the Wi-Fi buffers are frequency updated with the most recent inertial data samples and are sent away, through UDP. The UDP code also utilizes an input mechanism, where the availability of an input data packet is checked upon every iteration of the discussed loop. This is beneficial when trying to receive the feedback results from MATLAB. The inputting of the given IP and port information from the MCU, allows one to establish the PC-MCU connection. Upon inputting the IP and port into the MATLAB code and running, MATLAB receives a train of data, from the MCU. In Arduino code, a byte is defined, in which if it is received from MATLAB, it halts the transmission of the data momentarily, until a 'resume' byte is received, which will re-initiate the train of transmission. The Arduino code is further commented in the Appendix section. Figure 5 illustrates a summary of the wireless communication parties and information exchange.

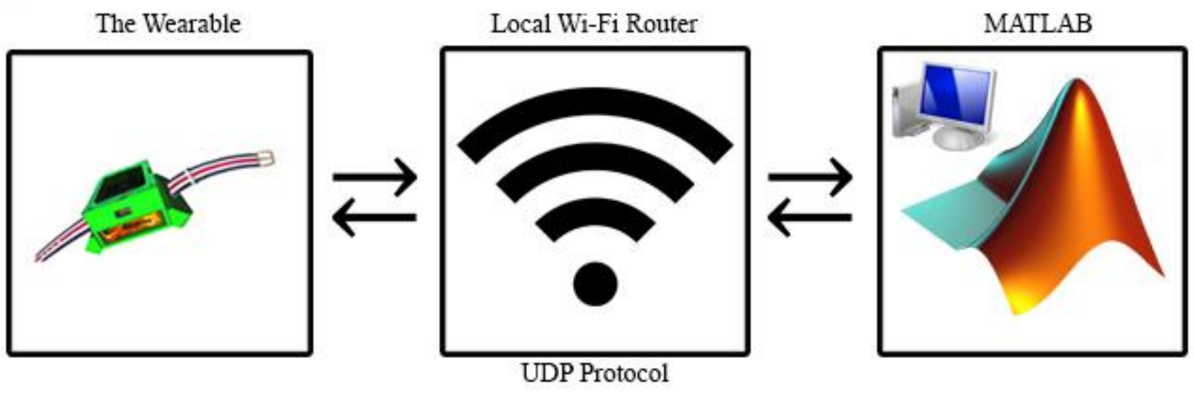


Figure 5. The Implemented Wireless Communication Entities and Relations, In Order to do Postural Correction

Accelerometer works by measuring the acceleration force. Finding the acceleration values in three axis follows Newton's second law provided bellow, in Equation 1. The acceleration force is either sensed by the voltage induced in a piezoelectric element, or by changes in capacitance between microstructures, located on device. The acceleration provided by the accelerometer is m/s^2 [23,24].

$$\vec{F} = m\,.\,\vec{a} \qquad (1)$$

Where $\vec{F}$ is the force vector, m is the mass and $\vec{a}$ is the acceleration vector, in the 3D space.The accelerometer used in this project works by measuring the changes in capacitance between two actuating plates. The microstructure of this accelerometer has a mass attached to a spring that moves along a direction. As the mass moves toward a certain direction, it leans more in the opposite direction of the motion, thus causing the capacitive plates to change their capacitance [26].

The Gyroscope has a similar design to the accelerometer, but it measures rate of rotation in three dimensions, including the yaw, pitch and roll [24]. This is done by using the Coriolis Effect. Coriolis effect explains that when a mass is moving in a direction and an angular momentum is applied to it externally, a force will be generated perpendicular to the direction of the mass which will cause the displacement of the mass. Like the accelerometer, the change in capacitance measures the displacement of the mass [26].

Magnetometer is the third type of sensor used for this project. This sensor senses the magnetic filed vector of the earth, with respect to three axes. The sensor in magnetometer is a Hall-effect sensor, which produces a voltage proportional to the strength and the polarity of the magnetic filed intensity. This can also be done by using a magnetic field dependant resistive material, which changes its resistivity, in the presence of a magnetic filed [25].

4.1.2 Processing

The processing of the data is done in MATLAB. Using an infinite while loop, the process will be run in real-time, as long as the wearable is turned on, connected to the router, and transmitting the data. Before the while loop, initial conditions for different matrices are set and required libraries are recalled. The general processing procedure in the MATLAB consists of data acquisition from

MCU, feature extraction and then the decision-making algorithm that detects a specific postural error/fault based on the observed features, extracted from the input data.

For data acquisition, in MATLAB's while loop, MATLAB is connected wirelessly to the MCU, by also connecting to the local router. Initially, the available wireless networks are realized on the Windows PC. Thereafter, the PC is connected to the established Wi-Fi network, given the SSID and password. In MATLAB, the IP and Port ID provided by the MCU are entered, thus establishing the desired wireless connection. Thereafter, the MATLAB and MCU are synchronized by adjusting the wireless input read frequency, on MATLAB. Setting this timing for MATLAB is crucial because the wrong timing will result in connection 'Time-out' initialized by MATLAB, causing a disconnection. To make ensure avoiding a connection time-out, MATLAB halts the data transmission by wirelessly sending a 'halt' byte, upon the arrival of each data packet and is re-initiated right after, using a 'resume' byte. Using this method, the disconnection is avoided. This method will also ensure the transmission consistency of data between the two devices. The data is read from the Wi-Fi port and is then converted from a 'string' type to a 'double'. The received data consists of a multitude of inertial data families such as gravity excluded linear acceleration, absolute Euler angular orientation, and magnetic vector data, from the magnetometer. Each family of data is placed onto its three-dimensional matrix. Figure 6 represents the plot of 2 families of raw data, received wirelessly from the MCU.

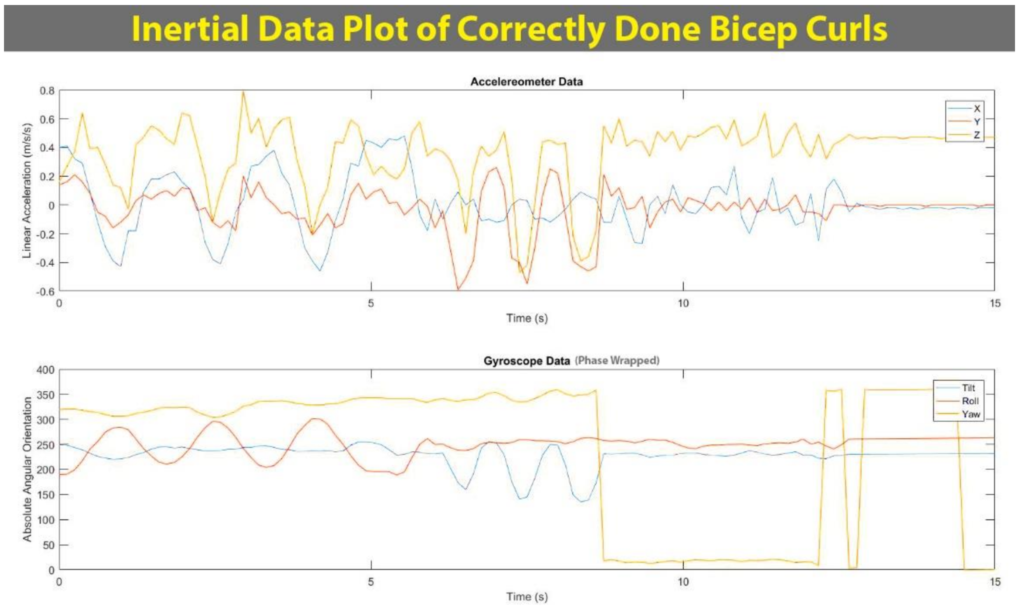


Figure 6. The Linear Acceleration Data and Phase Wrapped Absolute Orientation Data of 15 Seconds of Bicep Curls

First step of the processing was using simple moving average algorithm to filter linear acceleration independently in each direction. The data was averaged over three points which made the output smoother. This method is typically known as three-point average low-pass filtering [13].

The first feature which was extracted and used to detect a faulty bicep curling posture was through thresholding the magnitude of the gravity-excluded linear acceleration in the Y-direction. A threshold of 15 was set for this value, during the biceps exercise, any rapid movement would result into exceeding this value and therefore observing error.

The second feature which was extracted and used to detect a faulty posture, was the value of the absolute Euler angular orientation in the Z-direction, which came from the gyroscope, on the wearable. During the bicep curl exercise, the wrist must be steady and should not rotate. Any rotation or tilting would be detected as error by setting a Boolean threshold of 3 for this value.

The Magnetometer's magnetic filed vector data in the Y-direction was used to extract the third set of features. The objective of this feature extraction was to observe the dynamic range of the arm movement in Y-direction. A truncating window with a length of 10 sample data points was set and the values within the window were analyzed to observe the dynamic range. The local maxima and minima values of the windowed data were and the distance between them were determined, as the range. Any range lower than value of 60 would trigger a postural fault.

Using the same dataset, a bicep curl repetition counter was also implemented by finding the number of peaks in the derivative of the magnetometer data, in the Y-direction. The peak detection algorithm also consists of a thresholding mechanism. The threshold was set to 30. In the following, equation 2 can be used to express the bicep curl repetitions, from the magnetometer data.

$$y(n) = \sum_{n=1}^{\infty} x(n-1) - x(n) \qquad (2)$$

$$C = \begin{cases} C+1, & x > 30 \\ 0, & x < 0 \end{cases}$$

In Equation 2, x(n) is the updated magnetic field intensity, in the Y-direction, y(n) is the discrete derivative of x(n), and C is the repetition count. Figure 7 represents the signal processing procedure, done in the MATLAB environment, that would result in the respective postural fault detections.

