REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DEVELOPING A HAPTIC HAND INTERFACE WITH FORCE FEEDBACK FROM VIRTUAL ENVIRONMENT AND CONTROLLING IT VIA EMG SIGNALS

MIRVAHID AHMADIPOURI NAEIM

MSc. THESIS
DEPARTMENT OF MECHATRONIC ENGINEERING
PROGRAM OF MECHATRONIC ENGINEERING

ADVISER
ASST. PROF. DR. CÜNEYT YILMAZ

İSTANBUL, 2018

REPUBLIC OF TURKEY

YILDIZ TECHNICAL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DEVELOPING A HAPTIC HAND INTERFACE WITH FORCE FEEDBACK FROM VIRTUAL ENVIRONMENT AND CONTROLLING IT VIA EMG SIGNALS

A thesis submitted by Mir Vahid AHMADIPOURI NAEIM in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE** is approved by the committee on 21.06.2018 in Department of Mechatronic Engineering, Mechatronic Engineering Program.

Thesis Adviser

Asst. Prof. Dr. Cüneyt YILMAZ

Yıldız Technical University

Approved By the Examining Committee

Asst. Prof. Dr. Cüneyt YILMAZ

Yıldız Technical University _____

Prof. Dr. Vasfi Emre Ömürlü, Member

Yıldız Technical University _____

Prof. Dr. Hikmet Kocabaş, Member

Istanbul Technical University _____

This study was supported by the Scientific Research Project Coordination of Yildiz Technical University (BAP) Grant No: 2016-06-04-YL01 (1057).

ACKNOWLEDGEMENTS

First of all, I would like to thank my thesis advisor Asst. Prof. Dr. Cüneyt YILMAZ of the Department of the Mechatronic Engineering at the Yıldız Technical University for his patience. He consistently allowed this paper to be my own work, but steered me in the right direction whenever he thought I needed it.

I would also like to thank all the academic members who helped me to join the master program.

I would also like to special thanks to Op. Dr. Yakup AVŞAR at Avşar Polyclinic for valuable supports to 3d print the mechanical parts.

March, 2018

Mir Vahid AHMADIPOUR

TABLE OF CONTENTS

Page

LIST OF SYMBOLS

Ω	Ohms
A	Amperes
S	Seconds
V	Volts
mA	Milliamps

LIST OF ABBREVIATIONS

ADC	Analog to Digital Conversion
DH	Denavit–Hartenberg
DOF	Degree of Freedom
EEG	Electrogastrography
EMG	Electromyography
IMU	Inertial Measurement Unit
sEMG	surface Electromyography
SMA	Sampling Moving Average
SMA	Simple Moving Average
SNR	Signal to Noise Ratio
VE	Virtual Environment
VR	Virtual Reality

LIST OF FIGURES

Page

LIST OF TABLES

Page

ABSTRACT

DEVELOPING A HAPTIC HAND INTERFACE WITH FORCE FEEDBACK FROM VIRTUAL ENVIRONMENT AND CONTROLLING IT VIA EMG SIGNALS

Mirvahid AHMADIPOURI NAEIM

Department of Mechatronics Engineering

MSc. Thesis

Adviser: Asst. Prof. Dr. Cüneyt YILMAZ

A multi-functional human-machine interface was designed to assist fingers flexion and extension motions. The proposed exoskeleton system may help and improve the rehabilitation progress of hand fingers. This system was designed with 2 degrees of freedom (DOF) for the thumb finger and 3 DOF for the other fingers actuated via direct-driven linear DC actuators, which had PID position and force controllers. An electromyogram (EMG) armband with a built-in gyroscope and accelerometer unit supplies EMG and hand orientation signals to synchronize the exoskeleton hand and fingers with the ones designed in a three-dimensional virtual environment. In addition, this exoskeleton system has also a novel force sensing mechanism designed to sense fingertip forces and also help the motors freely follow human finger movements with a minimal resistance in its virtual reality applications. A contact detection method was developed and applied to sense the 3 dimensional (3D) objects in the virtual environment. The studies showed that the position and force feedback controllers suitably worked for both sensing the 3D virtual objects and for the mechanism's following the user fingers' movements with minimal resistance. In conclusion, such a mechanism with proposed special capabilities could be used to be utilized for finger rehabilitation and also haptic training applications.

Keywords: Virtual Rehabilitation, Virtual Reality, Exoskeleton Device, Electromyography Feedback

ÖZET

SANAL ORTAMDAN KUVVET GERİ BESLEME İLE BİR HAPTİK EL ARAYÜZÜ GELİŞTİRİLMESİ VE EMG SİNYALLERİ İLE EL PROTEZİ KONTROLÜ

Mirvahid AHMADIPOURINAEIM

Mekatronik Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Tez Danışmanı: Dr.Öğr.Üyesi Cüneyt YILMAZ

Parmaklara fleksiyon ve ekstensiyon hareketlerine yardımcı olmak için çok fonksiyonlu bir arayüz tasarlandı. Önerilen exoskeleton sistemi el parmaklarının rehabilitasyon ilerlemesine yardımcı olabilir. Bu sistem, baş parmak için 2 serbestlik derecesi (DOF) ve diğer parmaklar için 3 DOF ile PID konumu ve kuvvet kontrolörleri olan doğrudan tahrikli doğrusal DC aktüatörler ile çalıştırılan tasarlanmıştır. Sanal ortamda tasarlanan el ile senkronize olmak için bir elektromiyogram (EMG) kol bantı ve yerleşik bir jiroskop ve ivme ölçer üniteleri, el yönlendirme sinyalleri sağlar. Ayrıca, bu dış iskelet sistemi, parmak ucu kuvvetlerini algılamak için ve aynı zamanda sanal gerçeklik uygulamalarında küçük bir dirençle motorların insan parmak hareketlerini serbest takip etmesine yardımcı olan yeni bir kuvvet algılama mekanizmasına da sahiptir.Sanal ortamda 3 boyutlu (3D) nesneleri algılamak için bir tespit yöntemi geliştirildi ve uygulandı. Bazı sanal gerçeklik uygulamaları için pozisyon ve kuvvet verileri elde etmek için, önerilen bir dış iskelet mekanizmasının normal bir kişi üzerindeki temel deneysel çalışmaları yapılmıştır. Mekanik ve sanal sistem arasındaki gecikme, kullanıcı tarafından önemli ölçüde hissedilmeyen 13 milisaniye olarak belirlendi. Çalışmalar, konum ve kuvvet geri bildirimi denetleyicilerinin, hem 3D sanal nesneleri algılamak için hem de mekanizmanın kullanıcı parmaklarının hareketlerini az dirençle takip etmesi için uygun şekilde çalıştığını gösterdi.Sonuç olarak, önerilen özel yeteneklere sahip bir mekanizma, parmak rehabilitasyonu ve haptik eğitim uygulamaları için kullanılabilir.

Anahtar Kelimeler: Sanal Rehabilitasyon, Sanal Gerçeklik, Dış iskelet Cihazı, Elektromiyografi

CHAPTER 1

INTRODUCTION

1.1 Literature Review

What is the Bio-mechatronic?! From the beginning of the first wheel invention, technology has passed long way at the time. The technology passed different eras from Hunting to agrarian and then Industrial era. Industrial era itself can be categorized into five different levels. Nowadays, by the means of the robotic developments, bio-mechatronic has an important effect on the Industry 4.0 framework [1]. Exoskeletons are one of the examples of bio-mechatronic enhancements. Exoskeletons are wearable robots which have the capability to collaborate with human body actions and machine by translating them to each other. In bio-mechatronics, enhancing a user to exoskeletons can develop and support motor functions of his / her muscles. Exoskeletons should be compliant with the user's movements and sense at least part of the motor force necessary to accomplish the movements. As a result, it can help a user to have better performance [2, 3] or even open new opportunity to develop robots which can be useful for patient rehabilitation [4]. On the other hand, developing a virtual environment interacting with a user, can make new opportunity to simulate a specific environment [5] such as training [6] or clinical rehabilitation proposes [7]. Furthermore, virtual environment can pair with an exoskeleton to get haptic feedback which has the capability to operate robotic hand from the distance [8]. General sections of a Bio-mechatronic system are shown in Figure 1.1. Following paragraphs will talk about the previous researches have been done about bio-mechatronics. Based on our main field of interest, most of the collected kind of literature is focused on hand exoskeleton, especially for rehabilitation applications. The reason for this interest will be described in next sections.

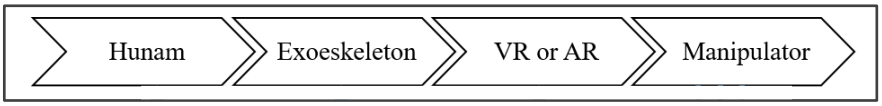


Figure 1.1 General sections of a Cyber-physical system

Based on our main interest on some of the researchers focused on design concept and control system of a biomechatronic system for rehabilitation propose. Instance, Physiotherabot/WF [9] which is a 3 Degree of freedom (DOF) upper limb rehabilitation robot that can perform the pronation/supination movement for a forearm, abduction/adduction and flexion-extension movements for the wrist. It can perform passive, active assisted, isotonic, isometric and isokinetic exercises. Some attempts focused on design VR environment to be able to integrate with an exoskeleton, MHaptic [10] presents a library for bimanual force feedback haptic interaction within generic C++ based on the virtual environments.

The methods which mentioned above are items for a general concept to complete a biomechatronic system. RML glove [11] is a good example would integrate all sections together to design haptic interface cable transmission system that fits on a hand and provides haptic force feedback to each finger maximum 12N of the hand without constraining their movement. A portable haptic master hand developed [12], utilizes a mechanical tape brake at the rolling-link mechanism for passive force feedback which allows fast natural finger movements which connected to a C++ based VR. The virtual world consists of a simple finger model with 4 DOFs and a straight wall and circular object for touching. In other designs, the mechanical mechanism may consist of individual resistive forces to the extension/flexion of the fingers through serial kinematics chains attached to the distal phalanges of the fingers [13] using under-actuated hand exoskeleton. The exoskeleton has been integrated into a VR hand environment where a virtual representation of the users' hand while grasping and manipulating virtual objects, is displayed. Exoskeleton finger with three points of attachment to the operator's finger using tendon transmission mechanism that all together can be worn as a glove by the user [14]. Feedback controllers and haptic interface for Virtual Environments or Tele-manipulation are designed for step input force of varying in a range between 100mN and 500mN. Using similar VR environment are common in the most of the researchers. But other mechanisms are quite different. For example, Glove Prototype with steel leaf springs, finger tension cords, bending sensor, hand plate, tension-cord stops, metal wrist torque transmission, wrist tension cords, and forearm cuff designed [15]. The main goal

is to demonstrate design and development steps involved in a complex intervention while examining the feasibility of using an instrumented orthotic device for home-based rehabilitation after stroke. Other researchers although improve 7- DOF arm exoskeleton, controlling a VR hand for an astronaut in microgravity, supporting the telemanipulation with force feedback of anthropomorphic robotic arms [16].

One of the most important subjects of the biomechatronic is to sense signals from the limb which connected to a cyber system. There are Electromyography (EMG) is frequently used to monitor the motor power of the special muscles [17-19]. Electrogastrography (EEG) is another method to sense magnetic field around the brain to interact with an exoskeleton or VR [20, 21]. Collected data from these biosensors are mostly filtered and interpret by Analog to Digital Conversion (ADC) [22] or Artificial intelligence methods [23]. Image processing methods although used for the kinematical mapping [24, 25].

The result of a review of the mentioned works of the literature presents a big research assay to build complete passive and active human-machine interface, especially for rehabilitation aims. Following, we will explain the general concept of our project and the reasons.

1.2 Objective of the Thesis

Hand exoskeleton that helps the user to perform movements in real, can be more effective to reinstate neuroplasticity and improves motor functions when compared with conventional physiography training. According to evidences, robotic therapy combined with traditional therapy programs can enhance functional motor learning [26-28]. On the other hand, haptic virtual enrolments would make a good procedure simulator, especially for surgery.

Our device could also be applied in the area of rehabilitation of stroke or spinal cord injury patients. It could reduce the number of therapists needed by allowing even the most impaired patient to be trained by one therapist, whereas several are currently needed. Also, training could be more uniform, easier to analyze retrospectively and specifically customized for each patient. Moreover, an exoskeleton can be worn like a glove through the arm. Nowadays, in the rehabilitation of stroke or spinal cord injury patients, it could be used as an active manual orthosis.

Implementation of the haptic feedback into robotic surgical systems can transform the physician's user experience by enabling identification of different tissue structures, preventing tissue damage, ensuring correct suture placement and decreasing task completion time. In the medical field, it can be used for enhanced precision during surgery. Implementation of this device into medical devices can benefit practitioners, patients, and device manufacturers. For a medical practitioner, using instrumentation and hand tools with haptics helps improve confidence by providing immediate feedback via haptics alerts and force feedback to help guide the procedure, ultimately reducing errors by reinstating the sense of touch.

1.3 Hypothesis

All mentioned reasons claim that achieving the biomechatronic technology must lead to start to research more detailed subjective that included in a biomechatronic system. Moreover, as shown in Figure 1.2 developing our project as a haptic hand interface with force feedback and virtual environment would be part of full upper limb exoskeleton including joints of the arm for the future studies.

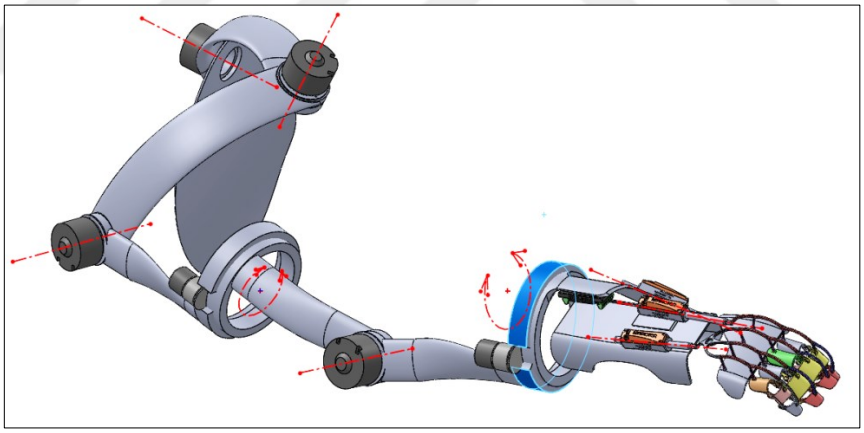


Figure 1.2 Our suggested upper limb exoskeleton concept for future studies with our current project would be a part of it

Our project is a multifunctional exoskeleton which includes all necessary sections of a complete biomechatronic system with all sections shown in Figure 1.2. A novel bar-linkage mechanical mechanism will be designed to perform most of the finger functions with three direct drive DC motors. Details will be described in section 2.1 of chapter 2. The mechanical exoskeleton will work both in active and passive modes depending on applied force form Real hand or collision in VR environment.

Using EMG for the arm muscles, and Force sensors for three of fingers and although sensors to detect orientation and accelerations of the arm hand will help us to interact VR hand with a real hand.

One virtual hand in a configurable 3D environment will be designed. The VR hand will be able to move around. This makes capability for easy software development both in VR and Augmented Reality (AR) for future researchers. The VR fingers will flex and extent by analyzing EMG signals and applied forces to exoskeleton by an active hand. EMG sensors can recognize hand movements although from other person or left hand to the stroke hand which is unable to move fingers. Moreover, it has the capability to connect to a tele-controllable robot which can mimic the movements of the human hand.