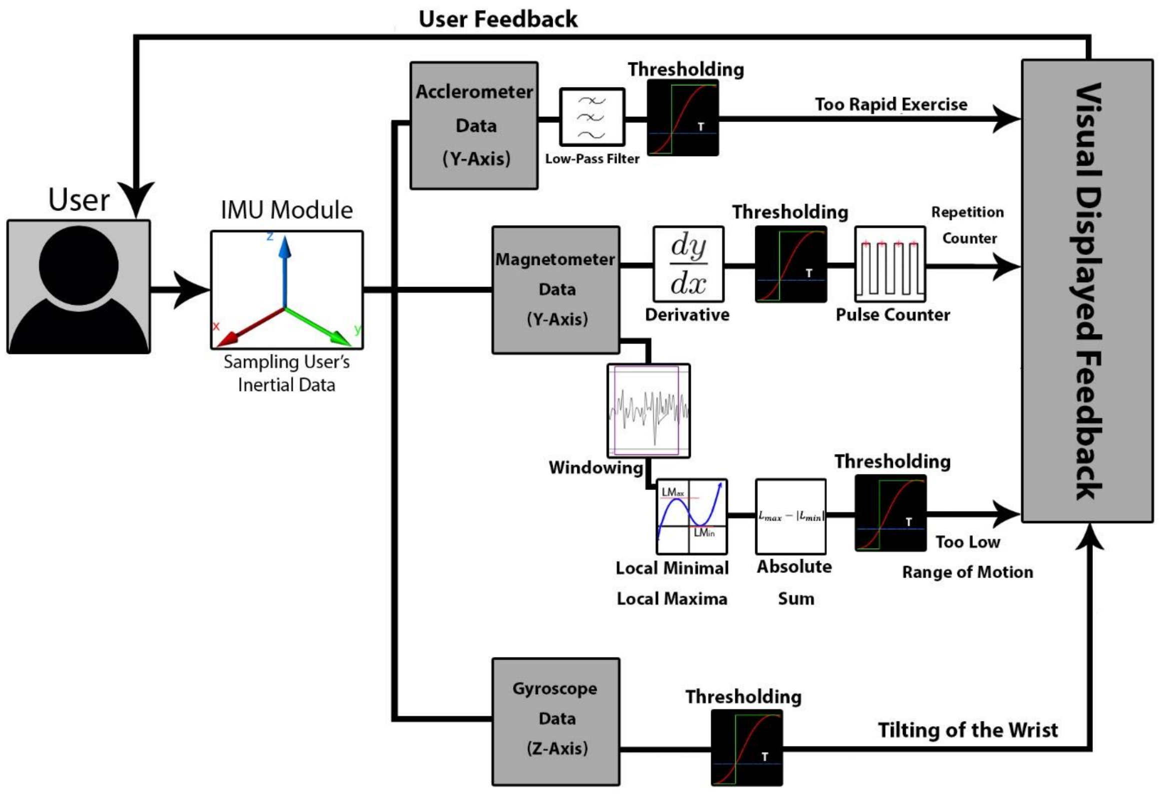
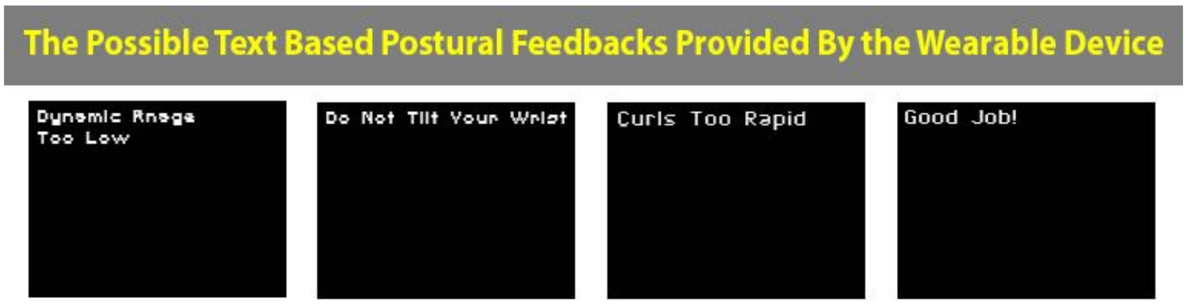


Figure 7. Signal Processing Procedure Block Diagram

4.1.3 Output

In this project, the output is the visual feedback of the wearable device. This feedback is provided visually on a TFT screen, on the device. The displayed feedback is in the form of text and an image slideshow, showing the correct methodology of bicep curls, as well as a textual statement, outlining the user’s postural mistake. The main MCU will wirelessly receive a byte command from the MATLAB environment, where the exercise faults are detected. The received MATLAB command signifies what is the proper feedback that would correct the user’s posture. If there are no mistakes in the user’s posture, the sent byte command will acknowledge the correctness of the user’s posture. The replied byte would be followed by a series of serial bits, in which the number of the pulsed bits would signify the number of bicep curl repetitions, done by the user. For example, when MATLAB detects an overly rapid exercise, a byte command will be sent to the MCU, to turn on the TFT and display and write out the users’ mistake, followed by a slideshow, showing the corrected method. Since the LCD module requires a 5 V supply and input logic and the main MCU runs on 3.3 V, as discussed, another microcontroller is dedicated for graphical processing. The implemented Graphical User Interface programmed on the second MCU and executed on the TFT, works by receiving a pulsed dial signal from the main MCU, meaning that depending on the number of pulsations that has been send, the second MCU will know what to illustrate on the TFT screen. Furthermore, in MATLAB, for each response or command, specific pulse dial sequences is set. More information about the GUI is explained in the Interface section, of the report. Figure 8 illustrates the possible text based feedbacks that is displayed on the wearable's TFT screen. Figure 9 illustrates the slideshow that is displayed to the user, after the text based feedback, showing the correct methodology of doing bicep curls.


The Possible Text Based Postural Feedbacks Provided By the Wearable Device

Dynamic Range Too Low

Do Not Tilt Your Wrist

Curls Too Rapid

Good Job!

Figure 8. Text Based Feedbacks

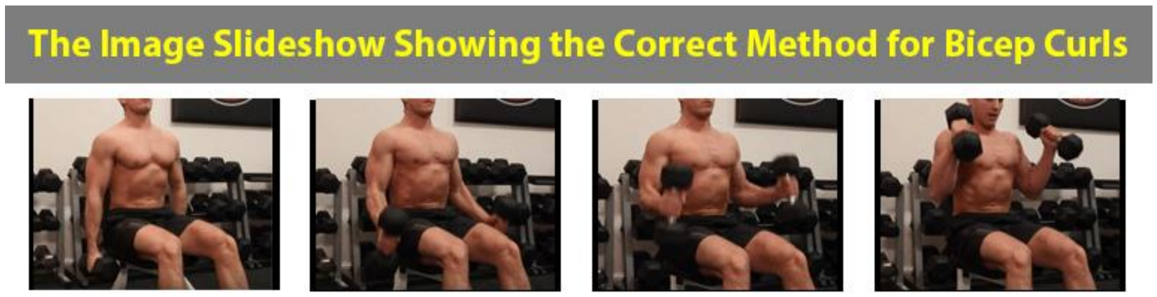

The Image Slideshow Showing the Correct Method for Bicep Curls

Figure 9. (Ordered Left to Right) Displayed Image Slideshow of a Correct Bicep Curl

5.0 System Structure

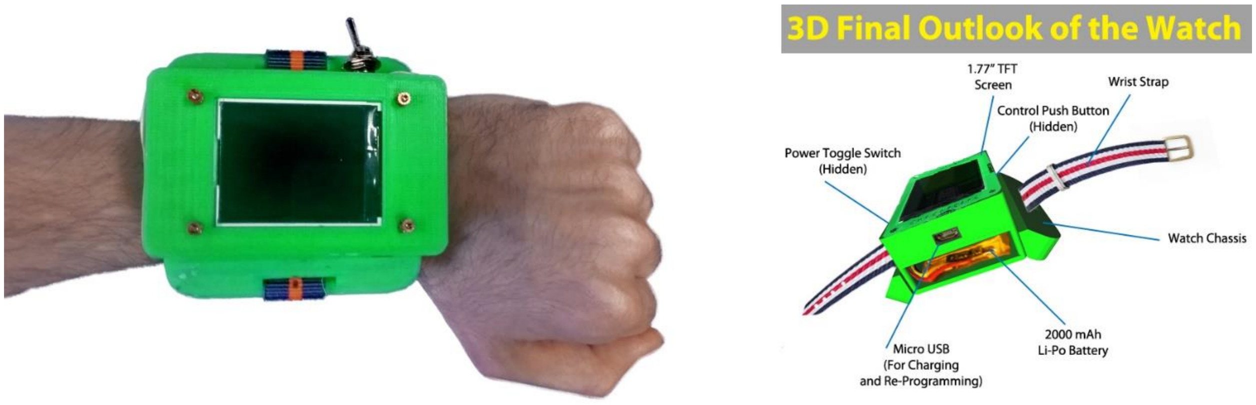

3D Final Outlook of the Watch
1.77" TFT Screen
Wrist Strap
Control Push Button (Hidden)
Power Toggle Switch (Hidden)
Watch Chassis
2000 mAh Li-Po Battery
Micro USB (For Charging and Re-Programming)

Figure 10. The Assembled Version of the Prototyped Wearable

Figure 10 represents 3D illustration of the assembled wearable, as well as the real life assembled version, with the integrated components inside. The wearable encases a MCU, capable of battery charging and Wi-Fi capability, an IMU unit that has the accelerometer, gyroscope, and magnetometer, an on/off toggle switch, a tactile push button as an input interface, a 2000 mAh Li-Po battery, a TFT screen, a second MCU for graphical processing, and a boost switching converter, to produce 5 V from 3.3 volts. The components will be enclosed in a casing with two strap lugs, allowing for the firm harnessing of a watch strap. The case was designed in AutoCAD by the group. The casing consists the main chassis, which encloses all the components, as well as a top lid which will close the top end of the chassis and holds the TFT in place. A separated chamber was implemented, as a part of the main chassis, which holds the battery in place. This was done to give the user an easier access to the batter, allowing him/her to quickly exchange the battery. The components will work in harmony to collect IMU data, transmit the data, receive the results, and display the feedback for the user. The display and the button will respectively establish an output and input user interface for the wearable. Figure 11 represents the 3D design of the chassis, in AutoCAD. The chassis is dimensioned at 68x68x30 mm.

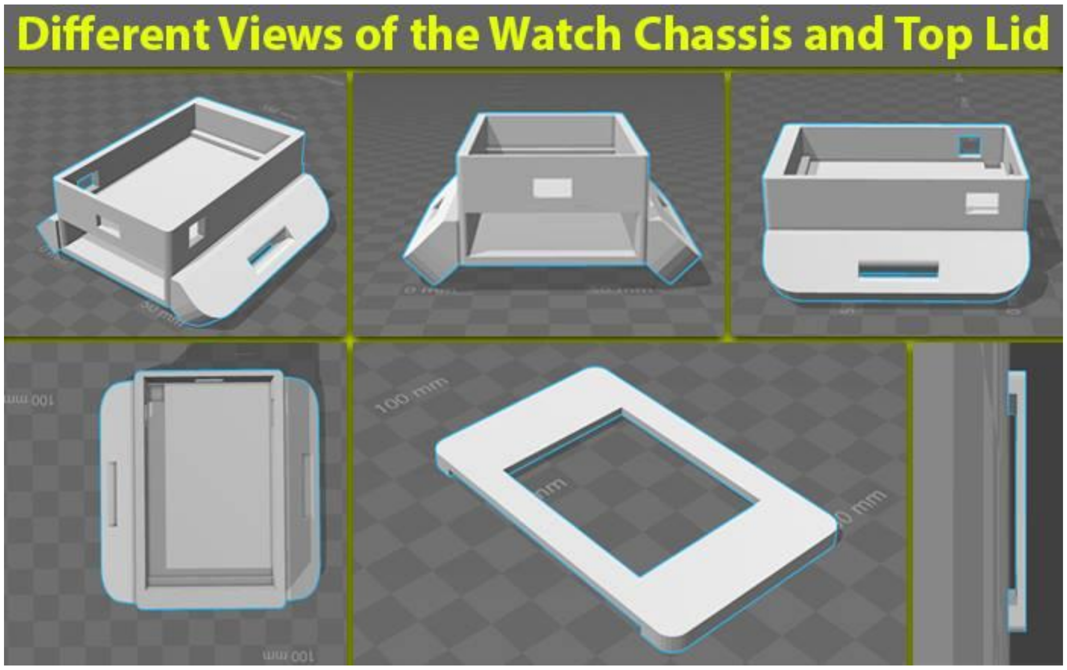


Figure 11. The 3D Design of the Chassis

5.2.1 Description for Components

5.2.1.1 MKR1000

The MKR1000 module is a board made by Arduino that has a Li-Po battery charger, Wi-Fi module, and a M0+ ARM Cortex MCU. This module was chosen as the main MCU. The USB port allows for the charging of the connected battery, as well as the programming of the MCU. The module has 7 analog inputs and 1 analog output, with a clock speed of 48MHz [22]. The high-performance functionality of the board, its easy programming through the Arduino IDE, and its small size, makes it a great candidate for wearable applications [1]. The schematic and data sheet for the component is provided in the reference [21, 22]. The board has SPI, I^2C, and digital I/O ports, allowing for the communication between the peripherals and the MCU. The module also can power the selected peripherals, through its internal voltage regulation. The module also has a small power consumption, allowing for a longer battery life, when running on a Li-Po battery. The module has an operating voltage of 3.3 V, making it limited to work with 3.3 V compatible peripherals. Figure 12 represents an image of the MKR1000 MCU module.

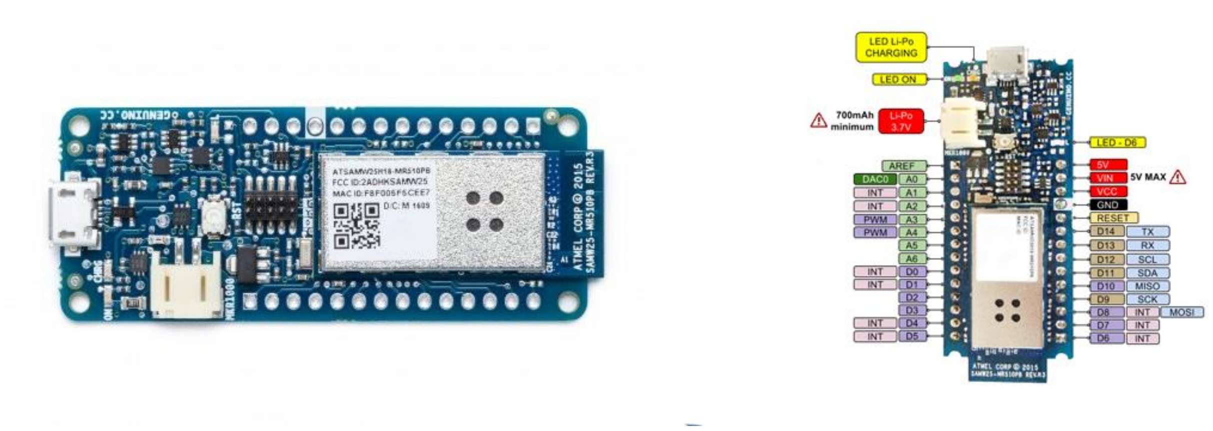


Figure 12. Arduino MKR1000 Module (Main MCU)

5.2.1.2 **BNO055** ***IMU***

BNO055 is an outstanding Inertial Measurement Unit (IMU). This is so due to being a 9 Degrees of Freedom (DoF) sensor, containing an accelerometer, gyroscope, and magnetometer. The unit also contains a ARM based MCU inside, allowing it to independently perform signal processing. The internal processor can fuse the inertial data, thus allowing the designer to directly extract absolute orientation vector data, as well as linear acceleration data that has the gravity component subtracted from it. The processing unit allows it to perform auto-calibration. All these features would accelerate the prototyping process. The module is both 3.3 V and 5 V compatible, making it a good candidate to use with the MKR1000 microcontroller. The device offers a 16-bit resolution gyroscope, with an adjustable measurement range of 125 to 2000 ^{o}/s. The accelerometer has a 14-bit resolution, with an adjustable measuring range of 2 to 16 m/s^2. The device has an automatic mode, allowing the device to quickly switch between measurement ranges, when necessary. The device can output data, at a maximum rate of 100 Hz. At this output data rate, it draws a total of 12.3 mA, making it a low power device. This characteristic makes it suited for wearable applications [23]. Figure 13 represents an image of this module and its internal architecture.

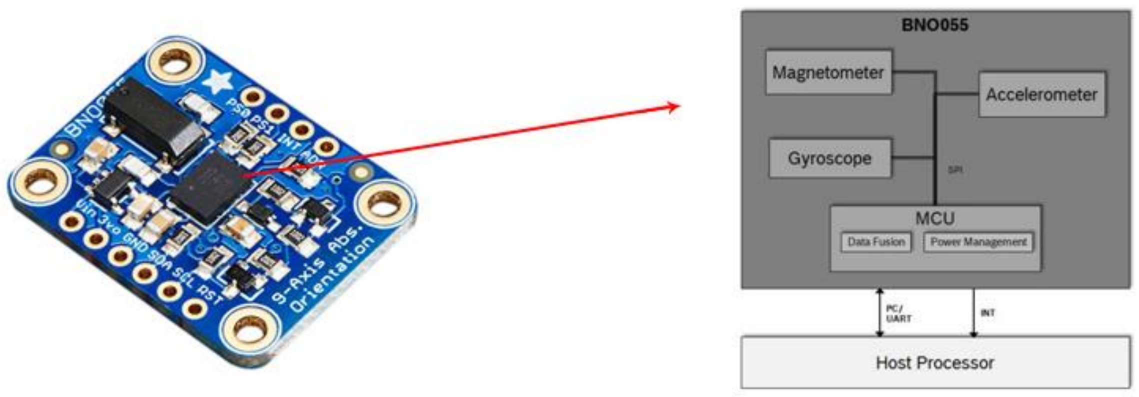


Figure 13. BNO055 IMU Module. (Right) BNO055 Internal Architecture. (Left) BNO055 External View

5.2.1.3 Switches

There is a total of 2 switches on the device. One of them is a tactile push button, which acts as a user input interface, allowing the user to user the GUI, initiate data collection, terminate data collection, see results, and restart the device. The power switch is a toggle switch, which is run in-series with the cathode of the Li-Po battery. This feature allows the device to be turned off, when needed. This also ensures a null power consumption, during the off time. Figure 14 illustrates the switch and the button that is mounted onto the watch chassis.

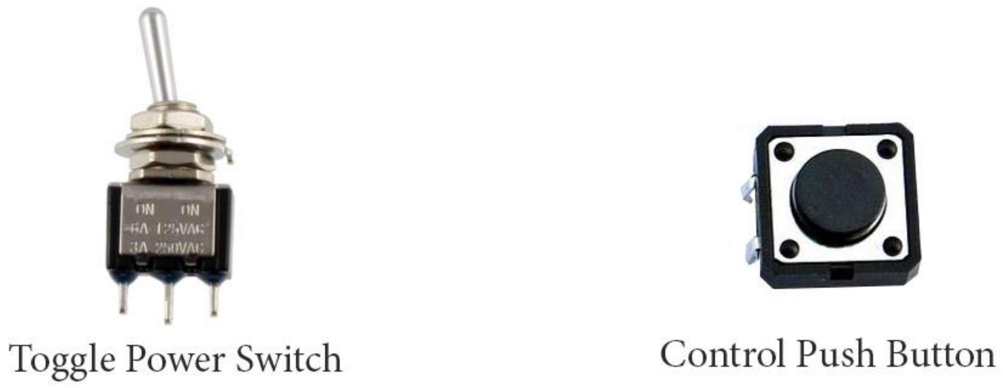


Figure 14. (Right) The User Interface Push Button. (Left) The Toggle Power Switch

5.2.1.4 Li-Po Battery

The battery used is a 2000 mAh Li-Po battery. The high capacity of the battery allows the wearable to last long before needing to be re-charged. The battery has a built-in protection circuit that prevents over-charging, short-circuit, and over-current. The battery utilizes a connector that straightly connects to the MKR1000, allowing it to provide power and be re-charged. Figure 15 represents an image of the Li-Po battery.

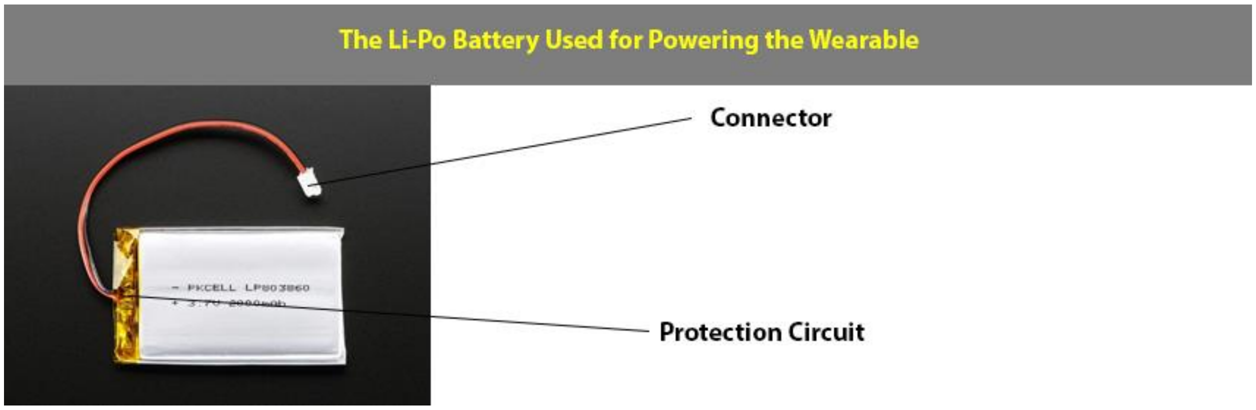


Figure 15. Li-Po Battery Used in the Prototype

5.2.1.5 TFT Display

The TFT display is used to provide visual feedback to the user on his/her athletic performance. The TFT displays use the SPI interface to communicate with the mini MCU. The TFT screen also accepts an SD card, allowing it to hold more memory. This memory is used to store the slide show pictures, due to the limitations on the MCU memory. The TFT display is made by Arduino. The screen is 1.77 inches in diameter, with 160 by 128-pixel resolution. This screen was chosen because the manufacturer had provided a lot of instructions on how to run the screen. Unfortunately, this screen runs on 5 V, preventing it to work with the MKR1000. Due to this fact, a second 5 V MCU was chosen and a boost converter was implemented to provide 5 V both to the second MCU and the TFT screen [TFT screen citation]. Figure 16 represents the outlook of the TFT screen.

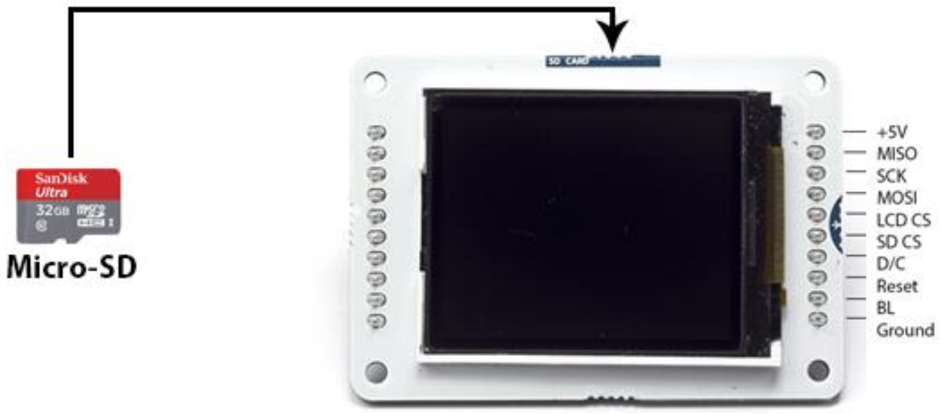


Figure 16. The TFT Screen, Used in the Prototype

5.2.1.6 S7V8A - Switching Boost Converter

This device was chosen to boost the 3.3 V output voltage of the MKR1000 to 5 V. The boost converter has 4 pins (Input, Ground, Output, and Enable). The 'Enable' pin allows the module to be turned on and off, allowing the main MCU to decide when to turn on the device. The module has a surface mount potentiometer, allowing the designer to adjust the output voltage. The potentiometer was pre-set, allowing the module to output 5 volts. The module is small and can provide a continuous output current of 800 mA, making it suitable for this application [29] . Figure 17 represents the outlook of the boost converter.