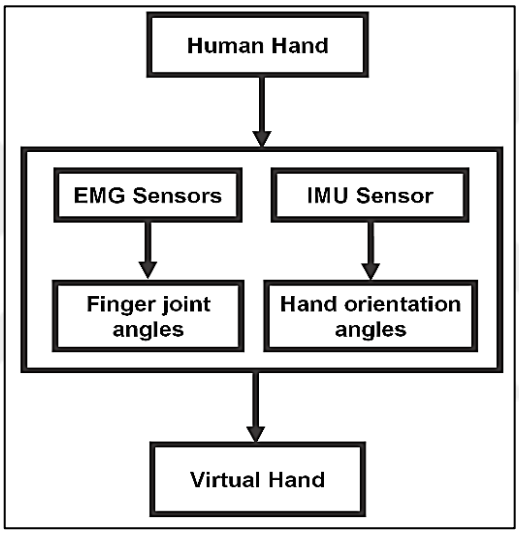


Figure 1.3 General overview of our concept

Our project can be used as a portable rehabilitation device for people who have the problem with flexion and extension in their own daily life. On the other hand, the main function of a powered exoskeleton is to assist the wearer in boosting their strength and endurance. It could also be applied in the area of rehabilitation of stroke or spinal cord injury patients.

This master thesis specifically describes the mechanical, electrical, programming and control design of the exoskeleton. Moreover, it presents an experimental trial with a healthy volunteer and also the data characterizing the different types of control implemented for the device. This document organized into four chapters. Chapter 1 presents an introduction and literature review in the field of exoskeletons for rehabilitation and understanding the framework of this thesis. Chapter 2 describes all the technical and mathematical aspects of the exoskeleton as shown in Figure 1.4. This chapter shows

Mechanical designs which contain Right Hand Exoskeleton and three dc motors, software programming includes Virtual Environment and C# scripts although hardware including Arduino controller, dc motor drivers, Force Sensors, and Bio-sensors. Chapter 3 describes the experiments and the results. Finally, Chapter 4 presents a conclusion and the future work planned for the use of the exoskeleton in the rehabilitation field.

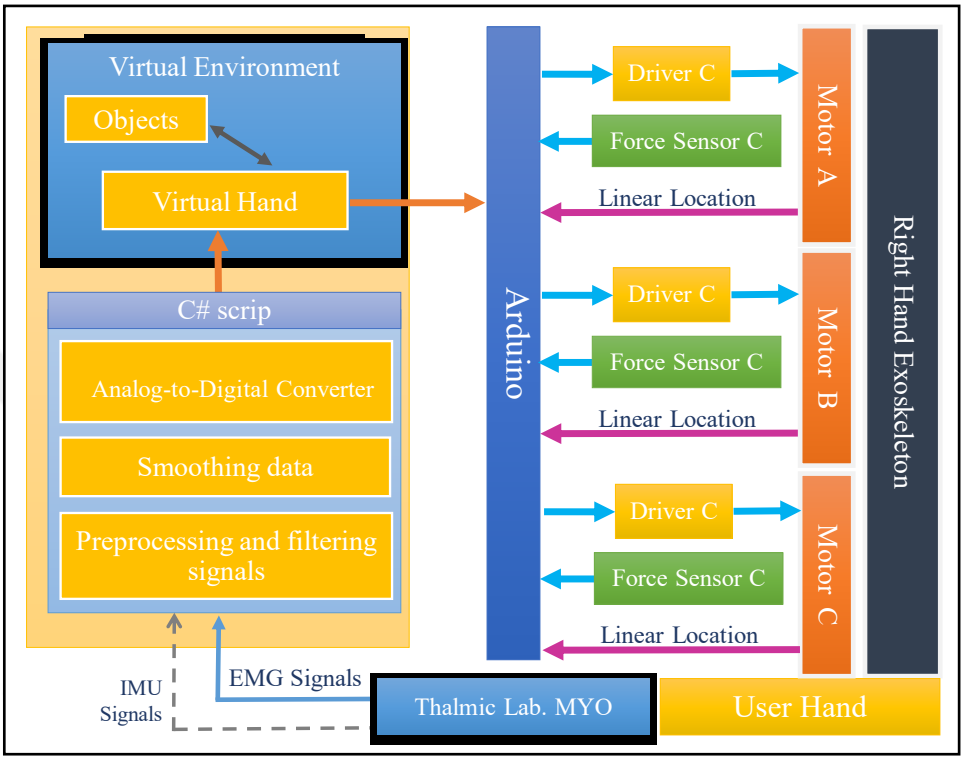


Figure 1.4 Different sections of our projects

CHAPTER 2

METHOD

In this chapter, we detailed the mechanical design which consisted of the right-hand exoskeleton mechanism and 3 dc motors to actuate the fingers, and electronic hardware design including microcontroller, dc motor drivers, force sensors, and bio-sensors, and then the software design including a simple virtual environment and the controller scripts.

2.1 Mechanical Design

2.1.1 Design of Hand Exoskeleton

The general concept of the 3D exoskeleton model was developed based on the linkage mechanism in which the fingers were actuated by three linear motors. Two motors work for the thumb and the index fingers, while the third motor moves the other 3 fingers (middle, ring, and little ones) together on the exoskeleton mechanism.

Before a detailed explanation of the general concept, that's ought to focus on basic biomechanical specifications of a human hand. As shown in Figure 2.1, the fingers include four kinds of bones, except the thumb has 1 boneless. Moreover, the kinematic configuration of the human hand represents 3bar links and 4 DoF in joints of each finger, as shown in Figure 2.2 We use the information in Table 2.1 suggestingthe range of motion for the angles of each joint [29].

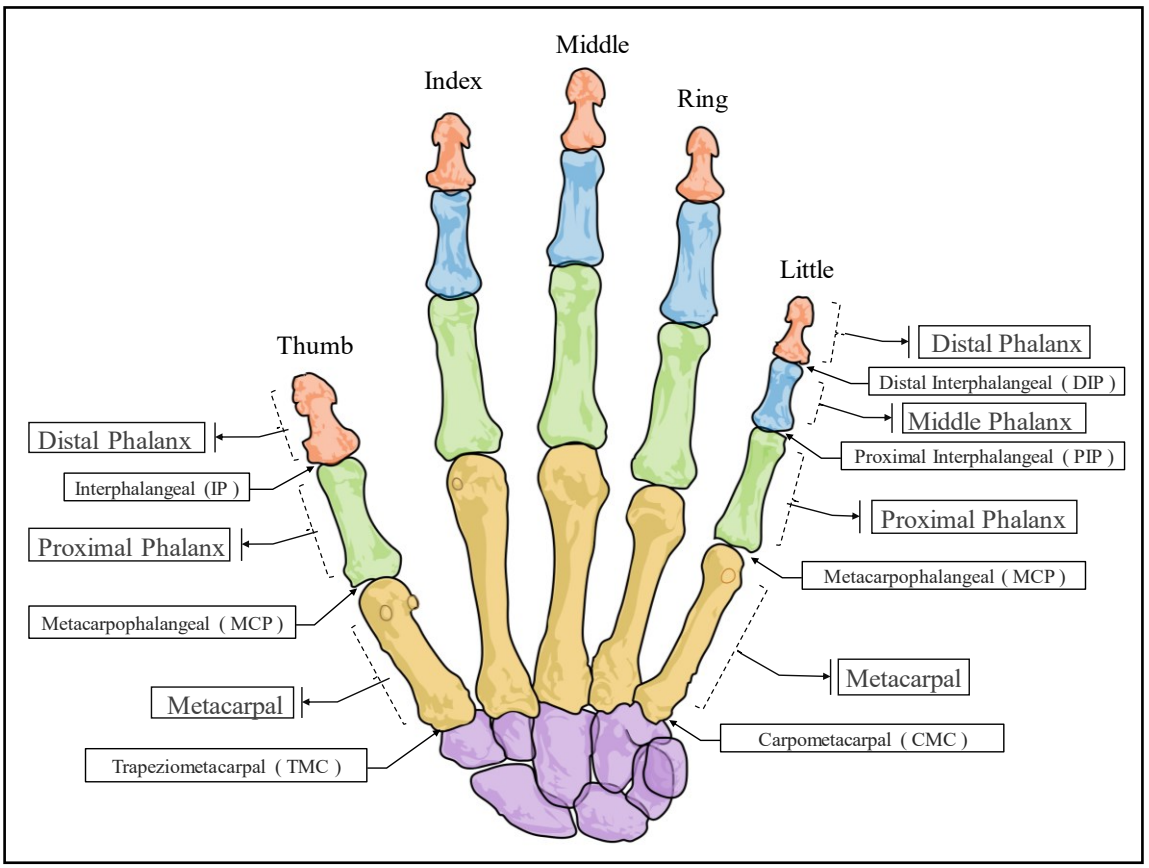


Figure 2.1 Anatomy of human hand with the name of bones and joints of each finger

Table 2.1 Suggested range of motion for each angles [29]

Finger	Joint	Flexion	Extension	Abduction /Adduction
Thumb	TMC	50° - 90°	15°	45° - 60°
	MCP	75° - 80°	0°	0°
	IP	75° - 80°	5° - 10°	0°
Index	MCP	90°	30° - 40°	60°
	PIP	110°	0°	0°
	DIP	80° - 90°	5°	0°
Middle	MCP	90°	30° - 40°	45°
	PIP	110°	0°	0°
	DIP	80° - 90°	5°	0°
Ring	MCP	90°	30° - 40°	45°
	PIP	110°	0°	0°
	DIP	80° - 90°	5°	0°
Little	MCP	90°	30° - 40°	50°
	PIP	135°	0°	0°
	DIP	90°	5°	0°

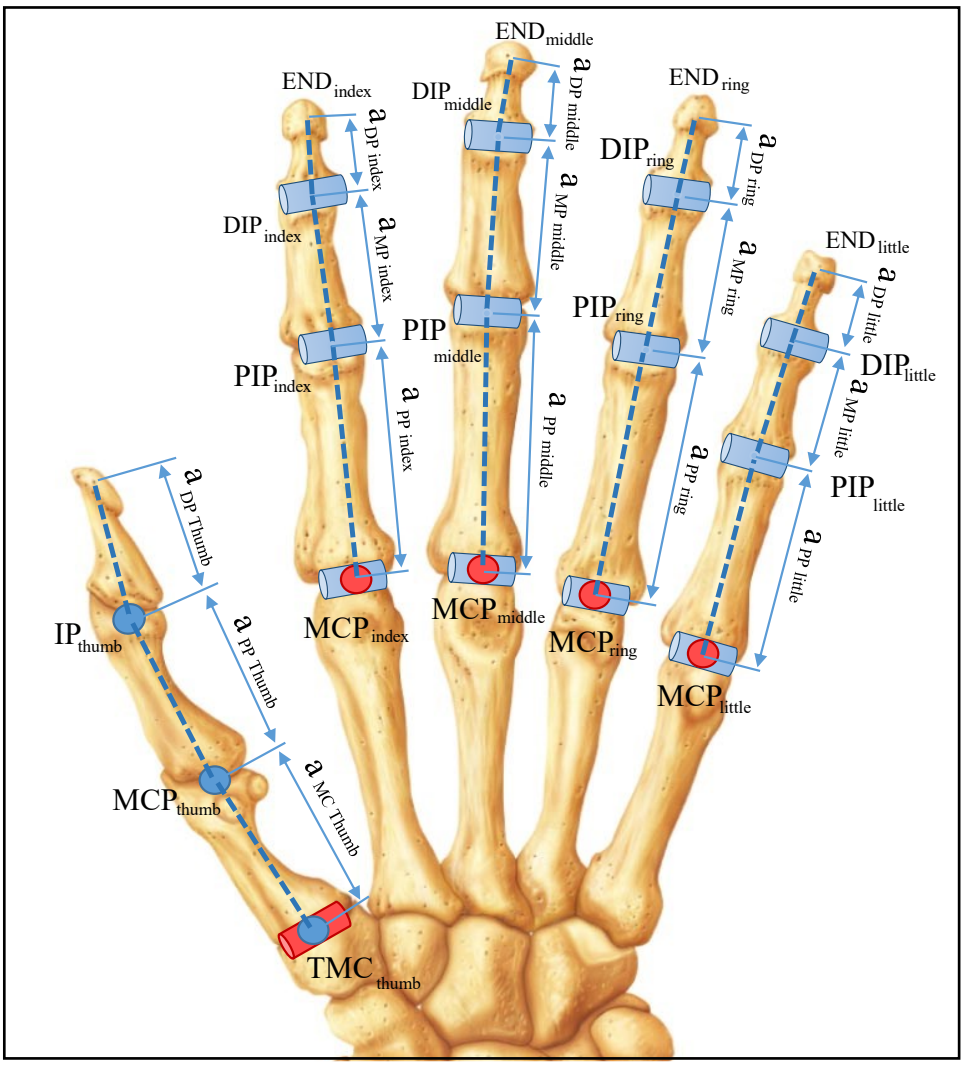


Figure 2.2 The kinematic configuration of the human hand. Each finger includes 3 links and 4 degrees of freedom

2.1.2 Kinematical Analyses of Human Fingers

Regarding the mechanism of our exoskeleton, all the joints rotate only around one ax. So all of the fingers have similar calculations. We can assume kinematical analyses as three rotation (3R) for each finger joint which shown in Figure 2.3 to be used to make Table 2.2.

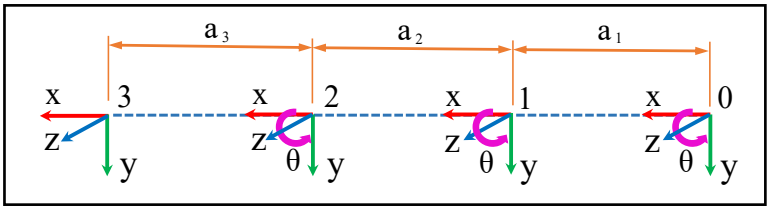


Figure 2.3 Assumed kinematical configuration of each human hand finger

Table 2.1 Denavit–Hartenberg (DH) parameter table

Joint i	Thumb finger	Other fingers	θ_i	d_i	a_{i-1}	α_{i-1}
0	TMC	MCP	θ_0	0	0	0
1	MCP	PIP	θ_1	0	a_1	0
2	IP	DIP	θ_2	0	a_2	0
3	END	END	0	0	a_3	0

Using DH parameter (Table 2.2), Equation (2.1) is direct kinematical analyses for each Link-Coordinate transformation of each finger is represented from equations (2.2-2.5) to get equation (2.6).

$$T_{\mathrm{k}-1}^{\mathrm{k}} = \begin{bmatrix} C\theta_{\mathrm{k}} & -C\alpha_{\mathrm{k}}\, S\theta_{\mathrm{k}} & S\alpha_{\mathrm{k}}\, S\theta_{\mathrm{k}} & a_{\mathrm{k}}\, C\theta_{\mathrm{k}} \\ S\theta_{\mathrm{k}} & C\alpha_{\mathrm{k}}\, C\theta_{\mathrm{k}} & -S\alpha_{\mathrm{k}}\, C\theta_{\mathrm{k}_1} & a_{\mathrm{k}}\, S\theta_{\mathrm{k}} \\ 0 & S\alpha_{\mathrm{k}} & C\alpha_{\mathrm{k}} & d_{\mathrm{k}} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

$$T_0^1 = \begin{bmatrix} C\theta_0 & -S\theta_0 & 0 & 0 \\ S\theta_0 & C\theta_0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

$$T_1^2 = \begin{bmatrix} C\theta_1 & -S\theta_1 & 0 & a_1 \\ S\theta_1 & C\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

$$T_2^3 = \begin{bmatrix} C\theta_2 & -S\theta_2 & 0 & a_2 \\ S\theta_2 & C\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

$$T_3^{\mathrm{END}} = \begin{bmatrix} 1 & 0 & 0 & a_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

$$T_0^{\mathrm{END}} = \begin{bmatrix} C_{012} & -S_{012} & 0 & a_1\, C_0 + a_2 C_{01} + a_3 C_{012} \\ S_{012} & C_{012} & 0 & a_1\, S_0 + a_2 S_{01} + a_3 S_{012} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

To simplify the equation (2.6), we trace the fingers angles while flexion which is shown in Figure 2.4. We complete Figure 2.5 for an index finger and Figure 2.7 for a thumb

finger. Using data in Figure 2.6 and Figure 2.8 can help us to find the approximate relation between extension/flexion angles of the joints in a finger.