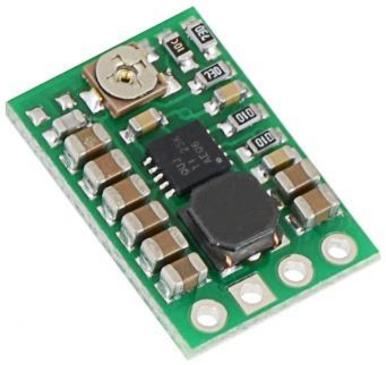


Figure 17. The S7V8A Boost Converter

5.2.1.7 Arduino Pro Mini

The Arduino Pro Mini was chosen as the GPU that runs the TFT screen. The board works on 5 V and has a base clock frequency of 16 MHz. The MCU is very small and does require an external FTDI breakout to be programmed. This makes it suitable for the wearable application. The MCU provides the TFT screen with 5 V logic communication and as well, it lifts some computational load off from the main MCU. The SPI programming of the TFT screen imposes a great deal of computational load. The addition of the MCU was meant to also speed up the process. Since this is also an Arduino board, its programming is easy and the available libraries are abundant. The board can also accept 3.3 V as an input, when the input pins are configured as an input pullup. The board is glued at the back of the TFT screen and is wired to the TFT screen, using SPI [31]. Figure 18 represents the outlook of this microcontroller, as well as the FTDI breakout used to program it.

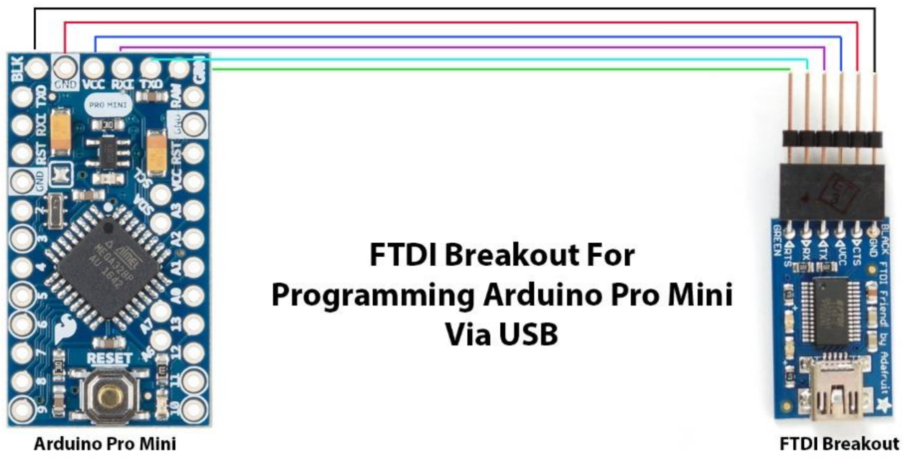


Figure 18. Arduino Pro Mini and its FTDI Breakout Board, Used for Programming

5.2.2 System Interaction

Figures 19 outlines the interaction between the components in the wearable.

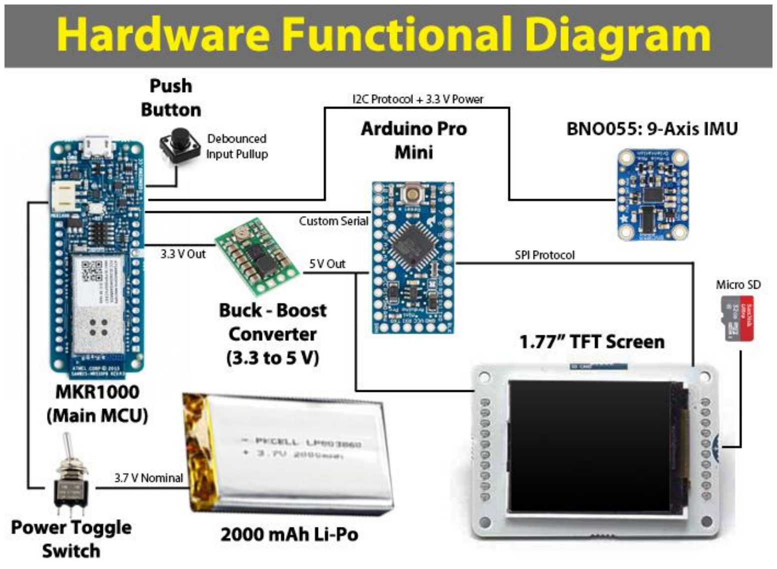


Figure 19. Diagram Showing the Functional Interaction Between the Hardware Components

The system starts at the Li-Po battery, where the required energy is stored and a nominal voltage of 3.7 V is provided. The battery is in series with a switch which allows the wearable to be turned on and off. The power line continues, until it reaches the main MCU, the MKR1000. The MKR1000 is constantly listening (polling) to the pressing of the push-button. The button connects an input pullup pin and the ground pin. When the button is pressed, the gate voltage of the input pin is pulled to zero and a logic low is detected, where it can trigger an event. The MCU is also constantly receiving inertial data from the IMU and transmits the data through Wi-Fi if it is not told to stop, by MATLAB. The main MCU is also constantly wirelessly listening to MATLAB for any incoming data packets. The main MCU's V_{cc} pin, providing 3.3 V is connected to the boost converted, where the voltage is boosted to 5 V, supplying the TFT screen and the Arduino Pro Mini. Lastly, two of the main MCU's output pins are connected to two input pullup pins on the Arduino Pro Mini, where the pulsed dialing communication takes places, and the displayed content is controlled. The main MCU grounds the second MCU's input pull-ups, thus delivering the intended message. Figure 20 illustrates the CMOS connection between the MKR1000's output and Arduino Pro Mini's input, hinting about how they communicate. As seen in Figure 20, the output of the MKR1000 can only ground the input of the Arduino Pro Mini. The input of Arduino Pro Mini is normally in a high state and the high state of the MKR1000 does not affect this because the n-channel CMOS, on the MKR1000 has a negative over-drive voltage and cannot be turned on. Though, a logic low output from the MKR1000 will discharge the gates of the Arduino Pro Mini's CMOS buffer, which will result in a logic low signal. The MKR1000's I/O pins can only handle a maximum current of 7 mA. The input pullup resistance of the Arduino Pro Mini's pull up resistance is high enough to prevent the over-currenting of the MKR1000's input pins. This can be proven, using Ohm's law calculations shown below.

$$R = \frac{V}{I} \qquad I = \frac{V}{R} \qquad I = \frac{5\,V}{20\,000\,\Omega} = 25\,\mu A \tag{3}$$

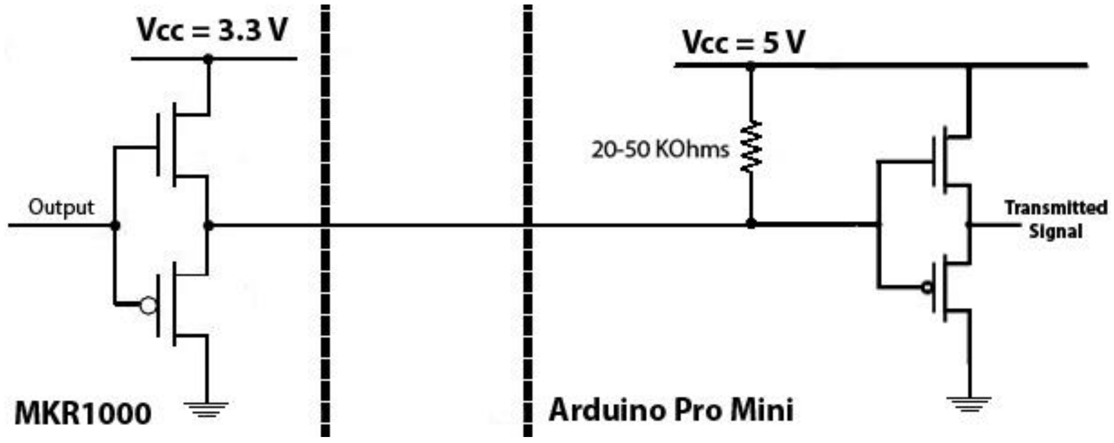


Figure 20. A Closer Look at the CMOS Connections Between Arduino Pro Mini and MKR1000

The Arduino Pro Mini is connected to the TFT screen, through the SPI protocol. The TFT screen requires 10 wire connections, including: MISO, MOSI, SCK, LCD Slave Select, SD-card Slave Select, Reset, Enable, Ground, backlight, and V_{in}. Most of the communication happens through MOSI and the SCK provides a clock signal, synchronizing the two together. SPI is a fast data transfer communication protocol and can handle quick refresh rates on the TFT screen. The Arduino can selectively write/read into/from the TFT and the SD-card slot, by driving the respective slave select pins to a logic high. The backlight and V_{in} pins are connected to the 5 V rail. The ground pin is common grounded to the Arduino Pro Mini and MKR1000. Figure 21 illustrates a more detailed diagram of the wiring schematic of the prototype.

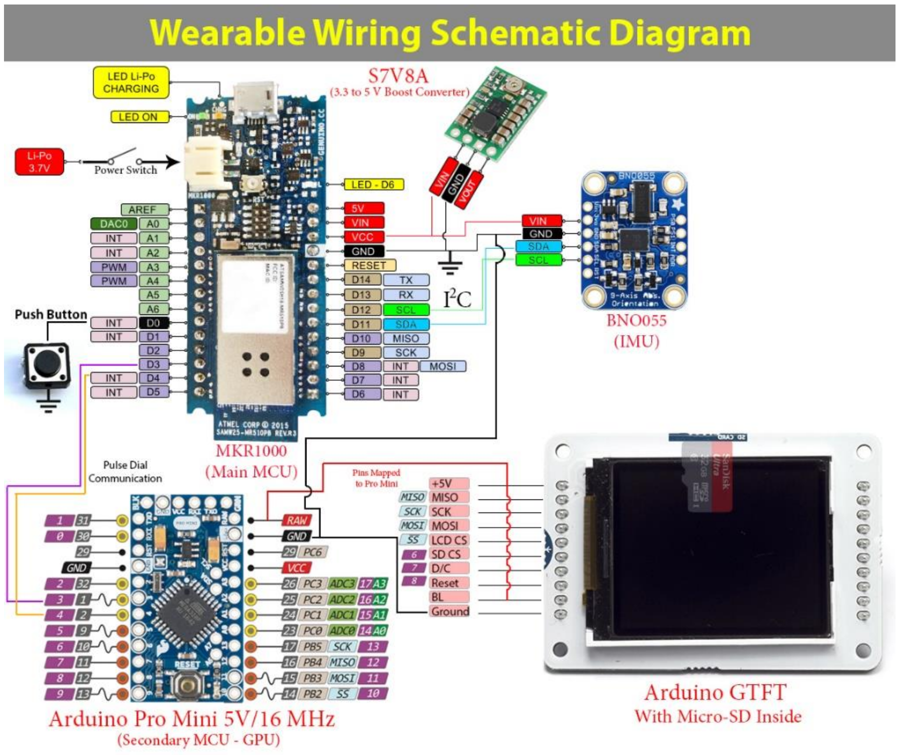
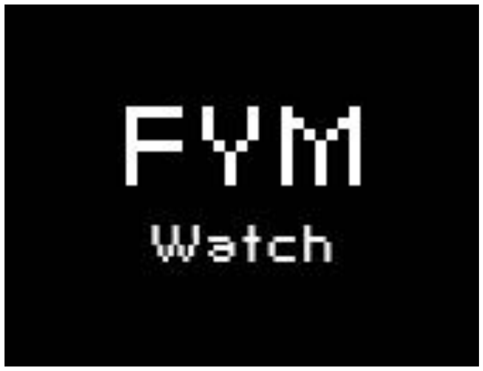