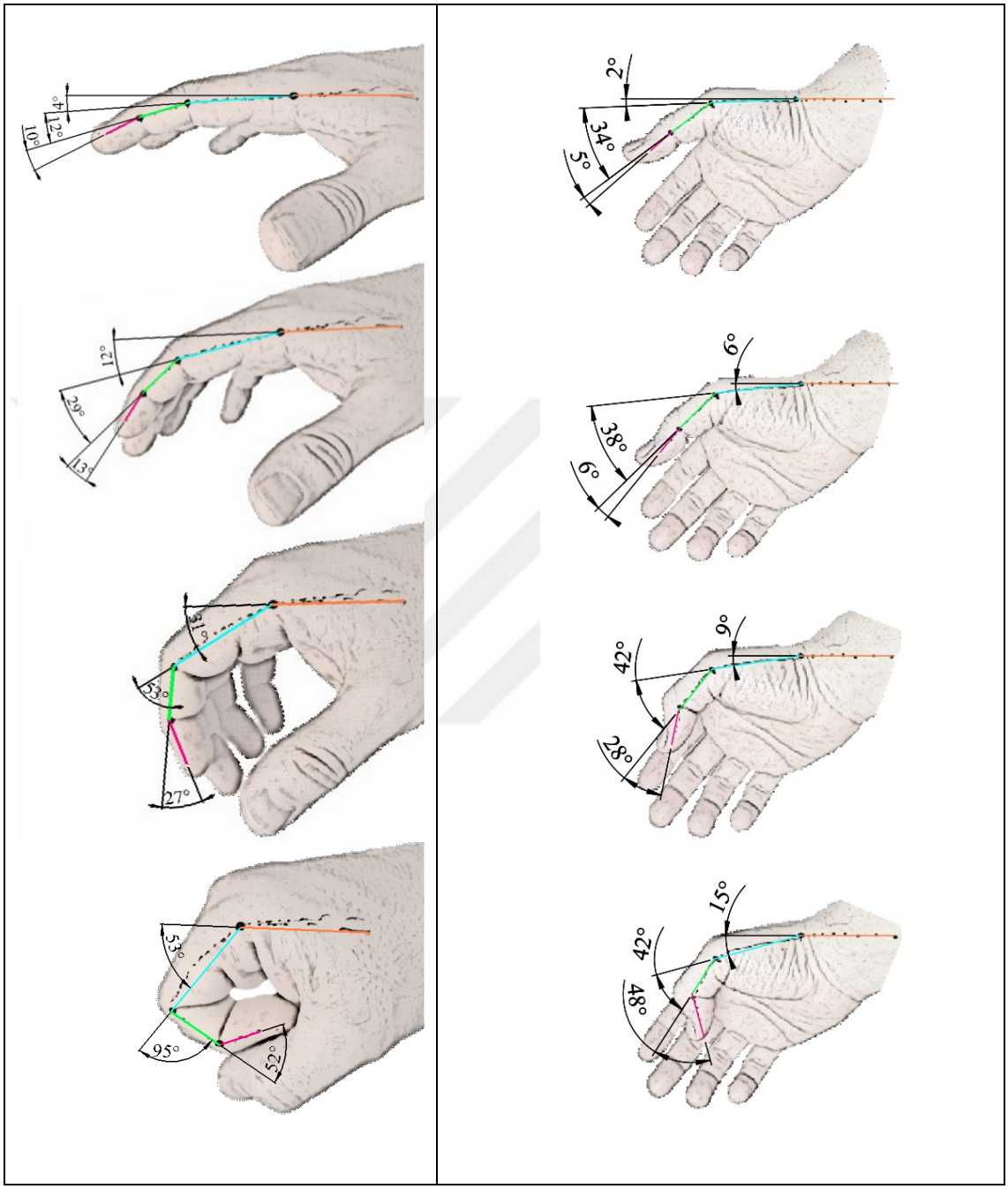
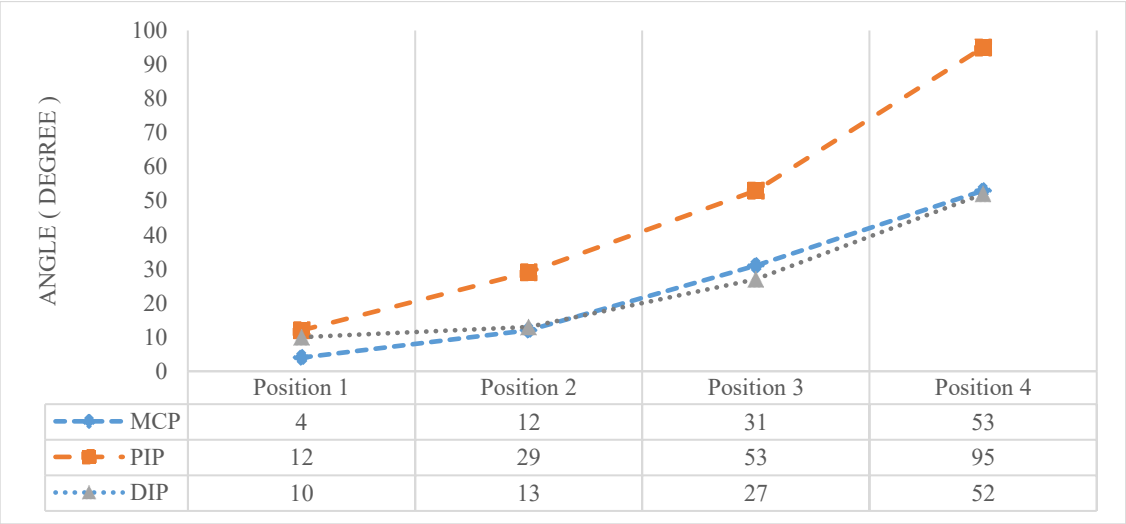


Figure 2.4 Sampling of the finger joint angles while flexion the finger (right; thumb finger, left; index finger)



	Position 1	Position 2	Position 3	Position 4
MCP	4	12	31	53
PIP	12	29	53	95
DIP	10	13	27	52

Figure 2.5 Flexion angle of different joints for the index finger while flexion mapped from Figure 2.4

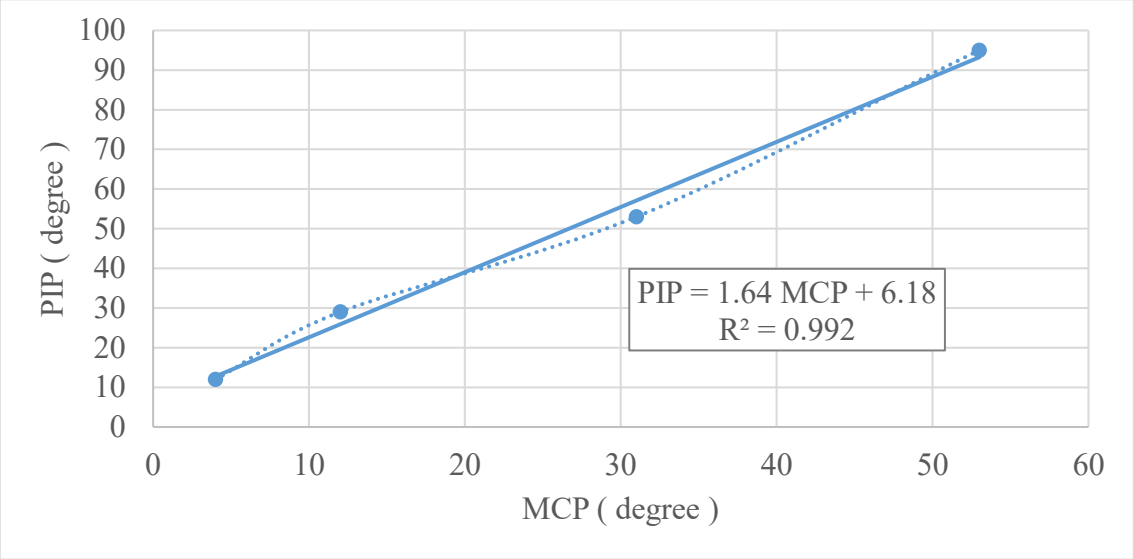


Figure 2.6 Linear relation between PIP and MCP values

As seen in Figure 2.5, MCP and DIP values are approximately equal in different positions. The relation between MCP and PIP values in Figure 2.6 can make linear relation with 0.992 R-squared value. To simplify the calculations, these relations can be generalized to middle, ring and little fingers too.

The same procedure applied to get the relation between the joints of thumb finger individually, which presented in Figure 2.7 and Figure 2.8.

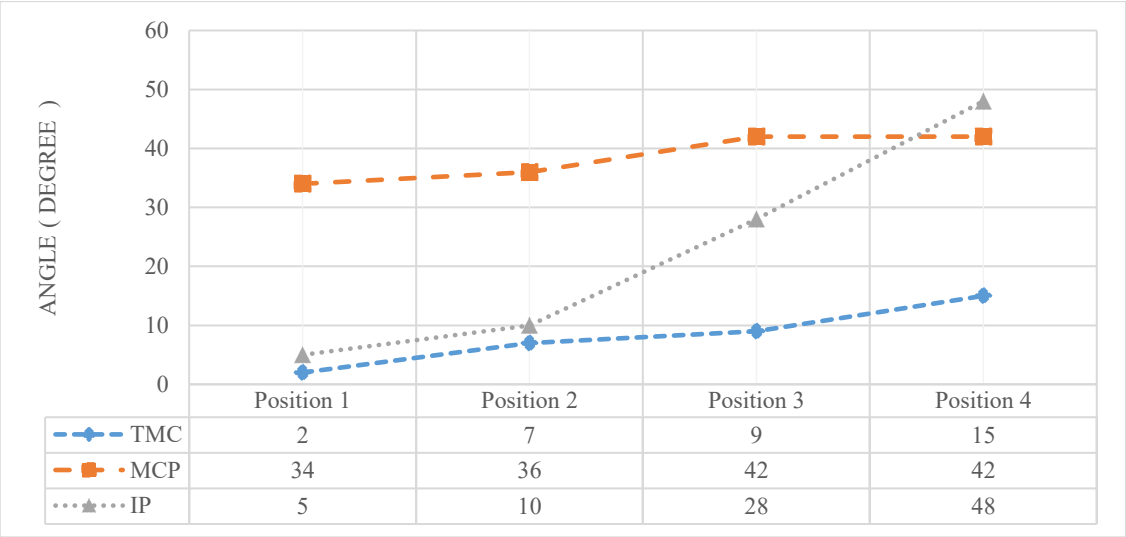


Figure 2.7 Flexion angle of different joints for the thumb finger while flexion mapped from Figure 2.4

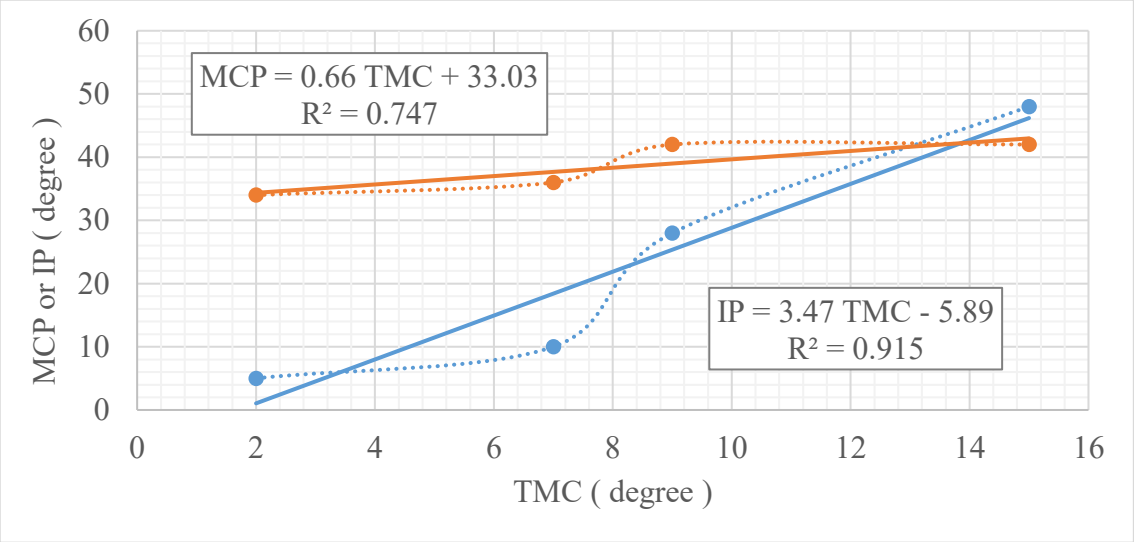


Figure 2.8 Linear relation between MCP and TMC values also linear relation between IP and TMC values

Table 2.2 Relation between joints of fingers during flexion

Thumb	Index, Middle, Ring, Little
TMC	MCP
MCP = 0.66 TMC + 33.03	PIP = 1.64 MCP + 6.18
IP = 3.47 TMC - 5.89	DIP = MCP

Replacing the results of Table 2.3 into the formula (2.6) we can produce mathematical maximum (x,y) values for the endpoint of fingertips :

$$\begin{cases} x = a_{\mathrm{MC}} Cos(TMC) + a_{PP} Cos(TMC + MCP) + a_{DP} Cos(TMC + MCP + IP) \\ y = a_{\mathrm{MC}} Sin(TMC) + a_{PP} Sin(TMC + MCP) + a_{DP} Sin(TMC + MCP + IP) \end{cases} \quad (2.7)$$

For thumb finger (TMC = 15°) :

$$\begin{cases} \mathrm{MCP} = 31.12\,(15)^{0.1104} = 41.96 \\ \mathrm{IP} = 3.47\,(15) - 5.89 = 46.16 \end{cases}$$

$$\rightarrow \begin{cases} x = 40\,Cos(15) + 30\,Cos(56.96) + 20\,Cos(103.12) = -19.91 \\ y = 40\,Sin(15) + 30\,Sin(56.96) + 20\,Sin(103.12) = 48.50 \end{cases}$$

For index finger (MCP = 53°) :

$$\begin{cases} \mathrm{MCP} = 1.64\,(53) + 6.18 = 41.96 \\ \mathrm{DIP} = 53 \end{cases}$$

$$\rightarrow \begin{cases} x = 40\,Cos(53) + 20\,Cos(146.1) + 20\,Cos(199.1) = -44.67 \\ y = 40\,Sin(53) + 20\,Sin(146.1) + 20\,Sin(199.1) = 17.34 \end{cases}$$

For middle finger (MCP = 53°) :

$$\begin{cases} \mathrm{MCP} = 1.64\,(53) + 6.18 = 41.96 \\ \mathrm{DIP} = 53 \end{cases}$$

$$\rightarrow \begin{cases} x = 50\,Cos(53) + 25\,Cos(146.1) + 20\,Cos(199.1) = -53.93 \\ y = 50\,Sin(53) + 25\,Sin(146.1) + 20\,Sin(199.1) = 26.30 \end{cases}$$

Flowing graphs in Figure 2.9 and Figure 2.10 represents the relation between the feed rate of dc electromotor and calculated mean flexion degree of each joint of the fingers. Flexion degree is in radians which are calculated by the relations in Table 2.3. The dc electromotor actuators have 10-bit position sensor which the end of the stroke is represented by the max value as 100.