Figure 21. The Wiring Schematic of the Prototype

The discussed pulse dial communication between the Arduino Pro Mini and MKR1000 is a made-up communication protocol. The pulsations made by the MKR1000, in two separate wires can control the things that are displayed onto the screen. The two wires include a green wire (between pin 3 of the MKR1000 to the pin 3 of the Pro Mini) and a blue wire (between pin 4 of the MKR1000 and the pin 4 of Pro Mini). One pulse dial by the MKR1000 consist of a 50 ms grounding of the respective pin. The green wire (displayed in Figure 21 as pink) usually does tasks such as skipping a page on the GUI and the blue wire (displayed in Figure 21 as orange) has a more executive power. Both wires are uni-directional. The green wire is initially used to toggle between the first initial pages of the GUI. A single grounding of the green wire would result in the flipping of the GUI screen to the preceding screen. When it comes to the rep courting, the MKR1000 wirelessly knows the number of performed bicep curl repetitions from MATLAB. MKR1000 will then sequentially pulse the number of repetitions to the Pro Mini. For example, if 15 repetitions are done, 15 pulses are made. The pulses are through the green wire, and are 50 ms in duration and 100 ms apart. When the repetition pulsations are done, the blue wire is pulsated, letting the Pro Mini to know that there are no more repetitions to be displayed. Now that it's time to display the text based feedback, the green wire is again used. For the “Curls too rapid” response one green wire pulse is needed. Two green pulses would translate to the “Don’t tilt your wrist” response. Similarly, three green pulses translate to “Curl Dynamic Range Too Low” response. Lastly, four green pulses translate to the “Good Job” response. Thereafter, a blue pulse from the MKR1000 results in the text feedback being displayed, followed by the image slideshow. Thus, a major constraint of the device is that it can only display one of the available feedbacks and displaying multiple feedbacks is not a possibility.

5.3 Graphical User Interface Design

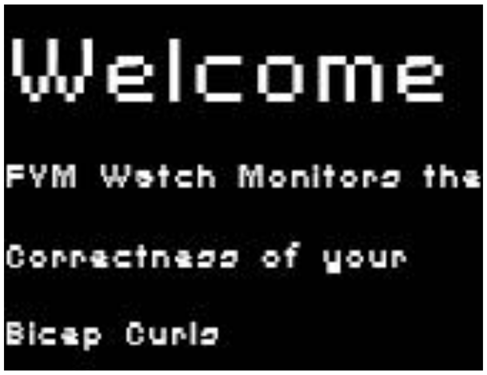
The Graphical User Interface designed for this project heavily relies on the text-based visuals. After turning on the wearable, by flicking on the power switch, the screen will display the watch’s logo, writing “FYM”, which is the name of the wearable. Figure 19 illustrates this display.


FYM

Watch

Figure 22. The Opening Display, on the TFT Screen, When the Wearable is Turned On

The GUI will the proceed with displaying a welcome message to the user, and provides a brief description on the main purpose of the watch. Figure 20 illustrates this.


Welcome

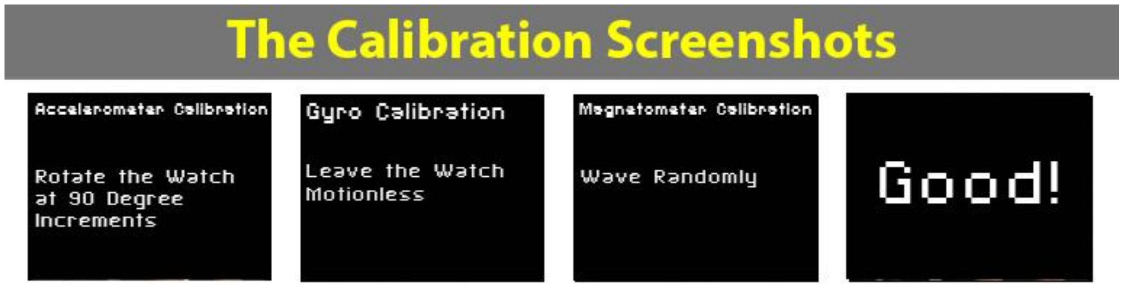
FYM Watch Monitors the

Correctness of your

Bicep Curls

Figure 23. Welcome Screen and Statement of Purpose

Thereafter, the device will ask the user to perform specific hand motions, in order to calibrate the IMU module. For calibration of each set of data (Accelerometer, gyroscope, magnetometer and gravitometer), specific hand movements are required, therefore the interface will ask the user to do each, one at a time. For example, the accelerometer will be calibrated via rotation of the watch by 90 degree increments in the sagittal, coronal, and transverse planes. The gyroscope will be calibrated when the watch is held motionless. When the calibration of each IMU component is successfully over, a "Good" text message will appear on the screen. Figure 24 illustrates these displays.


The Calibration Screenshots

Accelerometer Calibration

Rotate the Watch at 90 Degree Increments

Gyro Calibration

Leave the Watch Motionless

Magnetometer Calibration

Wave Randomly

Good!

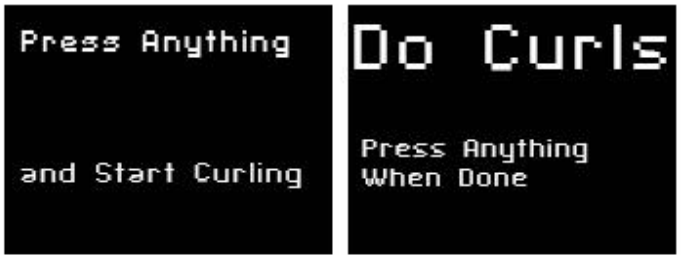
Figure 24. (3 on the Left) Calibration Pages with Hand Motion Instructions. (Right) Message Displayed When Each is Calibrated

Figure 25 illustrates the displayed message when the IMU is calibrated.

IMU Ready

Figure 25. Message Displayed When the IMU is Calibrated

For the program to know exactly when to start collecting and transmitting the inertial data, the interface asks the user to press the push button before starting the exercise. The user is also required to stop the data collection by pressing the push button, after the exercise is over. Figure 26 bellow illustrates this.


Press Anything

and Start Curling

Do Curls

Press Anything
When Done

Figure 26. (Left) Message that Waits for the User to Press the Button and Start Exercising. (Right) Message Telling the User to Perform Bicep Curls and Press the Push Button When Done

The main objective of the interface is to provide feedback to the user regarding his/her posture and the exercise. Thereafter the already discussed feedback message consisting of four possible outcomes of: “Curls too Rapid”, “Don’t Tilt Your Wrist”, “Curl Dynamic Range Too Small”, and “Good Job” is displayed to the user, based on his/her performance. The feedback also includes the display of the number of repetitions to the user. Thereafter, the discussed slideshow is displayed to the user. Lastly, after the slideshow, the device resets.

5.4 Experimental design and major constrains

The major subjects of the design that were researched and studied before the implementation of the prototype were the IMU, MCU, MATLAB code, and an LCD. These components remained as an essential part of the design. However, there were some changes in which specific parts to select and what other components are needed to make the design work smoothly. This section will explain the reasoning behind the design and the constrains that forced the group to use parts. In this section, the experimental design alongside major challenges in the design and implementation of this system will be discussed, in detail.

5.4.1 The Microcontroller

As it was mentioned before, for this project the MKR1000 has been used, as the main microcontroller. Initially, the Teensy 3.6 was selected to serve as the main MCU of the wearable. The Teensy 3.6 was purchased and experimented on. It was realized that this microcontroller is an outstanding micro-controller in terms of power consumption and performance. Alongside of the Teensy 3.6 MCU, a Wi-Fi module and a battery charger were separately purchased. It was realized that the three items are bulky, when put together. Thus, group researched further and found the MKR1000, which has a battery charger, a Wi-Fi module, and an imbedded MCU.

5.4.2 IMU

The purpose of IMU is to observe sensory inertial data precisely and accurately and send it to the microcontroller. The first purchased IMU was the GY801. The module has separate dies for the accelerometer (L3F4200D), gyroscope (ADXL345), and magnetometer (HMC5883L). The GY801 IMU was found to be bulky and hard to calibrate. In addition, it was found that the gyroscope data had a lot of angular drift, making it difficult for the group to leave the IMU on for very long and subtract the gravity vector from the accelerometer data. The module also required a lot of fine turning, in its setting register, for the optimal performance. Thus, the team considered the BNO055, which had automated all of the fine tunings and calibration. As well, the IMU performed internal data processing, providing the group with ready to use data. The BNO055 was found to be less bulky and power consuming than the GY801 [30].

5.4.3 LCD

The initial design decision of the group was to illustrate the feedback, using a short video. The initially purchased TFT was powerful enough to perform such a task. The TFT also had touch capability. However, the TFT was malfunctioning from the beginning and the group had to return it and ask for a refund. Due to this defect and long shipping times, the group decided to downgrade the TFT into less powerful TFT screen which is still able to demonstrate a visual feedback and prove the concept behind the design. The initial TFT was called by Adafruit and was 2.8 inches in diagonal and had capacitive touch sensing. It also featured an micro-SD card slot and used the SPI protocol to display images as well [28] . The old TFT was switched with Arduino’s GTFT, which is smaller in size, does not have touch capability, and also has a micro-SD card slot.

5.4.5 Processing

Initially, many different processing approaches and proposed online codes were tried to process the inertial data. The open source codes and approaches proposed that gait-tracking is a good way of analyzing the bodily posture [15]. They also proposed the virtual 3D visualization of the body [16] as well as using the Kalman Filter to find the absolute orientation of the IMU module [17-10]. From the mentioned sources, the open source codes were analyzed and tried to be implemented. Unfortunately, do to the nature of the initial conditions which differed from the group's, the group could not observe the same results as the researched publications. However, the understanding from these researches were used to implement much simpler algorithms to detect the postural errors. Many attempts were made to find the absolute orientation of the IMU module, until the BNO055 IMU was purchased, which readily provided the group with such information.

5.5 Robust, Stable, and High Bandwidth Wi-Fi Communication

One important aspect of the design is the communication between the PC and the microcontroller. The aim is to have a Wi-Fi connection that can send a bundle of high resolution data, at a high sampling rate. This connection had to stable, meaning that it has to stay active without disconnection or any interruptions. This configuration had to be achieved through the optimization of the MCU code, and the MATLAB code. The achievement of this configuration requires time and proficiency for coding, and the understanding of the Wi-Fi communication protocol [1, 2, 3]. This was and still is a slight issue, such that the connection between the MCU and the PC experiences disconnections, rendering the postural correction procedure useless. A small delay in the transmission of the IMU data can trigger a disconnection from MATLAB's side, causing the entire connection to go down. This is thought to be fixable through MATLAB's error handling procedures.