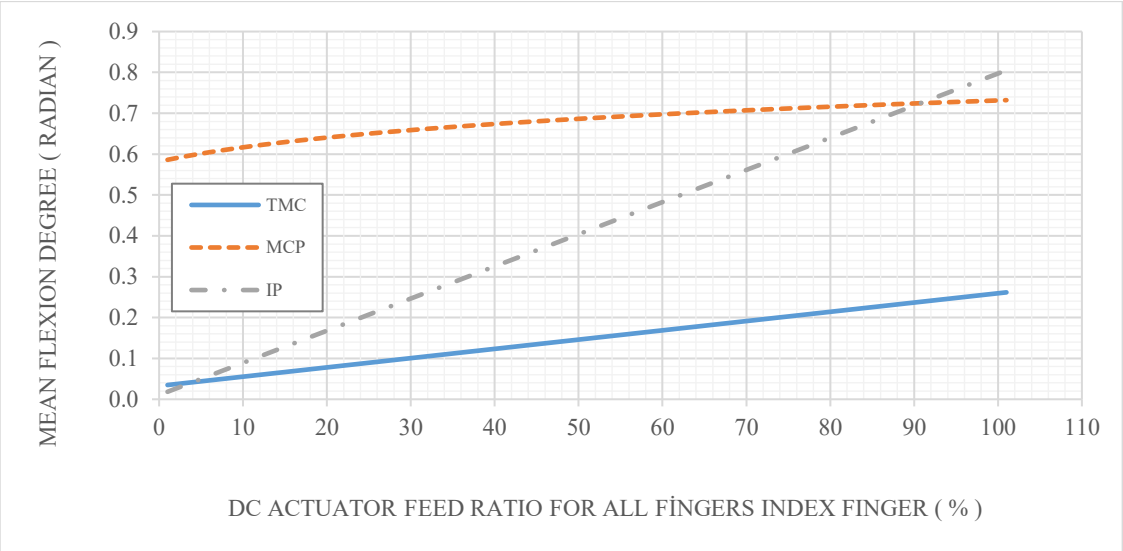


Figure 2.9 The relation between the feed ratio of dc electromotor and calculated mean flexion degree of each joint for the thumb finger

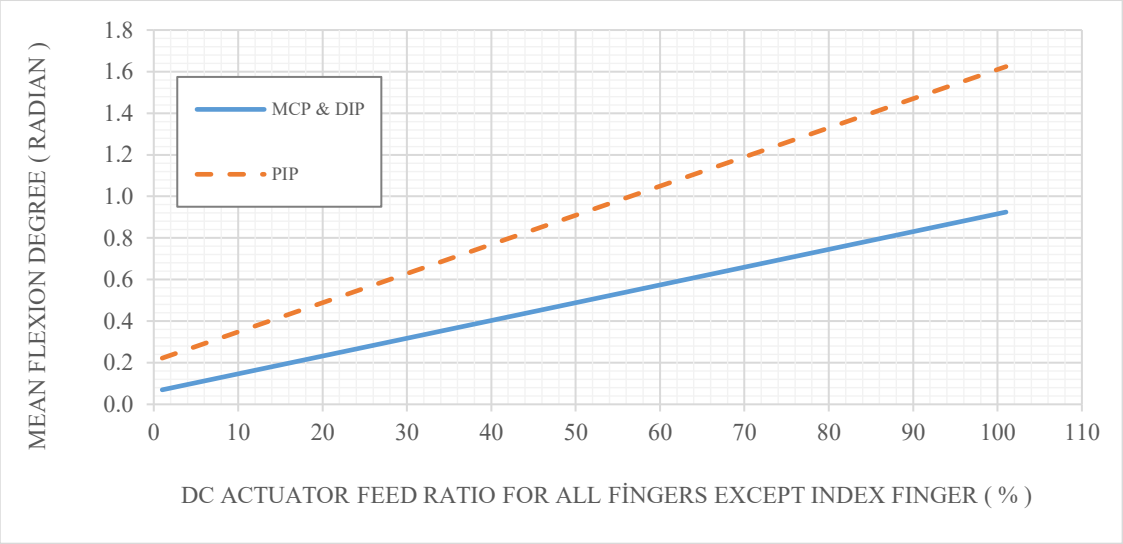


Figure 2.10 The relation between the feed ratio of dc electromotor and calculated mean flexion degree of each joint for the thumb finger

2.1.3 Mechanical Mechanism Design

Variety of concepts designed during the project to cover our scope as described in previous chapter. The base of the concept designed upon bar linkage system. Different concepts designed and prototyped as Figure 2.11 the results are satisfying the mechanism. But it needs some editions to fit well over the hand. A general problem caused by banded hinge system over finger joints which eliminated exoskeleton to fit over different hands sizes. Thanks to 3D printing technology, enabled the fast and flexible production of parts directly from 3D CAD data. PA 2200 powder was selected as a 3D printing material

which is explained in Appendix-A. The printing process is done by a laser sintering 3d printer. The result shows it can take small impacts and resist some pressure while being bent. 3D printed parts assembled successfully. Figure 2.12 shows how we prepared the designed parts into the laser sintering 3D printer as STL file format.

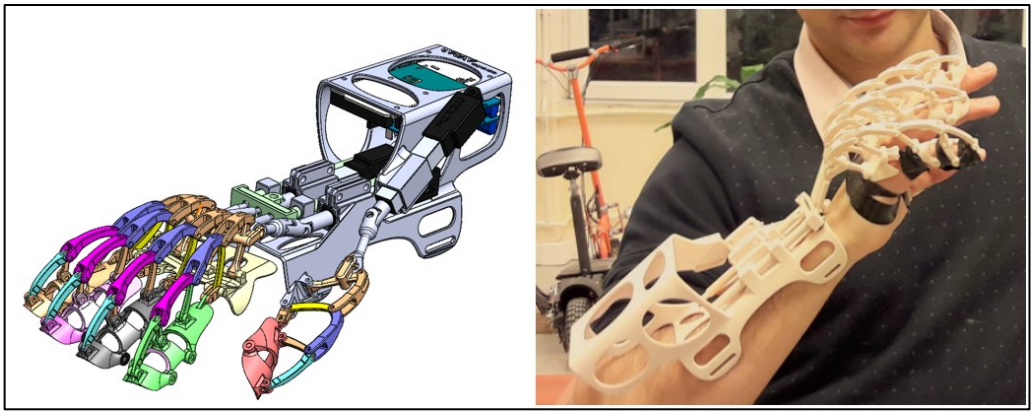


Figure 2.11 Primary concept of the exoskeleton mechanism (left), 3D printed Prototype of the concept (right).

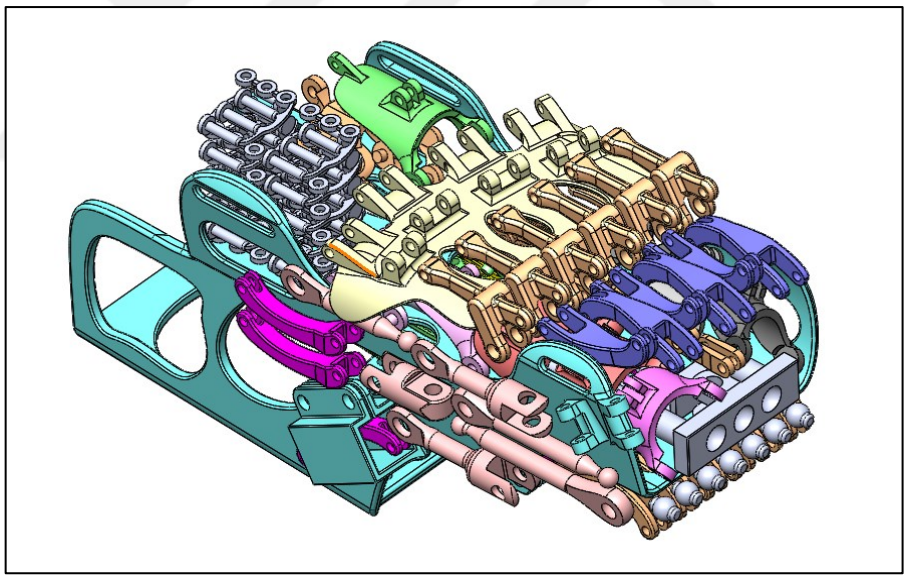


Figure 2.12 Prepared designed parts layouts to fit in the 3D printer sintering machine

At the last mechanical design revision, hinged linkages between the fingers removed showed in Figure 2.13 which help the hand to fit in various hands sizes. The revised mechanism is covered our required range of motions for fingers which calculated above (Figure 2.14). The next step will help us to calculate static analyses of the critical mechanical parts. The technical dimensions of the parts are available in APPENDIX – G.

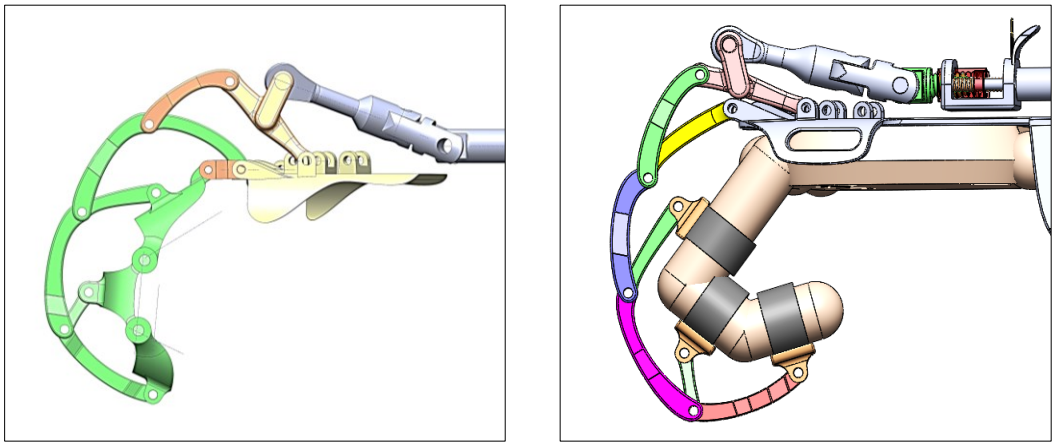


Figure 2.13 Primary prototype of the fingers (left), the final revision of the fingers mechanism (right)

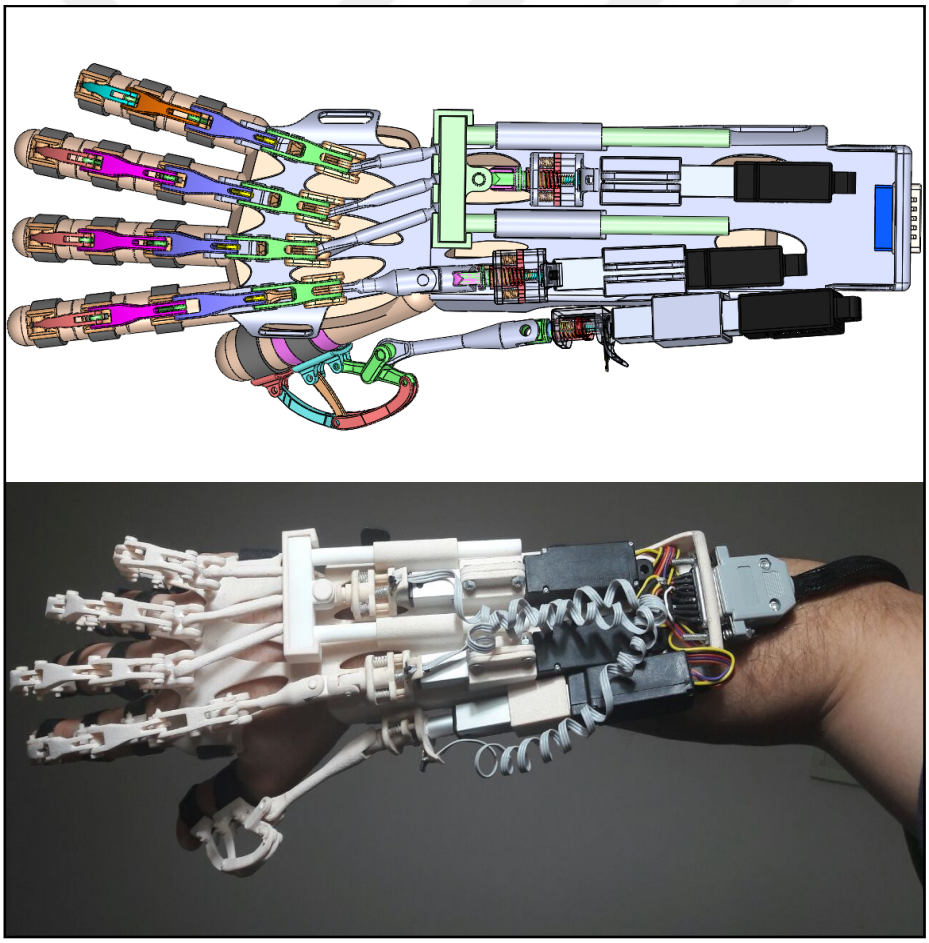


Figure 2.14 Final prototype of the hand exoskeleton

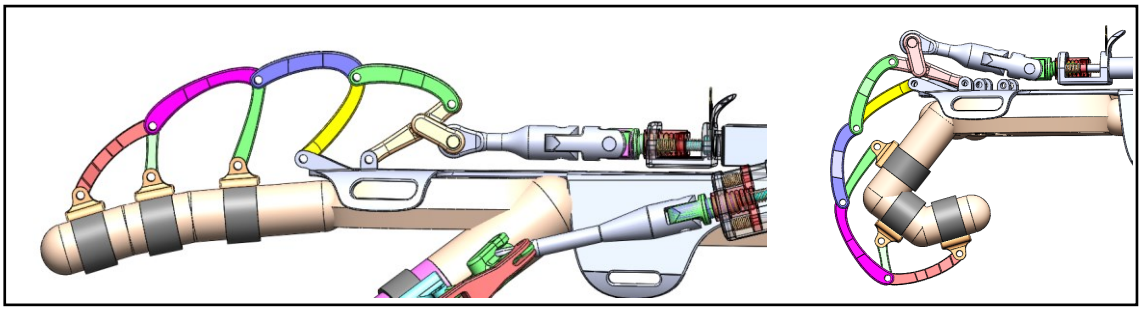


Figure 2.15 Ultimate flexion and extension position of the fingers

In order to choose appropriate static force, we calculate the perpendicular to phalange axis. [30] calculated continuous maximum force perpendicular to fingertips from 8 to 15 newtons. But regarding our limitations on the electromotor selectin, we assume to calculate the ultimate motor force of 100N according to APPENDIX – C. The critical condition is also assumed as shown in Figure 2.16. Calculation (2.8) represents the maximum impact force which all the arts should tolerate and (2.9) is the maximum applied momentum.

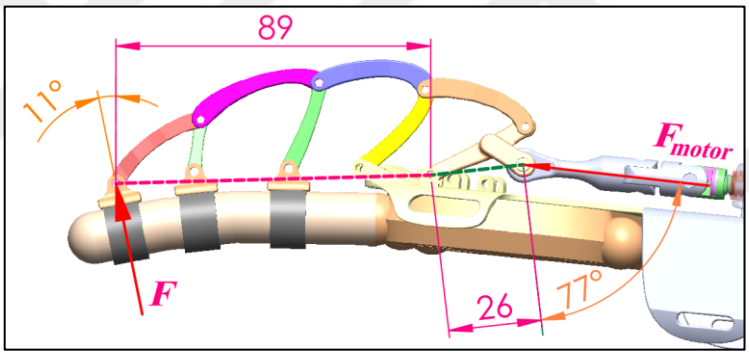


Figure 2.16 Force and coupling condition on the index finger

$$F_{ultimate} = \frac{26\ (100)\cos(77)}{89\ \cos(11)} = 6.7N \quad (2.8)$$

$$M_{max.} = 0.089\ (6.7)\cos(11) = 0.58\ N/m \quad (2.9)$$

Static analysis was done by COSMOS add-in software in Solidworks software. The aim of the analysis is to find the minimum sectin for the bars and hinges. As a sample, one of the parts is selected for the analysis which shown in Figure 2.17. Material properties for the analysis are also presented in Table 2.4.

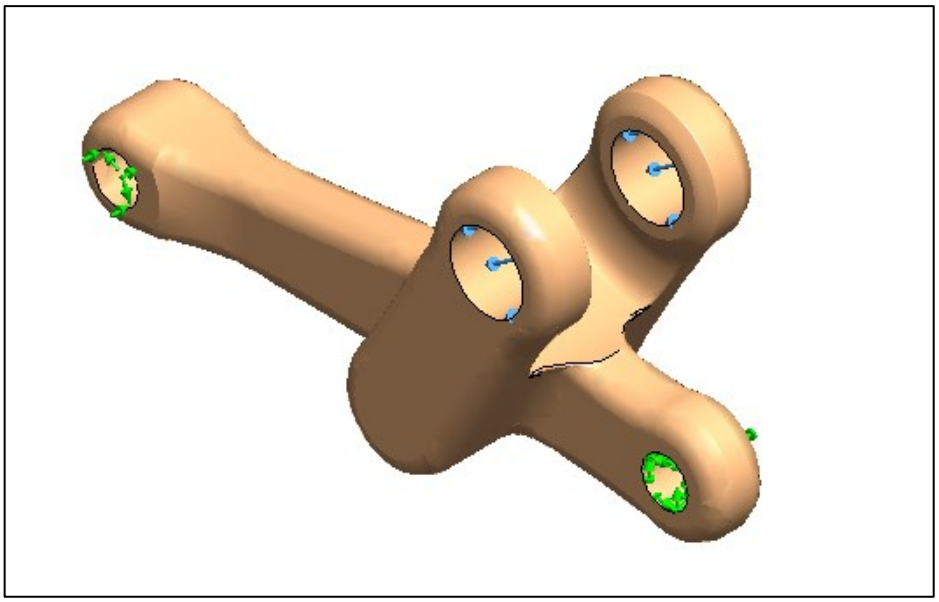


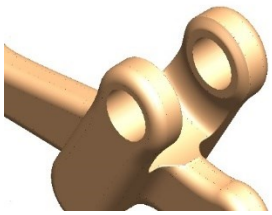
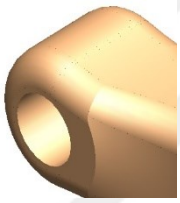
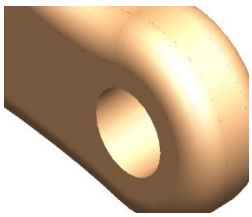
Figure 2.17 Sample part of the mechanical system that static analyses applied to it

Table 2.3 Material properties applied in the static analysis

Name:	Nylon 12 PA
Model type:	Linear Elastic Isotropic
Default failure criterion:	Max von Mises Stress
Yield strength:	2.241e+009 N/m^2
Tensile strength:	3.6e+007 N/m^2
Elastic modulus:	2.896e+009 N/m^2
Mass density:	1250 kg/m^3

Force and fixture condition on Table 2.3 are the ones that applied to finite element analysis. In Figure 2.18 the finite element analysis result for one of the critical parts with the described material which shows minimum 535.4 value for the factor of safety (FOS), maximum von Mises Stress of 4.186 e+6 which is smaller than the yield strength and displacement amount.

Table 2.18 Loading and fixture conditions that considered at analysis of the part. Result forces indicate the reaction of applied 6.7N force for each joint.

<table>
<tr><td>`Load name</td><td>Load Image</td><td>Load Details</td></tr>
<tr><td>Force</td><td>  </td><td>Entities: 2 face(s)
Type: Apply force
Values: ---, ---, 6.7 N</td></tr>
<tr><td>Fixture name</td><td>Fixture Image</td><td>Fixture Details</td></tr>
<tr><td>Fixed Hinge</td><td>  </td><td>Entities: 1 face(s)
Type: Fixed Hinge</td></tr>
<tr><td colspan="3">Resultant Forces
<table>
<tr><td>Components</td><td>X</td><td>Y</td><td>Z</td><td>Resultant</td></tr>
<tr><td>Reaction force(N)</td><td>6.4901</td><td>3.3121</td><td>-0.0004618</td><td>7.2863</td></tr>
</table></td></tr>
<tr><td>Fixed</td><td>  </td><td>Entities: 1 face(s)
Type: Fixed Geometry</td></tr>
<tr><td colspan="3">Resultant Forces
<table>
<tr><td>Components</td><td>X</td><td>Y</td><td>Z</td><td>Resultant</td></tr>
<tr><td>Reaction force(N)</td><td>4.6196</td><td>4.2118</td><td>0.0006085</td><td>6.2514</td></tr>
</table></td></tr>
</table>

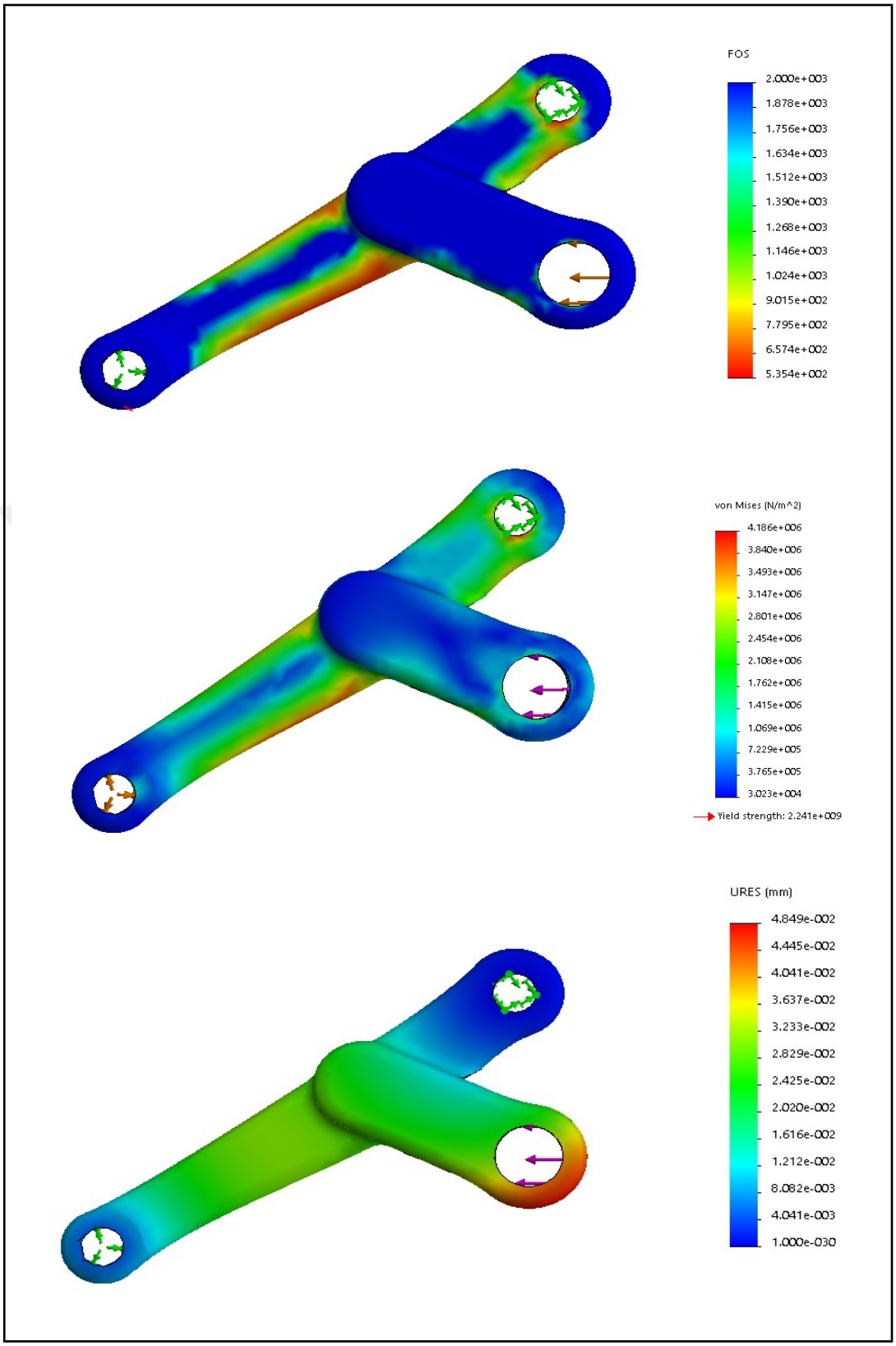


Figure 2.18 Finite element analysis result for one of the critical parts. The factor of safety (up), maximum von Mises Stress (middle) and displacement amount (down)

2.2 Biosensing

2.2.1 Myo Gesture Control Armband

The electromyography (EMG) is the base of our biosensing data that transfer conditions of the motor muscles interacting with the fingers flexion. As shown in the Figure 2.19, Myo gesture control armband gets motor interactions by 8 medical grade stainless steel sEMG(surface electromyography) sensors [31]. Similar to other surface electrodes, the EMG signals returned by the sensors represent the electric potential of the muscles as a result of muscle activation [32]. It senses the signals of the arm muscles which motors the fingers (Figure 2.20). The electric potential of muscle is small by the range of sub millivolts. So, signals are sensitive to other sources of electrical noise such as electrical noise induced by wall-electricity. The range of potentials provided by the Myo armband is between -128 and 128 in units of activation measured by the EMG sensors. The Myo armband is capable of pulling EMG data at a sample rate of 200Hz. The Myo armband also has a nine-axis inertial measurement unit (IMU) which contains a three-axis gyroscope, three-axis accelerometer, and a three-axis magnetometer [31] Figure 2.21 shows an example of MYO armband signals for random actions.The orientation data indicates the positioning of the armband in terms of roll, pitch, and yaw. The angular velocity of the armband is provided in a vector format and the accelerometer represents the acceleration the Myo armband is undergoing at a given time. However, the Myo armband is better suited for determining the relative positioning of the arm rather than the absolute position, a consideration to be aware of when applying pattern recognition algorithms. The Myo armband is able to pull IMU data at a sample rate of 50Hz [31].

Figure 2.19 The Myo gesture control armband [31]

Myo armband has built-in 250-mA/hr chargeable battery. EMG sensors using STMicroelectronics 78544 IC that is undefined but enhanced a signal conditioner and

amplifier. The main board comprises primarily the USB 2.0 port. The data transmit data using Nordic Gazelle 2.4 GHz protocol Bluetooth antenna [33].

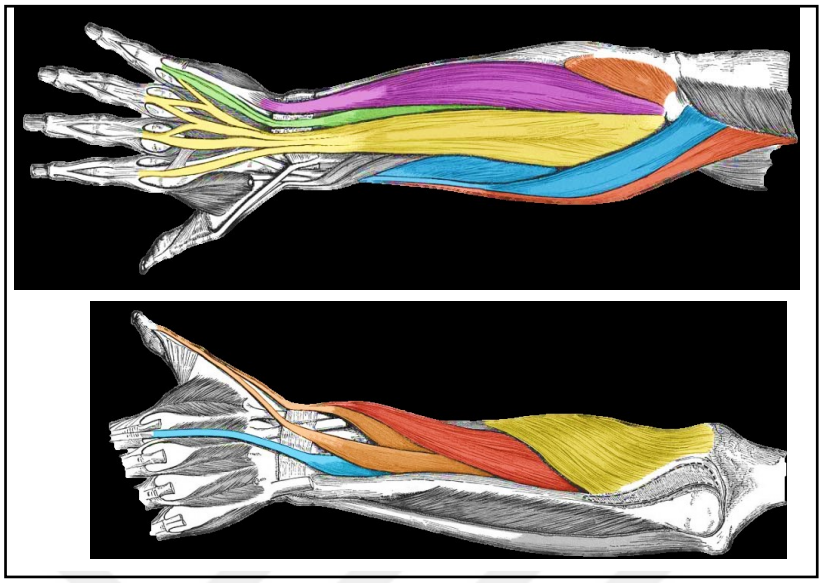


Figure 2.20 Anatomy of the lower arm muscles of the posterior forearm superficial deep teach anatomy

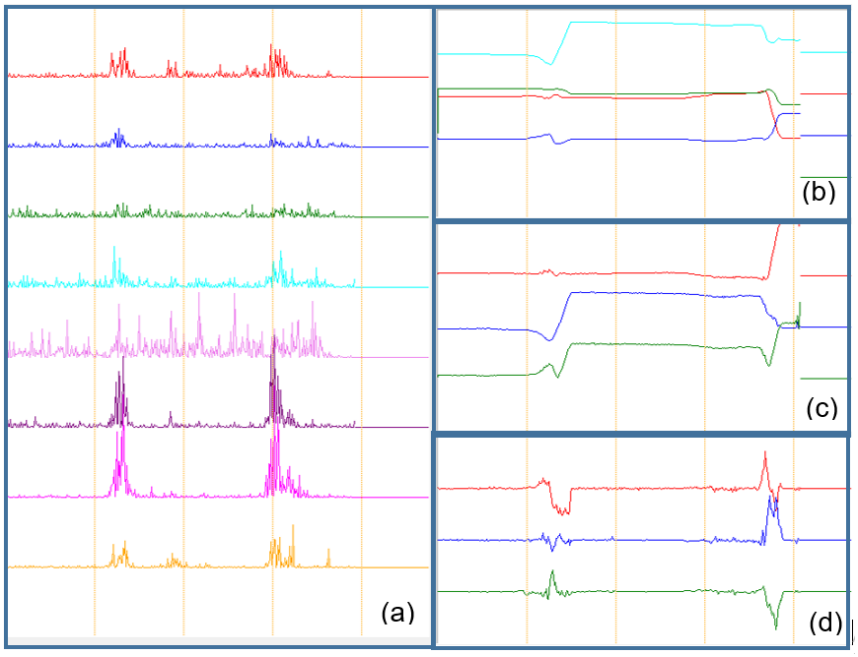


Figure 2.21 Example of MYO armband signals for random actions: (a) Absolute preprocessed of 8 EMG signals, (b) 4 orientation signals, (c) 3 accelerometer signals, (d) 3 gyroscope signals

2.2.2 EMG Data Signal Processing

Examining of EMG rough data during full fingers flexion and extension in different arm orientations (Figure 2.22) shows although the motor activity of fingers is recognizable

in the sensors placed near the related muscles, it needs to process and filter the noisy signals. Based on C# scripting, the rough data streaming was done by using ThalmicHub library as a connection channel. This library allows access to one or more Myo armbands. The general overview of the algorithim we used for processing EMG signals is shown in Figure 2.23.