5.5.1 Functionality of the IMU and its Timing

The IMU does not output its raw data as linear acceleration and absolute orientation. The raw data is in the terms of relative acceleration and angular velocity. Both the accelerometer and gyroscope units of the IMU have selectable sensitivities. Firstly, the sensitivity has to be selected according to the intended exercise that is up for analysis. The sensitivity control can be set to automatic, which allows for a smoother operation. For example, the accelerometer has four options for its sensitivity. These sensitivities are: ±2, ±4, ±8, and ±16 g. The accelerometer's sensitivities are in the unit of multiples of Earths' gravitational acceleration constant (9.815 meters per second per second). As well, the gyroscope has three sensitivities of: ±250, ±500, and ±2000 degrees per second (dps). These sensitivities respectfully govern how fast the device can be accelerated, and rotated. If the physical motion of the IMU surpasses its sensitivity, then the output will saturate and will be rendered useless. One other constrain of the accelerometer is that it will pick up the earths gravitational field (1 g). Based on the orientation of the IMU, in 3D space, this feature can be a nuisance in data analysis and must be compensated for. As for the Gyroscope, in order to get the absolute orientation of the IMU, its output needs to be integrated over time. This integration is done inside the MCU. Due to small latencies in the execution of the MCU, small latencies in the data transmission from the IMU, and the small noise present in the Gyroscopic data, the computed absolute orientation will introduce a drift, which is considered a noise in the data. Overtime, the X, Y, and Z orientation values will gradually increase at a constant pace, even if the IMU unit is absolutely stationary [3]. This artifact is eliminated, when the BNO055 does the internal signal processing.

5.5.2 Correct, Accurate, and Precise Posture Analysis/Detection

Most importantly, the detection system (the wearable and MATLAB) has to correctly detect and correct the user's posture. Thus, the implementation of the posture detection software must be novel and accurate. The software implementation has to include a variety of cascaded algorithms such as: data sorting, data and signal processing, relations/correlations, filtering, feature extraction, feature selection, and decision making. The experimenters were able to put all components together in the form of a watch which functions. Three different types of error were implemented which is enough to demonstrate the concept.

6.0 Results and observations

This section outlines the testing procedures done on the watch, as well as the results and the meaning of the performed examinations. One major constrain in testing was that the prototype is not fully ready when this report was written. Some final wrap ups are required to make the device fully functional. These final steps include the detailed integration of communication protocols between MATLAB, the MCU, and the TFT display. Due to this issue, only partially done testing are outlined in this section. The porotype is aimed to be completed for demonstration, during the EDP open-house.

6.1.1 Testing Issues

This device has been designed to detect the errors during the performance of bicep curls and provide feedback to the user. Therefore, the testing has to be done by wearing the wearable and observing the outcome, making sure the accuracy and correctness is offered. Since the prototype was not fully finished, the detailed issues during testing cannot be fully known. Some testing issues that were observed were the disconnection of the wireless communication between MATLAB and the MCU.

6.1.2 Classes of tests

Four sets of performance tests will be done during the testing phase. One test is to observe the outcome, when the watch is worn and a correct set of bicep curls are preformed. If the device returns a “Good Job” response, it passes the test. The second test would be to do an overly rapid set of bicep curls and if the response returns a “Curls Too Fast” response, the device will pass the test. The second test would be to do a set of bicep curls, while tilting the wrist to the side. If the response returns a “Don’t Tilt your wrist” response, the device will pass the test. Lastly, if the derive returns a “Curl Dynamic Range Too Low” response, during a low dynamic range bicep curl set, the device would pass the test.

5.3 Expected Software Response

The expected software response would provide a smooth transmission of IMU data from the MCU to the MATLAB environment, as well as a smoothly executed response, corresponding to the true state of the bicep curls.

6.4 Performance Bounds

Certain instructions will be provided to the user before the device starts working (IMU calibration). First instruction would ask the user to move his/her hand to a certain location. Reaching this point would make this possible for the device to detect a reference point, in which further analysis would be conducted.

The other performance bound would be the measurement of the expected battery life. Table 1 provides the average current consumption of each device.

Table 1. Measured Average Current Consumption of Each Active Component in the Wearable

Device	Current Consumption (mA)
MKR1000 MCU	120
BNO055 IMU	12
TFT Screen + Boost Converter + Arduino Pro Mini	80
Total	212

From the table above the average expected battery life of the wearable can be calculated. Considering a fully charged Li-Po battery, until the voltage drops to its re-charging voltage, the battery has a 2000 mAh of capacity. Equation 4 below illustrates the calculation of the expected battery life.

$$Battery\ Life = \frac{mAh\ Battery\ Capacity}{Current\ Load} = \frac{2000\ mAh}{212\ mA} = 9.43\ hours \qquad (4)$$

Although, the mathematical estimation of the battery life comes down to 9.4 hours, the realistic battery life expectancy is around 5 hours. The reason behind this has to be further investigated.

6.5 Identification of Critical Components

In this section, the critical components that demand the most attention are listed and elaborated upon. The most important components in the testing procedure is he IMU and the processing which is done on the MATLAB, running on the PC. IMU module has to be placed in a firm and tight fitted location, within the watch chassis, to ensure that it will be not loosely move. Moreover, the user has to make sure that there is a Wi-Fi connection in the area and the watch is programmed to connect to that network, as soon as it's turned on.

6.6 Observed Software Response

The observation of the text/image based feedback and the plotting of the inertial data by MATLAB are the group’s expected software response. It should be mentioned that the feedback generated on the PC and the watch were not comparable in the aspects such as response time. The observation on the PC was accurate and fast. The MATLAB could quickly count the repetitions accurately and could detect three types of errors in a quick manner and without any delays. It was responsive to all anticipated errors which were: rapid reps, low dynamic range, and wrist rotation and side tilting. The generated feedback on the MATLAB software was an error window, indicating what the detected error was in that specific instance. The observation of the feedback on the prototype was accurate but not as fast as the PC. Because two microcontrollers and the LCD were meant to be synchronized, the feedback demonstration period, turned out to be longer than anticipated. The device could walk the user through the exercise and the calibration process. The device interface turned out to be very user friendly and easy to understand. The device had the ability to count the number of repetitions, detect the error and demonstrate a visual feedback, indicating how the exercise should have been done.

7.0 Conclusion and Future Work

Sport injuries are some of the causes of referrals to ER room [5]. There are many reasons in which a sport related injury can occur, however an incorrect technique and posture are among the most probable causes [5]. As a result, providing correct instructions to maintain and learn correct techniques during sport activities will result in a fewer of injuries. These correct instructions can also improve the efficiency and effectives of one’s exercises.

Technology has always been a tool which have helped humans to increase their performance level and quality of life. Throughout past few decades, in this specific field, biometric technologies have been utilized to prevent sport injuries. The correct range of motions have been anticipated by biometric sensors and are transmitted to the user, through visual, auditory, and haptic feedbacks. Theses feedbacks are in place to navigate the user to maintain a correct posture and utilize appropriate techniques.

In this project, the team was asked to design and build a wearable biometric device which could provide appropriate feedbacks to the user so he/she may correct his/her posture. This aim was achieved by using an IMU, as the biometric sensor, to detect and correct the movements of the hand in a bicep curl exercise. Based on the defined set threshold for every related IMU output, the device can anticipate as if the exercise has been conducted correctly or the user should be informed about his/her mistake and learn the correct method, through the implemented feedback system. The designed system is in the shape of a smart watch, which contains the IMU and a LCD. Three specific incorrect movements were defined for the device: a low dynamic range, rapid bicep curls and wrist rotation. Based on the defined thresholds for magnetometer output in Y direction, the accelerometer output in the Y-direction and Gyroscope output in the Z-direction, one can correct the mentioned movements respectively. The device has the ability to count the bicep curl repletion of its user. Thereafter, it can inform the user about his/her mistake by providing a visual feedback displayed on the provided LCD, to demonstrate the error.

The device has the ability to detect the errors in real-time and with a decent accuracy. During the trials, within the defined limits, there were no observations indicating that device has missed any errors. Further research can be conducted in this field to expand the capabilities of this device. By the use of machine learning and neural networks, this device can be used to detect any incorrect exercise movements and will not be limited to a single exercise. Further development may go toward a device that can automatically anticipate the nature of the exercise activity and will have the ability to correct the techniques and user's dynamic posture, of that specific sport activity.

Compared to the reviewed state-of-art products, the proposed wearable is much simpler. Unlike the first reviewed paper, this project consists of a single device that is attached to the forearm, as a watch. Also, instead of using a vibrotactile feedback, a visual feedback on a TFT is provided to the user. The second reviewed product covers a variety of exercises and the feedback is auditory and visual. However, the product feedback is machine learning based, making it less straight forward. In rehabilitation, this may not be easily approved, unless the training sets are validated by a professional. The Proposed methodology in this project uses statistics and the fundamental data from the sensors to analyze the movement, therefore it is more reliable. Also, the feedback provided by this project can specifically explain the problem with the motion of the user, while Moov product estimates the possibility of the issue.

8.0 Acknowledgment

Special thanks and appreciation would go to Omar Grant (P.Eng). As the FLC and the supervisor of the project, he was always there for the team members. Another special thanks to Professor. Karthi Umapathy and Professor Kristiina Mai, the BME700/800 instructors and course coordinators. Special thanks to the Department of Electrical Computer Engineering, as well as the Faculty of Engineering and Architectural Science of Ryerson University, for making the Biomedical Engineering program. The last special acknowledgement should go to the members of the group: Seyed-Youns Sadat-Nejad, Faraz Sadrzadeh-Afsharazar, and Mohammadali Ahmadipour for putting their efforts into this project.

9.0 Bibliography

[1] D. Nedelkovski, “How I2C Communication Works & How To Use It with Arduino,” HowTo Mechatronics, 16-May-2016. [Online]. Available: http://howtomechatronics.com /tutorials/arduino/ how-i2c-communication-works-and-how-to-use-it-with-arduino/.

[2] Analog Devices, “3-Axis , ±2 g/±4 g/±8 g/±16 g Digital Accelerometer”, ADXL345 datasheet,2015.

[3] STMicroelectronics, “MEMS motion sensor: ultra-stable three-axis digital output gyroscope”, L3G4200D datasheet, 2010.

[4] MATLAB,” The MathWorks, 2017. [Online]. Available: https://www.mathworks.com

[5] Caine, Dennis, John DiFiori, and Nicola Maffulli. "Physeal injuries in children’s and youth sports: reasons for concern?." *British journal of sports medicine* 40.9 (2006): 749-760.

[6] Edmed, Shannon L., and Karen A. Sullivan. "The influence of injury cause, contact-sport participation, and personal knowledge on expectation of outcome from mild traumatic brain injury." *Journal of clinical and experimental neuropsychology* 36.3 (2014): 221-235.

[7] Jain, Anil K., Arun Ross, and Salil Prabhakar. "An introduction to biometric recognition." *IEEE Transactions on circuits and systems for video technology* 14.1 (2004): 4-20.

[8] Ristolainen, Leena, et al. "Sport injuries as the main cause of sport career termination among Finnish top-level athletes." *European Journal of Sport Science* 12.3 (2012): 274-282.

[9] Schneider, Sven, et al. "Sports injuries among adolescents: Incidence, causes and consequences." Journal of paediatrics and child health 48.10 (2012).