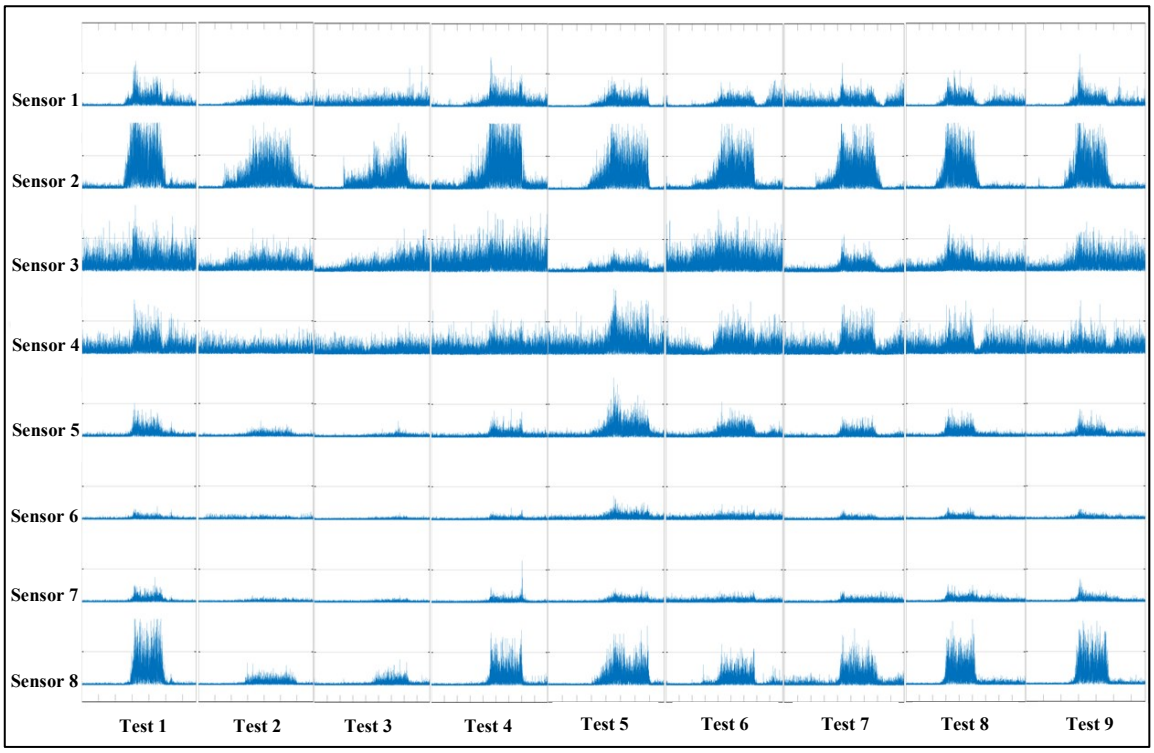


Figure 2.22 Preprocessed 8 signals tested 9 times. Each set of test recorded while flexion all the fingers (fist) then extension. Hand orientations are different for each set of test.

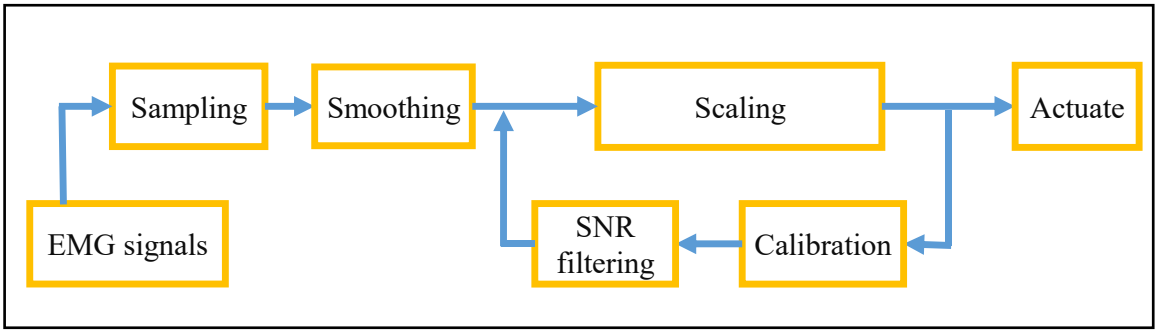


Figure 2.23 General overview of our EMG signal data handling

During the smoothing preprocess, first, the absolute value of streamed data calculated. It performs the amplitude of streamed value for each EMG sensor. Then, we discrete the values to 50 Hz of the sampling rate. The smoothing is done by using Sampling Moving Average (SMA) method [34] for 30 data samples. At the last step, we normalized the result between 0 to 100 as a percentage of fingers flexion which applies to actuate the dc

motors and VR hand, but the input value needs to be calibrated once before normalizing. At this step, the smoothed value of each EMG sensor streamed for a specified period of time while the flexion of the fingers is 100% (fist position). Then, mean average value for each sensor, is calculated. We repeat this procedure again while flexion is near 0% (Open hand position).

To get the better understanding of the Figure 2.23, we used statistical Student's t-test to compare 8 EMG signals while fist position and open hand position. As shown in Figure 2.24, each electrode gives statistically different signals according to different conditions. Nevertheless, Signal-to-Noise Ratio (SNR) analysis was also performed on these data sets. Generally, it is stated that the signals with 12 dB of SNR are very noisy, those with 12 to 20 dB are normal, those with 20 dB are better. Accordingly, as is shown in the Figure 2.25, the data obtained from 1,2 and 8th EMG electrodes are more clear.

In our program, to filter any noisy sensor values and also the dislocation of the Armband, on the next step, Signal to Noise Ratio (SNR) values are calculated according to formula (2.1). It is the name of EMG sensor. The sensors below the threshold are ignored and the remains are sum together. Formula (2.2) and (2.3) are used to get Upper Control Limit (UCL) and Lower Control Limit (LCL). Formula (2.4) scales the LCL and UCL between 0 to 100 and defines the sum of the new incoming values inside this domain.

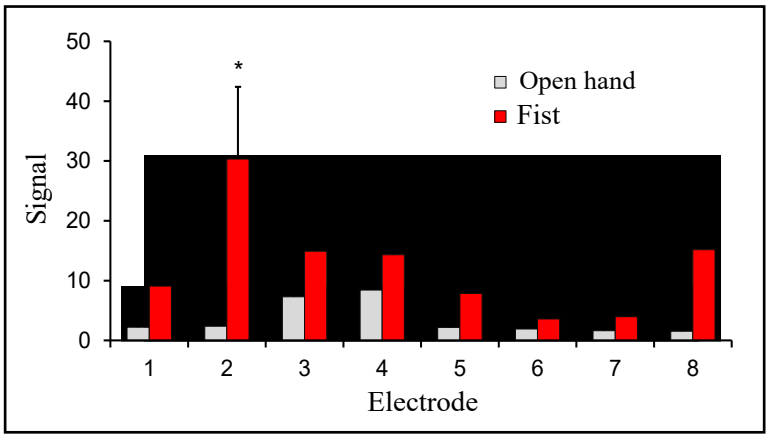


Figure 2.24 Student's t-test statistical analysis of EMG signals between open hand and fist. Data are shown as mean ± SD and * P <0.05

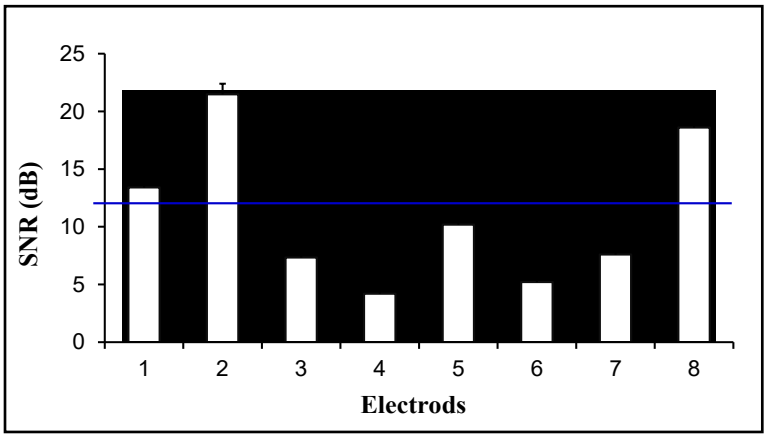


Figure 2.25 Signal-to-Noise Ratio (SNR) detected at each electrode.

$$\frac{Mean\ Average\ of\ Sensor\ i\ while\ grasp}{Mean\ Average\ of\ Sensor\ i\ while\ open\ hand} \geq Threshold \tag{2.1}$$

$$UCL = \sum_{i=0}^{8} Mean\ Average\ of\ Sensor\ i\ while\ grasp \tag{2.2}$$

$$LCL = \sum_{i=0}^{8} Mean\ Average\ of\ Sensor\ i\ while\ open\ hand \tag{2.3}$$

$$Actuate = \frac{100\ (\sum_{i=1}^{8}(SMA)_i - LCL)}{UCL - LCL} \tag{2.4}$$

2.3 Sensors

FSR 400 Short Tail is the model of force sensor that used in the design. Technical details about the sensor attached in APPENDIX – D. The force sensors are implemented to a mechanical mechanism to detect applied force in both directions during fingers flexion and extension. As shown in Figure 2.26, the stroke of movement balanced in the middle which effects the balanced force (threshold). We can manually tune the balanced force using screw shaped mechanical part at the top of the moving element. Backlash movement applied by a spring and the forward movements are applied by two springs. The measured values are defined as impedance controller which used to control motor position and velocity which is shown in Figure 2.27. In the current figure, measured amplitude belongs to the Resistance Force Sensor (RFS) which linked to the index finger. To smooth the data, The Simple moving average (SMA) for the 25 samples in each period applyed too. SMA process causing a time delay which shown in Figure 2.28. From

the time value, 0 to 200 sensor is in the steady condition. During the flexion, the amplitude reduces to reach below 350 units where the spring displacement was 3mm. On the other hand, the value change after 1200, caused by finger extension or physical resistance.

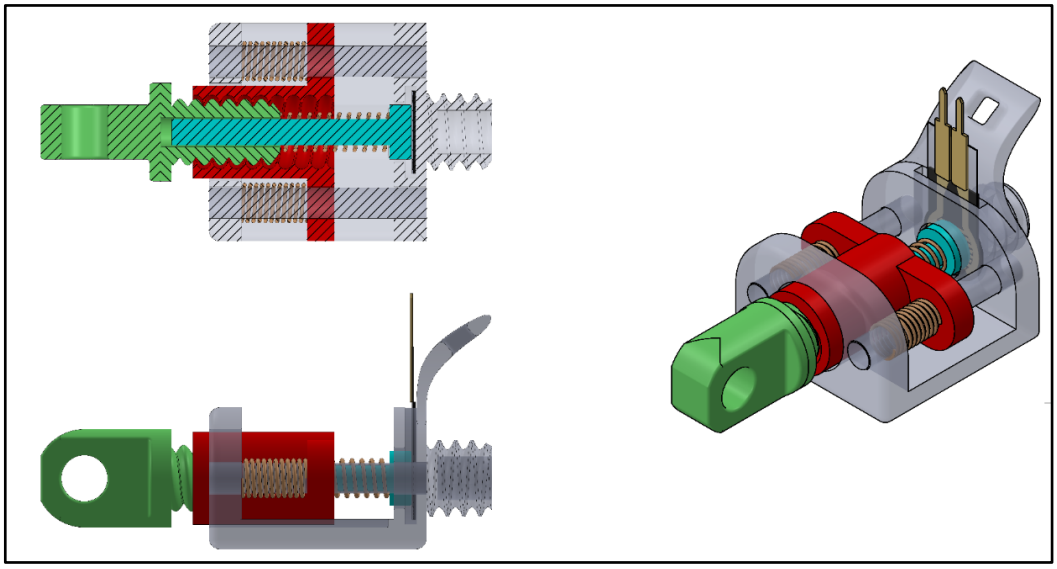


Figure 2.26 The mechanical mechanism designed to apply specific loading on FSR sensor

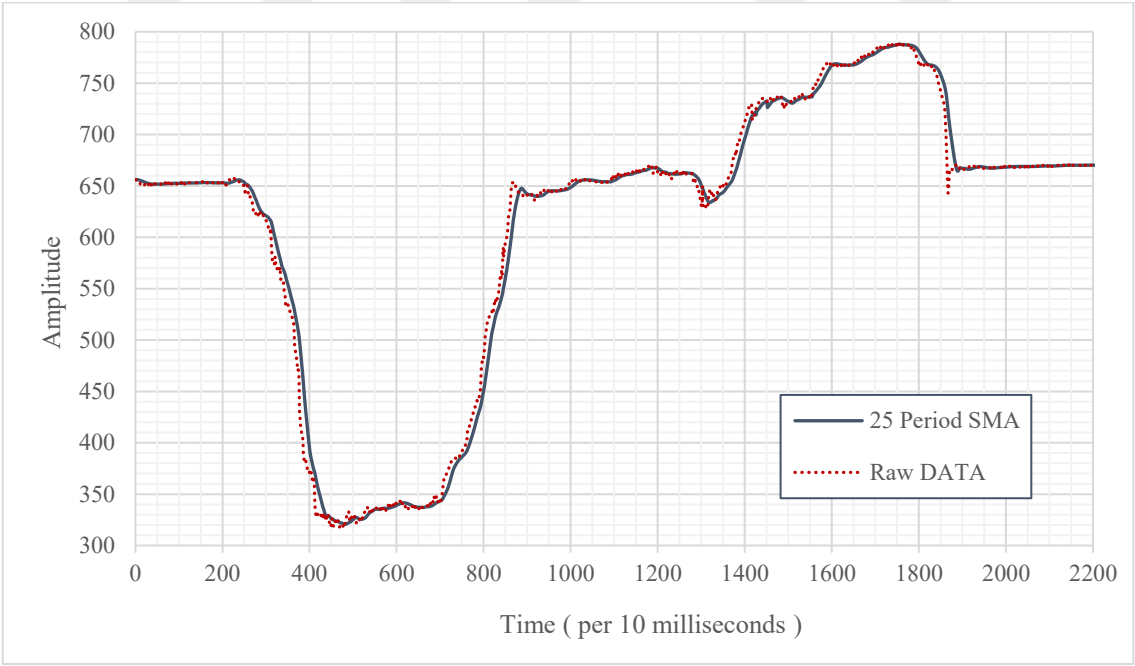


Figure 2.27 Measured values of amplitude of the Resistance Force Sensor (SNR)

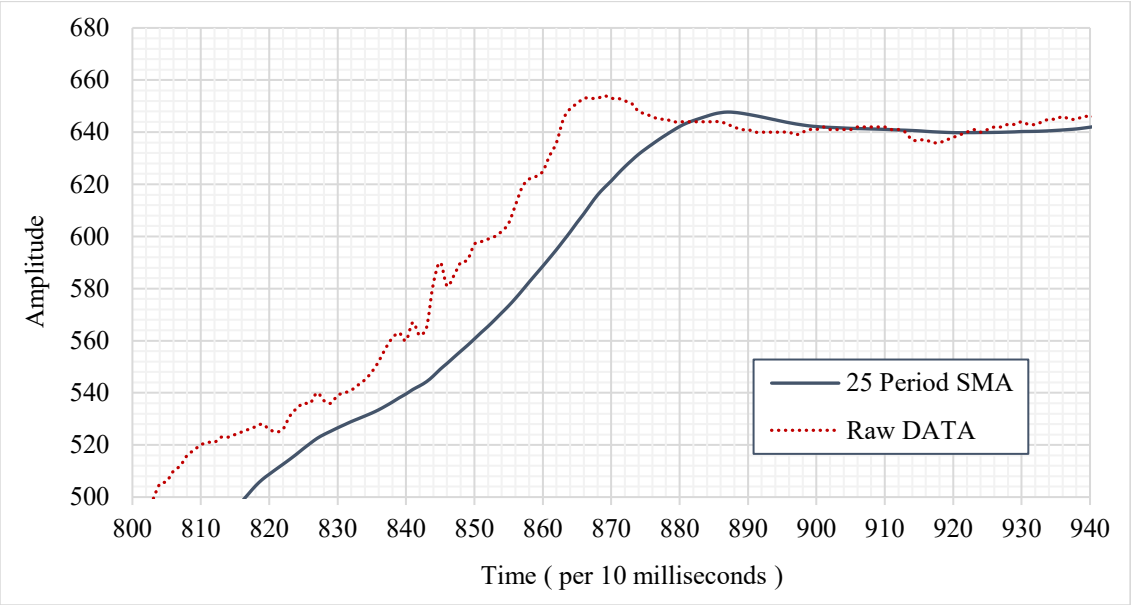


Figure 2.28 Zoom in Figure 2.16 shows a time delay after the smoothing

To calculate the applied forces to sensor we include our configurations to the voltage divider formula (2.4) from the information in APPENDIX – D. In this formula, 10kΩ used for resistant and 4.87v as the input voltage. Figure 2.29 is obtained by using the information on Figure 2.27.

$$V_{out} = \frac{R_m V}{R_m + R_{FSR}} = \frac{10 \times 4.87}{10 + R_{FSR}} \tag{2.4}$$

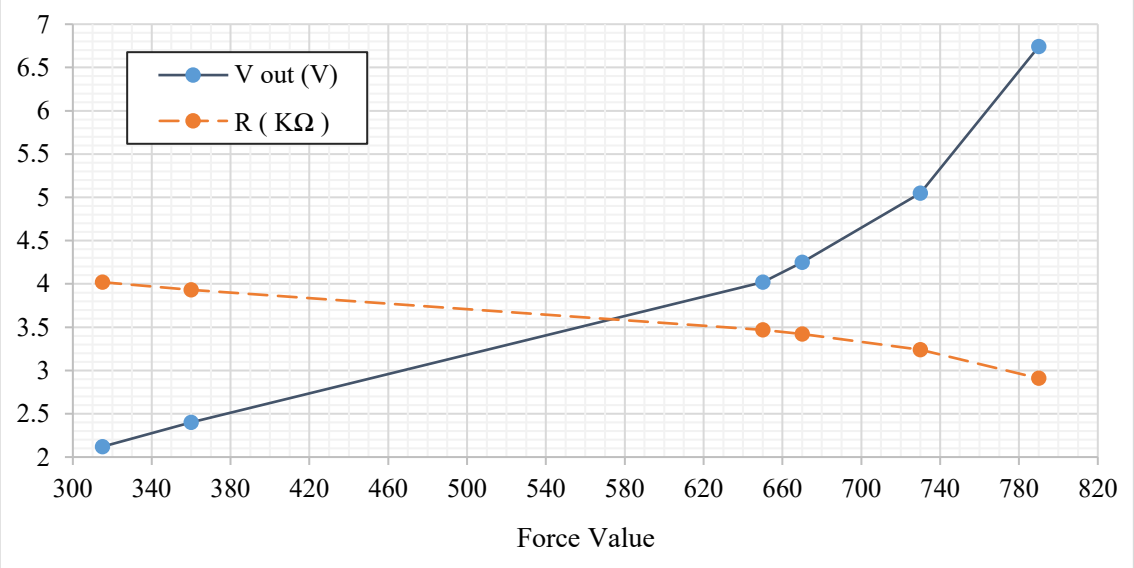


Figure 2.29 Relation between implemented force on a sensor, resistance (R_{FSR}) and calculated output voltage (V_{out}) via voltage divider formula (2.3)

2.4 Virtual Reality

One another part of the project is to create a Virtual Environment (VE). We need VE as visual interface between programmed scripts and the operator which can simulate the hand gesture in a VE. Using the Unity® programing language helped us to compile C# scripts to control a VE.

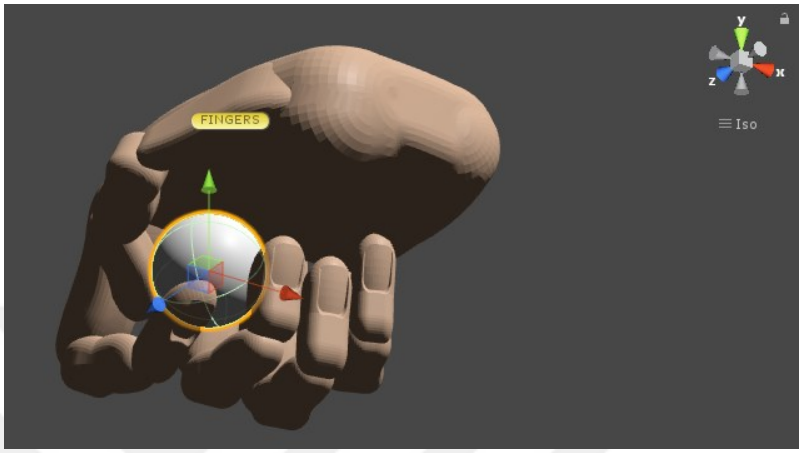


Figure 2.30 Collision can be detected between fingers of the virtual hand and the 3D objects

Some virtual objects have been included in the virtual environment in order to be touched and moved by the virtual hand (Figure 2.30). In the virtual environment interface, the sampling time is 12.8 ms. The slider bars are inserted as an option to change the value of the flexion amount of the fingers (Figure 2.31). In addition, exoskeleton hand movements can be observed in this virtual environment, and the forces applied to the virtual hand in scenarios designed in the virtual environment can be felt by the user by being feedback onto the exoskeleton. For example, as shown in Figure 2.33, when the virtual hand grasps a virtual round object, it can be applied to the fingers of the user using the outer skeleton to force the linear object to feel as though the virtual object is actually holding it.

As shown in Figure 2.32, there are 3 options for the interface between exoskeleton and virtual environment. SLIDERS option activate the slider bars to change the value for the flexion amount of the virtual fingers and exoskeleton fingers. EMG gets the positions of the exoskeleton fingers via the armband. EXO activate the exoskeleton motors to change the virtual fingers positions. The Calibration menu calibrates the EMG signal values. “Get Rest Value” and “Get Fist Value” logs EMG signals as upper control limit (UCL) and lower control limit (LCL). “ Save and exit” button apply the values to the calibration scripts which LCL and UCL scaled as 0 to 100. According to our algorithm on Figure

2.32 all the input data processed to be between these 0 to 100. Then the fingers flexions amount would send to VR hand (Figure 2.33) and as scaled position reference to electromotor actuators on the exoskeleton.

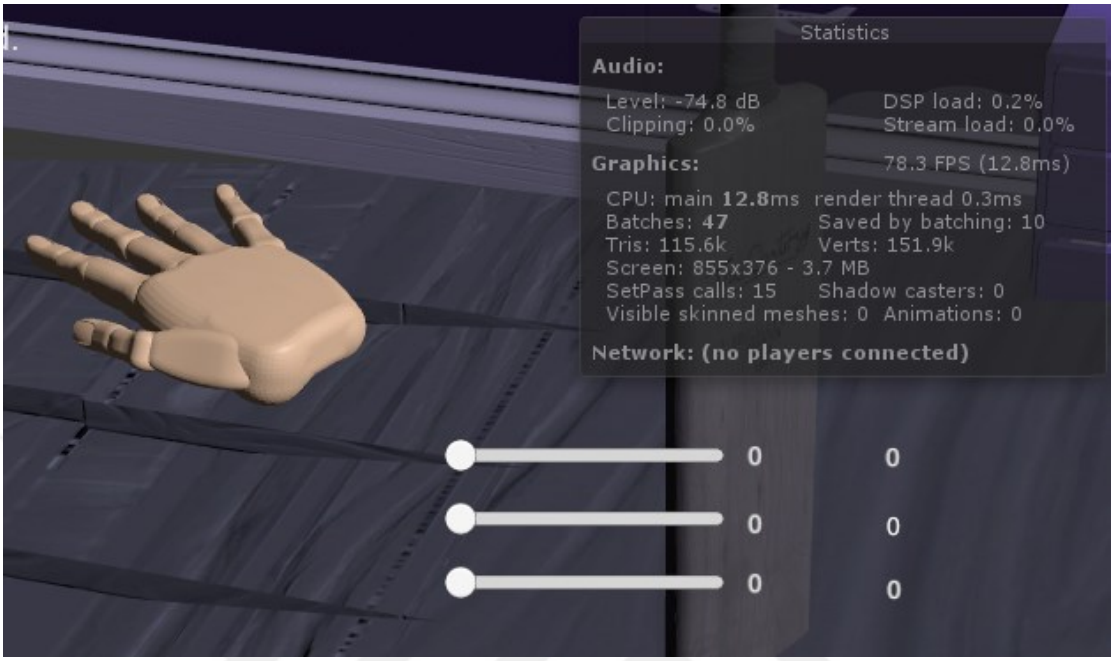


Figure 2.31 Designed virtual environment with the 12.8ms sampling time. The slider bars are inserted as an option to change the value of the flexion amount of the fingers.

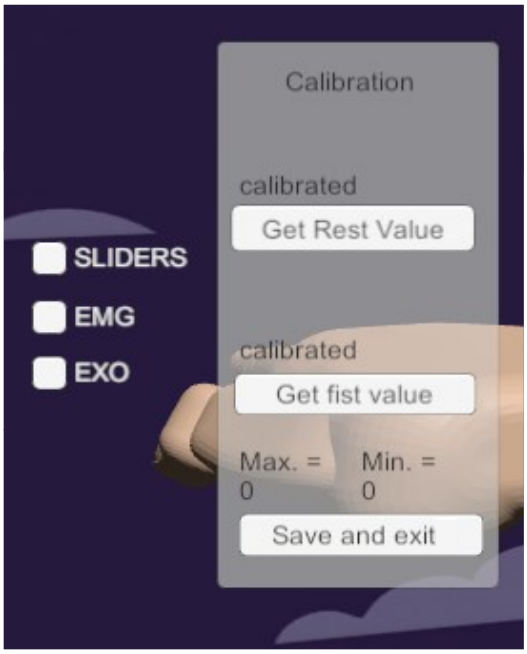


Figure 2.32 Different modes to operate the program

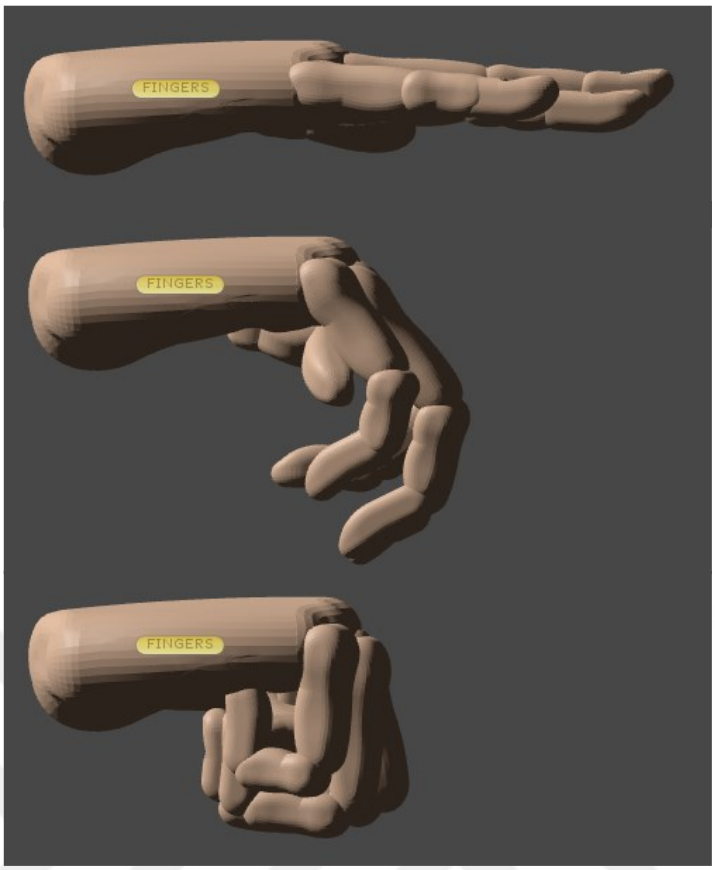


Figure 2.33 The virtual hand is fully open (0% flexion), half open (50% flexion) and fully closed (100% flexion)

As described above, the calculated values are also transferred to the microcontroller of the actuators. The transfer is done via COM port. Because we need to read values from the force and Potentiometer of each actuators we have to open the port before sending data and then close the port to make sure all data transferred completely. So we send data as a pack with a pointer before and after the main data as shown in Figure 2.34 and Figure 2.35. In the Figure 2.35, we used "<" character to show the start of the data and ">" character to inform the end of the data pack to wait to read data from the microcontroller. "Mode of the program" informs the exoskeleton bout the operation mode that described in Figure 2.32 (APPENDIX – G and APPENDIX – H).

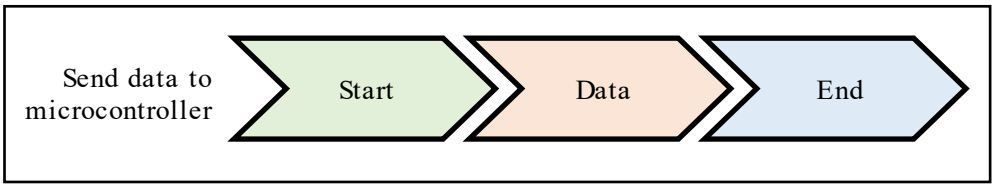


Figure 2.34 The method that data send to the microcontroller of the actuators

<	Byte 1	Mode of the program	>
	Byte 2	Thumb finger flexion amount	
	Byte 3	Index finger flexion amount	
	Byte 4	Middle finger flexion amount	
	Byte 5	Thumb finger collision alert	
	Byte 6	Index finger collision alert	
	Byte 7	Middle finger collision alert	

Figure 2.35 The values that transfer from the C# scripts to the microcontroller of the actuators

2.5 Actuators

The design and selection of the actuators were based on stroke and speed to pull/push the linked bar mechanisms. Linear hydraulic and pneumatic actuators have high power density, but also bulky and cannot be easily controlled as easy as DC motors. DC motors meet the criteria of necessary power with a compact and portable solution for wearable devices. Firgelli L16P-50-63-12-P is a 7.8W micro servo driven DC motor with 63:1 gearbox ratio and roll-screw Linear motion system (Figure 2.36). It also has built-in linear Potentiometer position feedback. More technical details about the motor can be found in APPENDIX – D.

Motors are driven by a PWM (Pulse Width Modulation) servo drive method using the dual full-bridge L298N processor (APPENDIX – E) which shown in Figure 2.36. A 12v/2A adaptor supplies the required power to the driver modules. It's better to notice that the L289n gets maximum 9.8v for 100% duty cycle because of 2.46v inner voltage drop.

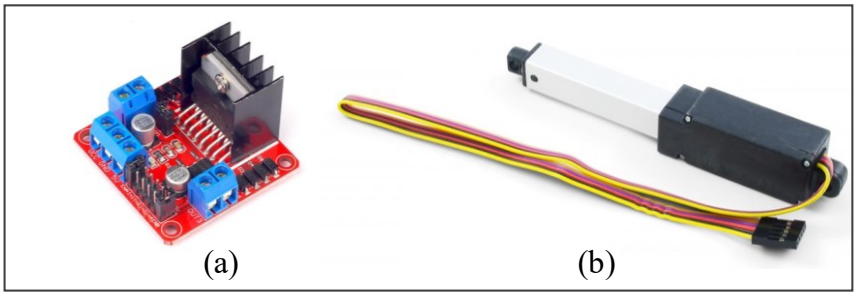


Figure 2.36 Motors and driver used as actuators: a) L298N based motor driver module; b) L16-P miniature linear actuator with feedback

2.6 Controller Unit and Cabling

In order to read the values of the sensors, process and control the actuators via motor drivers, we need a microcontroller unit to do all these mentioned processes and communicate with the computer via USB serial port. Figure 2.37 shows different parts and modules that included in it. A 12 volts 1.5 ampers can supply the required power to drive motors and microcontroller.