[10] Yoshikawa, T. O. M. O. A. K. I., et al. "The effects of muscle fatigue on bone strain." Journal of Experimental Biology 188.1 (1994): 217-233.

[11]“Arduino-Arduino MKR1000”, Aruino, 2017. [Online]. Available: https://www.arduino.cc/en /Main/ArduinoMKR1000. [Accessed: 10-Apr-2017].

[12]“Moov : running”, Moov website ,2017, [online], Available :https://welcome.moov.cc/ moovnow/

[13] S.W.Smith, “Moving Average Filters”, in *The Scientist and Engineer`s Guide to Digital Signal Processing*. Sandiego, California. California Technical Publishing, 1997, Chapter 15, pp.277-284.

[14] M. Rouse, “What is calibration? - Definition from WhatIs.com,” WhatIs.com, 2017. [Online]. Available: http://whatis.techtarget.com/definition/calibration. [Accessed: 01-Apr-2017].

[15]“xioTechnologies/Gait-Tracking-With-x-IMU,” GitHub, 10-Dec-2013. [Online]. Available: https://github.com/xioTechnologies/Gait-Tracking-With-x-IMU. [Accessed: 03-Apr-2017].

[16] Leonardo , “Bitbucket,” Leos_upm / Matlab: Real time 3D visualization of MPU6050 accelerometer and gyro - Matlab script / source / gy_521_send_serial.ino — Bitbucket, Feb-2016. [Online]. Available: https://bitbucket.org/Leos_upm/matlab-real-time-3d-visualization-of-mpu6050-accelerometer-and/src/3e794ff01ffb49eaae24fc2b5d4a907e0b9f9daa/gy_521_send_serial.ino?at=master&fileviewer=file-view-default. [Accessed: 04-Apr-2017].

[17] K. Murphy, “Kalman filter toolbox for Matlab,” Kalman filter toolbox for Matlab, 1998. [Online]. Available: http://www.cs.ubc.ca/~murphyk/Software/Kalman/kalman.html. [Accessed: 01-Apr-2017].

[18] T. Pycke, “MAV-blog,” MAV-blog : Kalman filtering of IMU data, 22-May-2006. [Online]. Available: http://tom.pycke.be/mav/71/kalman-filtering-of-imu-data. [Accessed: 01-Apr-2017].

[19] P. Riseborough , “Extended Kalman Filter Attitude, Velocity and Position Estimator,” GitHub, 31-May-2016. [Online]. Available: https://github.com/PX4/ecl/wiki/Extended-Kalman-Filter---Attitude,-Velocity-and-Position-Estimator. [Accessed: 01-Apr-2017].

[20] A. Causo, L. D. Tran, S. H. Yeo and I. M. Chen, "Vibrotactile motors on stationary arm as directional feedback to correct arm posture," *2012 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, Kachsiung, 2012, pp. 202-207.

[21] Arduino LLC. Arduino/Genuino MKR1000 Product Description.[online]. 2016 [cit. 2017-04-11]. Retrieved from:: https://www.arduino.cc/en/uploads/Main/MKR1000-schematic.pdf

[22] Atmel: Datasheet. : ATSAMW25-MR210PB [online]. 2016 [cit. 2017-04-11]. Retrieved from: http://www.mouser.com/ds/2/268/atmel-42395-smartconnect-atsamw25-mr210pa_datashee-1065560.pdf

[23] Datasheet – BNO055. *Bosch Sensortec.* 2014. Available: https://cdn-shop.adafruit.com/datasheets/ BST_BNO055_DS000_12.pdf

[24] C. Woodford, “How accelerometers work | Types of accelerometers,” Explain that Stuff, 08-Jun-2016. [Online]. Available: http://www.explainthatstuff.com/accelerometers.html. [Accessed: 13-Apr-2017].

[25] “ABOUT INNOVENTIONS®,” Magnetometer in Smartphones and Tablets, 2012. [Online]. Available: http://www.rotoview.com/magnetometer.htm. [Accessed: 03-Apr-2017]

[26] D. Nedelkovski, “MEMS Accelerometer Gyroscope Magnetometer & Arduino,” HowToMechatronics, 16-May-2016. [Online]. Available: http://howtomechatronics.com/how-it-works/electrical-engineering/mems-accelerometer-gyrocope-magnetometer-arduino/. [Accessed: 11-Apr-2017].

[27] N. Venkatesh , “UDP Untangled - Overview of how UDP works,” 2buntu · News for Human Beings, 13-Sep-2012. [Online]. Available: https://2buntu.com/articles/1209/udp-untangled-overview-of-how-udp-works/. [Accessed: 11-Apr-2017].

[28] Datasheet : Adafruit 2.8" TFT [online].2016 [cit.2017-04-22]. Retrieved from: https://www.arduino.cc/en/uploads/Main/HTF0177SN-01-SPEC.pdf

[29] Datasheet- RB-Pol-221 Step-Up / Step-Down Voltage Regulator S7V8A [online].2017. [cit. 2017-04-11]. Retrieved from: http://www.robotshop.com/media/files/PDF/datasheet-2118.pdf

[30] Datasheet- GY-801 BMP180 9-axis magnetic acceleration gyroscope atmospheric pressure module [online].2016 [cit. 2017-04-11]. Retrieved from: http://www.alldatasheet.com/view.jsp?Searchword=GY801

[31] Datasheet: Arduino Pro Mini 3.3V a [online].2017 [cit. 2017-04-11]. Retrieved from: https://learn.sparkfun.com/tutorials/using-the-arduino-pro-mini-33v/all.pdf

10.0 Appendix

MAIN ARDUINO CODE:

```
#include <Wire.h>
#include <SPI.h>
#include <WiFi101.h>
#include <WiFiUdp.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>

//Sampling Rate
#define BNO055_SAMPLERATE_DELAY_MS (100)
Adafruit_BNO055 bno = Adafruit_BNO055();


char buf1[100], buf2[100], buf3[100], buf4[100], buf5[100], buf6[100], buf7[100], buf8[100],
buf9[100], buf10[100];

int status = WL_IDLE_STATUS;
char ssid[] = "faraz"; //  your network SSID (name)
char pass[] = "11111111";    // your network password (use for WPA, or use as key for WEP)
int keyIndex = 0;            // your network key Index number (needed only for WEP)

unsigned int localPort = 2390;      // local port to listen on
char packetBuffer[255]; //buffer to hold incoming packet
WiFiUDP Udp;


void setup()
{

  Serial.begin(9600);
  if(!bno.begin())
  {
    /* There was a problem detecting the BNO055 ... check your connections */
    Serial.print("Ooops, no BNO055 detected ... Check your wiring or I2C ADDR!");
    while(1);
  }

  delay(1000);

  bno.setExtCrystalUse(true);

  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    // don't continue:
    while (true);
  }

  // attempt to connect to WiFi network:
  while ( status != WL_CONNECTED) {
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
    status = WiFi.begin(ssid, pass);

    // wait 10 seconds for connection:
    delay(10000);
  }
  Serial.println("Connected to wifi");
  printWiFiStatus();

  Serial.println("\nStarting connection to server...");
  // if you get a connection, report back via serial:
  Udp.begin(localPort);

  pinMode(6, OUTPUT);
  pinMode(0, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(0), ISR, FALLING);
}


void loop()
{

  digitalWrite(6, LOW);
  int packetSize = Udp.parsePacket();
  if (packetSize)
  {
    // Possible vector values can be:
  // - VECTOR_ACCELEROMETER - m/s^2
  // - VECTOR_MAGNETOMETER  - uT
  // - VECTOR_GYROSCOPE     - rad/s
  // - VECTOR_EULER         - degrees
  // - VECTOR_LINEARACCEL   - m/s^2
  // - VECTOR_GRAVITY       - m/s^2
    // read the packet into packetBufffer
    int len = Udp.read(packetBuffer, 255);
   if (len > 0) packetBuffer[len] = 0;
   Serial.println(packetBuffer);
   digitalWrite(6, HIGH);
   if (packetBuffer) wrf();
   }
}

void printWiFiStatus() {


  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}

void ISR ()
{
  uint8_t sys, gyro, accel, mag = 0;
  bno.getCalibration(&sys, &gyro, &accel, &mag);
  sprintf(buf7, "%d ", sys); sprintf(buf8, "%d ", gyro); sprintf(buf9, "%d ", accel); sprintf(buf10,
"%d ", mag);
  Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
    Udp.write("System: ");
    Udp.write(buf7);
    Udp.write(" Accelerometer: ");
    Udp.write(buf9);
    Udp.write(" Gyroscope: ");
    Udp.write(buf8);
    Udp.write(" Magnetometer: ");
    Udp.write(buf10);
    Udp.endPacket();
}

void wrf ()
{
  while (1) {
```

```
 imu::Vector<3> lin = bno.getVector(Adafruit_BNO055::VECTOR_LINEARACCEL);
imu::Vector<3> eu = bno.getVector(Adafruit_BNO055::VECTOR_EULER);
 sprintf(buf1, "%.2f ", lin.x()); sprintf(buf2, "%.2f ", lin.y()); sprintf(buf3, "%.2f ", lin.z());
sprintf(buf4, "%.2f ", eu.x()); sprintf(buf5, "%.2f ", eu.y()); sprintf(buf6, "%.2f ", eu.z());
  // send a reply, to the IP address and port that sent us the packet we received
    Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
    Udp.write(buf1);
    Udp.write(buf2);
    Udp.write(buf3);
    Udp.write(buf4);
    Udp.write(buf5);
    Udp.write(buf6);
    Udp.endPacket();
    int packetSize = Udp.parsePacket();
    if (packetSize)
  {
    int len = Udp.read(packetBuffer, 255);
    if (len > 0) packetBuffer[len] = 0;
    if (packetBuffer)
    break;
   }
    delay(20);
}

}
```

PRO MINI ARDUINO CODE:

```
#include <SPI.h>
#include <SD.h>
#include <TFT.h>
#define sd_cs  9
#define lcd_cs 10
#define dc     8
#define rst    7

TFT TFTscreen = TFT(lcd_cs, dc, rst);
char inByte = 0;

void setup() {
  TFTscreen.begin();
  TFTscreen.background(255, 255, 255);
  TFTscreen.stroke(0, 0, 255);
  TFTscreen.stroke(0, 0, 0);
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }
  SPI.setClockDivider(SPI_CLOCK_DIV4);
  TFTscreen.background(255, 255, 255);
  Serial.print(F("Initializing SD card..."));
  if (!SD.begin(sd_cs)) {
    Serial.println(F("failed!"));
    return;
  }

  TFTscreen.begin();

  pinMode(3, INPUT_PULLUP);
  pinMode(4, INPUT_PULLUP);
}

void loop() {
welcome();
pressany ();
calibration ();
initialize ();
curl ();
delay(5000);
reps();
feedback();
delay(1000);
}
```

```
void feedback ()
{
  PImage logo1;
  delay(1000);
  logo1 = TFTscreen.loadImage("1.bmp");
  delay(1000);
  TFTscreen.image(logo1, 0, 0);
  delay(1000);
  logo1 = TFTscreen.loadImage("2.bmp");
  delay(1000);
  TFTscreen.image(logo1, 0, 0);
  delay(1000);
  logo1 = TFTscreen.loadImage("3.bmp");
  delay(1000);
  TFTscreen.image(logo1, 0, 0);
  delay(1000);
  logo1 = TFTscreen.loadImage("4.bmp");
  delay(1000);
  TFTscreen.image(logo1, 0, 0);
  delay(10);
}

void fadein ()
  {
  int i = 0;
  while (i < 256) {
  TFTscreen.setTextSize(6);
  TFTscreen.stroke(i, i, i);
  TFTscreen.text("FYM", 30, 10);
  TFTscreen.setTextSize(2);
  TFTscreen.text("WATCH", 50, 70);
  i = i + 8;
  delay (30);
    }
  i = 0;
    while (i < 256) {
  TFTscreen.setTextSize(6);
  TFTscreen.stroke(255-i, 255-i, 255-i);
  TFTscreen.text("FYM", 30, 10);
  TFTscreen.setTextSize(2);
  TFTscreen.text("WATCH", 50, 70);
  i = i + 8;
  delay(30);
    }
  }

void welcome ()
{
  TFTscreen.background(0, 0, 0);
  fadein();
  delay(2000);
  TFTscreen.background(0, 0, 0);
  TFTscreen.setTextSize(3);
  TFTscreen.stroke(255, 255, 255);
  TFTscreen.text("Welcome", 0, 20);
  delay(100);
  TFTscreen.setTextSize(1);
  TFTscreen.text("FYM Watch monitors the", 0, 60);
  TFTscreen.text("correctness of your", 0, 80);
  TFTscreen.text("Bicep Curls", 0, 100);
  delay(3000);
}

void pressany ()
{
  TFTscreen.background(0, 0, 0);
  TFTscreen.setTextSize(3);
  TFTscreen.stroke(255, 255, 255);
  TFTscreen.text("Press Any", 0, 0);
  TFTscreen.text("Key", 0, 30);
  TFTscreen.setTextSize(1);
  TFTscreen.text("To Continue", 0, 60);
  wait();
}

void initialize ()
{
  int i;
  TFTscreen.background(0, 0, 0);
  TFTscreen.setTextSize(1);
  TFTscreen.stroke(255, 255, 255);
  TFTscreen.text("WiFi Connected", 0, 0);
  delay(500);
  TFTscreen.background(0, 0, 0);
  for (i=0; i<2; i++) {
  TFTscreen.text("IMU Ready.", 0, 60);
  delay(200);
  TFTscreen.text("IMU Ready..", 0, 60);
  delay(200);
  TFTscreen.text("IMU Ready...", 0, 60);
  delay(200);
  TFTscreen.background(0, 0, 0);
  }
  TFTscreen.background(0, 0, 0);
  TFTscreen.setTextSize(1);
  TFTscreen.text("Press Anything", 0, 0);
  delay(200);
  TFTscreen.text("and Start Curling", 0, 40);
  wait();
}

void calibration ()
{
  TFTscreen.background(0, 0, 0);
  TFTscreen.setTextSize(2);
  TFTscreen.stroke(255, 255, 255);
  TFTscreen.text("CALIBRATION", 0, 0);
  delay(500);
  TFTscreen.background(0, 0, 0);
  TFTscreen.setTextSize(1);
  TFTscreen.text("Gyro Calibration", 0, 0);
  TFTscreen.text("Leave the Watch Motionless", 0, 30);
  wait();
  TFTscreen.background(0, 0, 0);
  TFTscreen.setTextSize(4);
  TFTscreen.text("Good!", 25, 50);
  delay(500);
  TFTscreen.background(0, 0, 0);
  TFTscreen.setTextSize(1);
  TFTscreen.text("Accelerometer Calibration", 0, 0);
  TFTscreen.text("Rotate the Watch at ", 0, 30);
  TFTscreen.text("90 Degree Increments ", 0, 60);
  wait();
  TFTscreen.background(0, 0, 0);
  TFTscreen.setTextSize(4);
  TFTscreen.text("Good!", 25, 50);
  delay(500);
  TFTscreen.background(0, 0, 0);
  TFTscreen.setTextSize(1);
  TFTscreen.text("Magnetometer Calibration", 0, 0);
  TFTscreen.text("Wave Randomly", 0, 30);
  wait();
  TFTscreen.background(0, 0, 0);
  TFTscreen.setTextSize(4);
  TFTscreen.text("Good!", 25, 50);
  delay(500);
  TFTscreen.background(0, 0, 0);
}

void curl ()
{
  TFTscreen.background(0, 0, 0);
  TFTscreen.setTextSize(3);
  TFTscreen.stroke(255, 255, 255);
  TFTscreen.text("Do Curls", 0, 0);
  TFTscreen.setTextSize(1);
  TFTscreen.text("Press Anything, When Done!", 0, 50);
  wait();
}

void wait ()
{
  while (digitalRead(4) && digitalRead(3)) {;}
}

void reps ()
{
  int rep = 0;
  char out[4];
  TFTscreen.background(0, 0, 0);
  TFTscreen.setTextSize(3);
  TFTscreen.stroke(255, 255, 255);
  TFTscreen.text("Reps:", 0, 0);
  while (1)
  {
    if (digitalRead(3) == LOW) break;
    if (digitalRead(4) == LOW)
    {
    rep++;
    String r = String(rep);
    r.toCharArray(out, 4);
    TFTscreen.background(0, 0, 0);
    TFTscreen.stroke(255, 255, 255);
    TFTscreen.text("Reps:", 0, 0);
    TFTscreen.text(out, 0, 50);
    }
  }
  TFTscreen.stroke(0, 0, 0);
  TFTscreen.text("Reps:", 0, 0);
  TFTscreen.stroke(255, 255, 255);
  TFTscreen.text("You Did:", 0, 0);
  wait();
  TFTscreen.background(0, 0, 0);
  TFTscreen.setTextSize(1);
  rep = 0;
  while (1)
  {
    if (digitalRead(3) == LOW)
      if (rep == 1)
      {TFTscreen.text("Biecp Curls are too Fast!", 0, 0);
      wait(); delay(2000); break;}
      else if (rep == 2)
      {TFTscreen.text("Curl Dynamic Range Too Small!", 0, 0);
      wait(); delay(2000); break;}
      else if (rep == 3)
      {TFTscreen.text("Don't Tilt Your Wrist too Much!", 0, 0);
      wait(); delay(2000); break;}
      else
      {TFTscreen.text("Good Job!", 0, 0);
      wait(); delay(2000); break;}
    if (digitalRead(4) == LOW)
    {rep++; delay(200);}
  } }
```

MATLAB CODE:

```
close all;                    % close all figures
clear;                       % clear all variables
clc;

number = zeros(100, 15);
aX=zeros(100,1); aY=zeros(100,1); aZ=zeros(100,1); gX=zeros(100,1);  gY=zeros(100,1);
gZ=zeros(100,1); xAccFilt=zeros(100,1); yAccFilt=zeros(100,1); zAccFilt=zeros(100,1);
AccMag=zeros(100,1); arr=zeros(100,1); Gytrain=zeros(100,1);
repcount=0;
xVel=zeros(100,1); yVel=zeros(100,1);zVel=zeros(100,1);
xPose=zeros(100,1);yPose=zeros(100,1);zPose=zeros(100,1);

T=zeros(1,1);
tic;
dataAll=zeros(200,15);
j=0;
count= 0;
format long
k=0;
initial=zeros(3,3);
index=1:100;
tabs=3;
while 1
i=1;
```

```
u=udp('192.168.1.101', 2390,'LocalPort', 2391);
fopen(u);
fwrite(u, 1:10); %Get information from MCU
fwrite(u, 1:10); %Stop getting infromation from MCU
l= fread(u, 32);
word = transpose(char(l));
number(i,:) = str2num(word); %All data are saved into number matrix
fclose(u);
pause(0.06); %delay
%% Divide matrix of data to vectors of data with signale sensor-direction
xAcc=number(i, 1);
yAcc=number(i, 2);
zAcc=number(i, 3);
xGyro=number(i, 4);
yGyro=number(i, 5);
zGyro=number(i, 6);
xMagno=number(i, 7);
yMagno=number(i, 8);
zMagno=number(i, 9);
xGrav=number(i, 10);
yGrav=number(i, 11);
zGrav=number(i, 12);
xLNAcc=number(i, 13);
yLNAcc=number(i, 14);
zLNAcc=number(i, 15);

%% Simple Moving Average - Filter Accelometer
aX=[aX(2:end);xAcc];
xAccFilt=[xAccFilt(2:end);...
   mean(aX(100:-1:101-tabs))];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
aY=[aY(2:end);yAcc];
yAccFilt=[yAccFilt(2:end);...
   mean(aY(100:-1:101-tabs))];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
aZ=[aZ(2:end);yAcc];
zAccFilt=[zAccFilt(2:end);...
   mean(aZ(100:-1:101-tabs))];
%% Rappid Error
%plot
if yLNAcc>15

msgbox('Too Rapid','Error','error');
end
```

```
%% Rep Count
Gytrain=[Gytrain(2:end);yMagno]; %Gytrain sets off all Y-Magno
[L, numberOfPeaks] = bwlabel(diff(Gytrain)>30);
 repcounts=numberOfPeaks

%% Error in Dynamics

Gsum=(Gytrain(end-10:end)); %Creating windos of 10
if ne(Gsum,0)
maxW=max(Gytrain(end-10:end)); %Maximum value in the window
minW=min(min(Gytrain(end-10:end))); %Minimum value in the window
pause(0.02);
if maxW+abs(minW) < 60
    k=k+1;
    if k==1 || k==6 || k==12 || k==18 || k==24 || k==30 || k==36
        msgbox('Increase The Dynamic Range of Your Exercise','Error','error');

    end
end
end



%% Arm Rotation Error
if (zGyro)>3
%     display('You are turning your fist too much')
    msgbox('You are turning your fist too much','Error','error');

end



%% Plot

%% Square Plot
% limits=200;
% cla;
% line([0 DegthetaY],[0 0],'color', 'r' ,'LineWidth',2,'Marker','o');
% axis([-limits limits -limits limits]);
% axis square;
% grid on
% drawnow;
%% Visualize Data On Sphere Or any Other Objects
%     [x,y,z] = sphere;h = surf(x,y,z);axis('square');
%     title('AOL TEAM')
%     xlabel('x'); ylabel('y'); zlabel('z');
%     %Rotate Object
%     rotate(h,[1,0,0],DegthetaZ)
%     rotate(h,[0,1,0],DegthetaY)
%     rotate(h,[0,0,1],DegthetaX)
%     view(0,0);
%     drawnow
%% PLOT VS TIME
% index=1:100;
% aX=[aX(2:end);xAcc]; %shift data and add most recent reading
% aY=[aY(2:end);yAcc]; %shift data and add most recent reading
% aZ=[aZ(2:end);zAcc]; %shift data and add most recent reading
% gX=[gX(2:end);xGyro]; %shift data and add most recent reading
% gY=[gY(2:end);yGyro]; %shift data and add most recent reading
% gZ=[gZ(2:end);zGyro]; %shift data and add most recent reading

% % plot(index,(aX))
% plot(gX)
% drawnow;

%% 2D plot Orientation
%
% theta=yAng;
%
%
%%%%%%%% 2D object %%%%%%%
% th= -pi*theta/180;
% xsn=[cos(th) -sin(th); sin(th) cos(th)]*xxxxx;
% figure(1)
% xx=fill(xsn(1,:),xsn(2,:), 'r');
% axis([-2, 2, -2, 2])
% drawnow;
%
% sin(th)
% %%  Observe Data
j=j+1;
i=i+1;


end
```