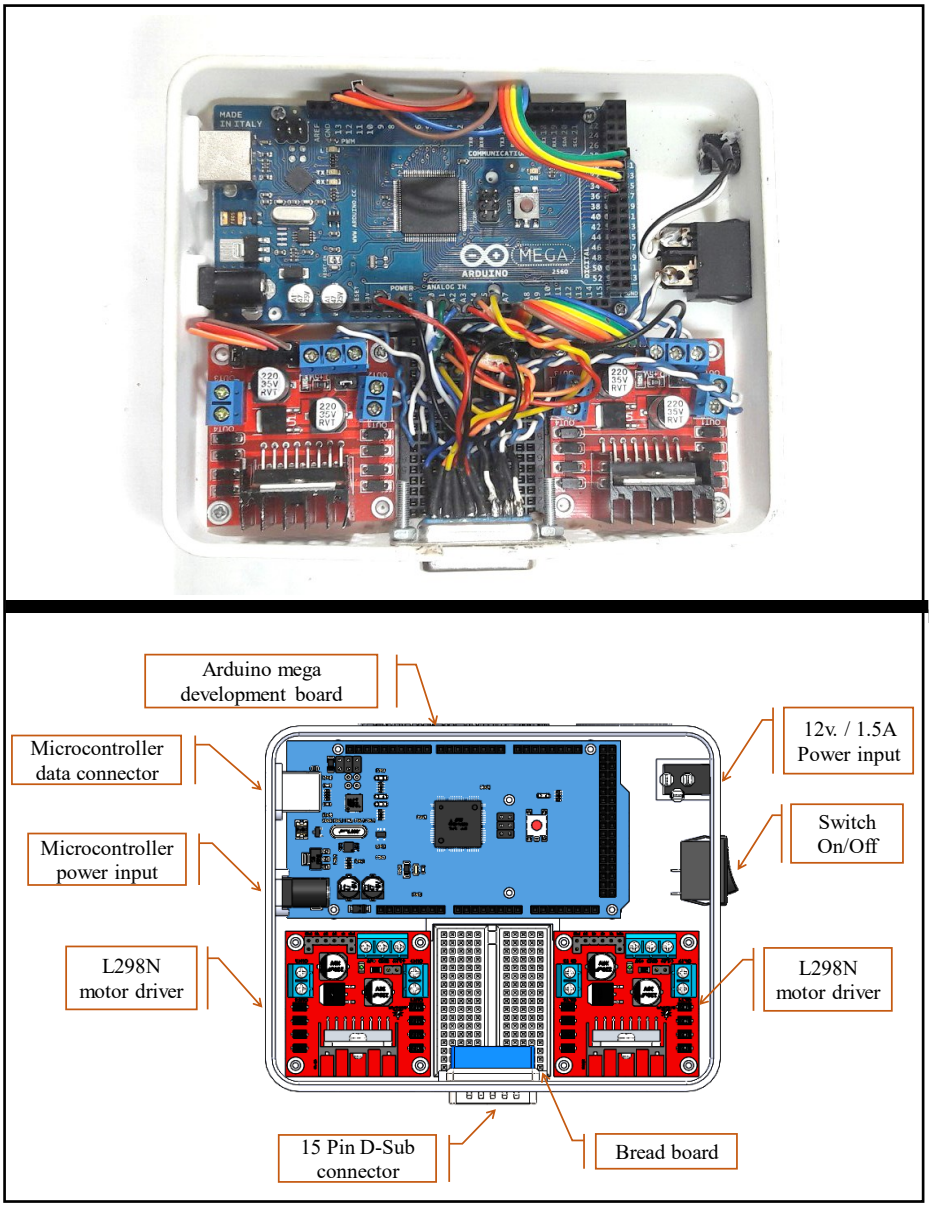


Figure 2.37 General schematic of the controller unit

The Arduino Mega Atmega2560 is a development board based on Atmega328 microcontroller. It has 54 digital input / output pins (14 of which can be used as PWM outputs), 16 analog inputs (with the range of 0.0 to 3.3 volts), 4 UARTs (hardware serial ports), a 16 MHz CPU, a USB connection, a power input, an ICSP connection and a reset

button. We used 5 volts power to derive the board which supplied by one of the motor drivers. Figure 2.38 schemes the connections between different electronic parts. Schematic of the Arduino is available in APPENDIX-F.

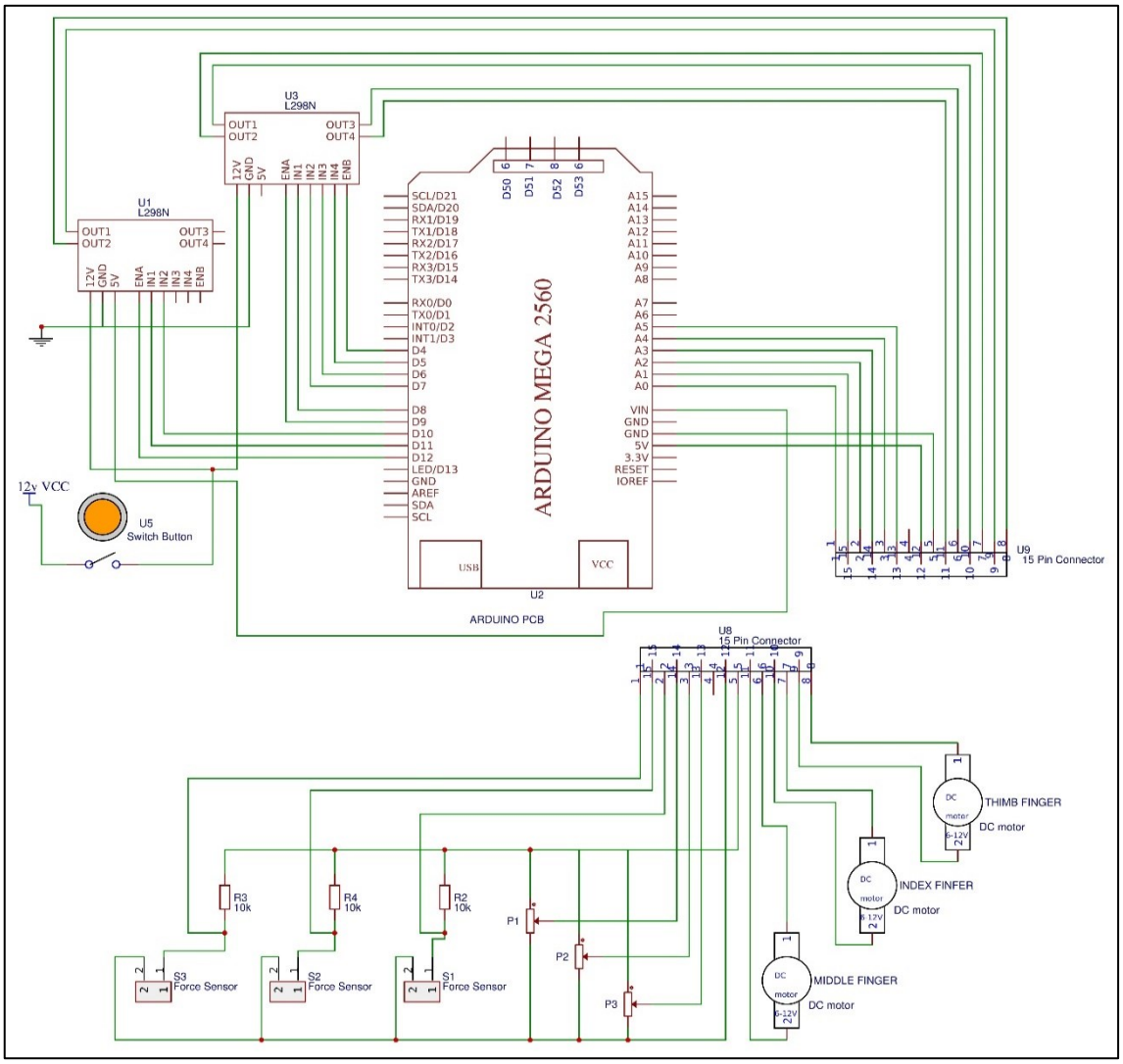


Figure 2.38 Schematic of connections between motors (U6, U7, and U8), potentiometer sensors (P1, P2, and P3), force sensors (S1, S2, and S3), motor drivers (U1, U3), 15 pin connectors and development board (U2)

2.6.1 PID Control Experiment – Tuning the Controller

Control strategies were developed in order to carry out some experiments with the exoskeleton. The control algorithm was developed based on C/C++ scriptings which completely seprated from the computer and all the process done inside the microcontroller. As mentioned before, the exoskeleton collaborates in 3 different operating modes. The aim is to control the speed of the motor to reach the target points. When the system is in Slider or EMG mode, The setpoint defined by the fingers position

which makes exoskeleton to flow the flexion rate of the VR fingers. And although, the EXO mode that activates impedance controller, gets its set point by using the current value of related force sensor (Figure 2.39).

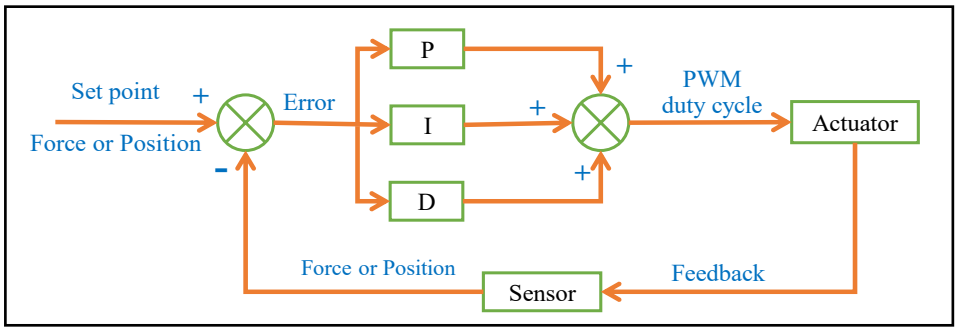


Figure 2.39 General overview of PID controller

PID controller is configured using three parameters acting in the sum to close the loop error – proportional gain (Kp), integral time (Ti), and derivative time (Td), where Kp reduces the steady state error and increasing Kp reduces rise time. Td interprets the change of slope of error changes. Td is the look-ahead time to try to estimate future system behavior and large values of Td create overshoots as it estimates wrongly. Ti eliminates steady-state error and reduces rise time. Ti may improve the response of the system. Tuning the loop is about finding a combination of these three parameters that give an appropriate response to a disturbance, as illustrated in the figure below. On the other hand, it depends on what is controlling and the expectations of the tuning (Figure 2.40). For controlling purpose, overshoot concerned to reduce mechanical damage to the gearbox of the motors with relatively fast and sensitive response.

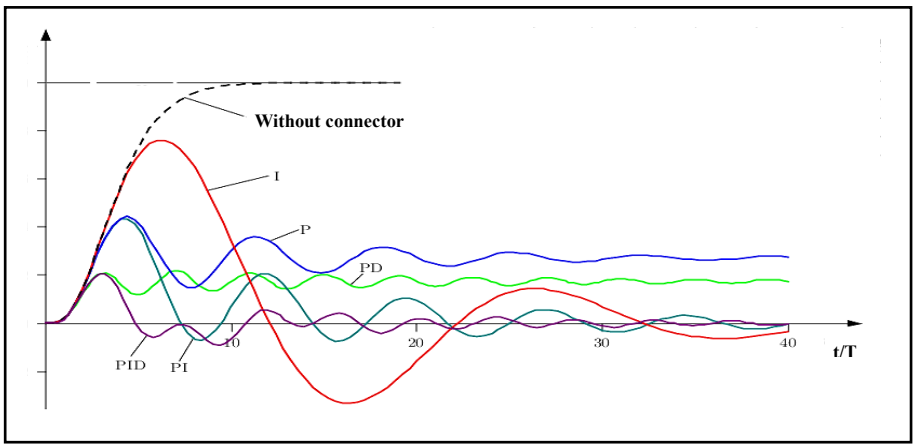


Figure 2.40 Effect of each parameter to PID controller response during time

The Zeigler-Nicols method has been around [35] which rules work well on processes where the dead time is less than half the length of the time is constant. To do this, the basic tuning steps start by tuning all gains to zero. Then, we increase the Kp until the response to a disturbance is steady oscillation which is called the 'ultimate' gain (Ku) and at this steady state measure the 'ultimate' oscillation period (Tu). Ku and Tu can then be used to calculate values for Kp, Ti and Td, depending on the type of control algorithm implemented, according to the table below (taken from [35]). Based on sensors, the PID controller will concentrate to get Kp, Ti and Td values that not exactly but near the "Some overshoot" control type.

Table 2.6 Zeigler-Nicols PID tuning method selection based on ultimate oscillation period (Tu) and gain (Ku)

Control Type	Kp	Ti	Td
P	0.5Ku	–	–
PI	0.45Ku	Tu/1.2	–
PD	0.8Ku	–	Tu/8
Classic PID	0.6Ku	Tu/2	Tu/8
Pessen Integral Rule	0.7Ku	Tu/2.5	3Tu/20
Some overshoot	0.33Ku	Tu/2	Tu/3
No overshoot	0.2Ku	Tu/2	Tu/3

In order to find the Kp, Ki and Kd values for the PID controller, to control motor speed based on force value, the steady oscillation condition of each finger has been tested as shown in Figure 2.41, Figure 2.42 and Figure 2.43.

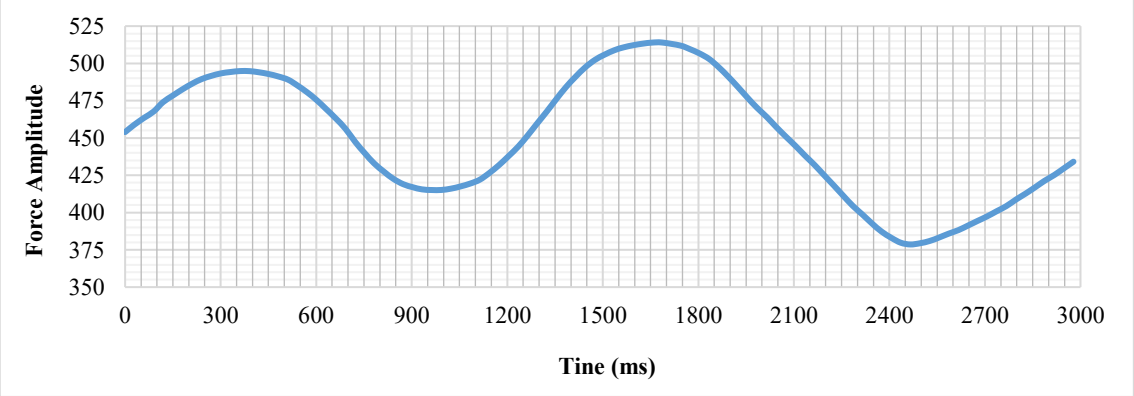


Figure 2.41 Steady oscillation test of force value for tuning motor speed on thumb finger. The ultimate gain (Ku) = 17 and Tu = 1.36 seconds

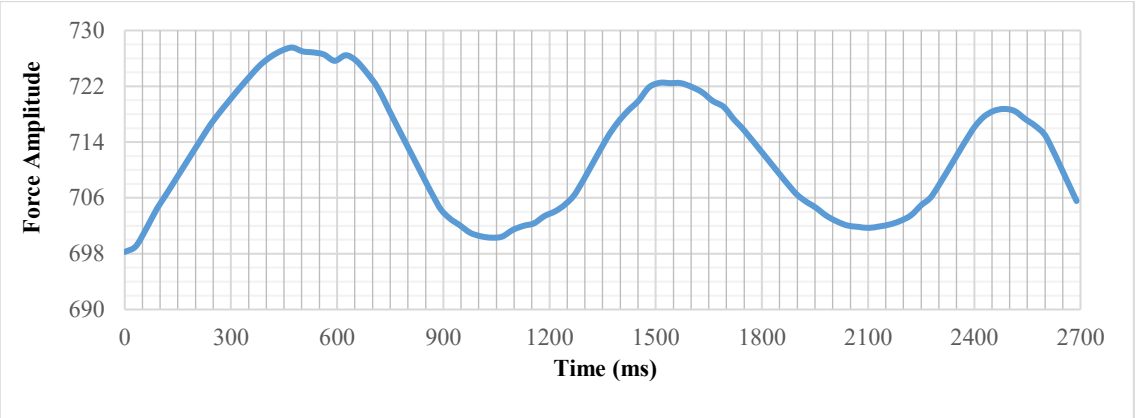


Figure 2.42 Steady oscillation test of force value for tuning motor speed on the index finger. The ultimate gain (Ku) = 60 and Tu = 1.03 seconds

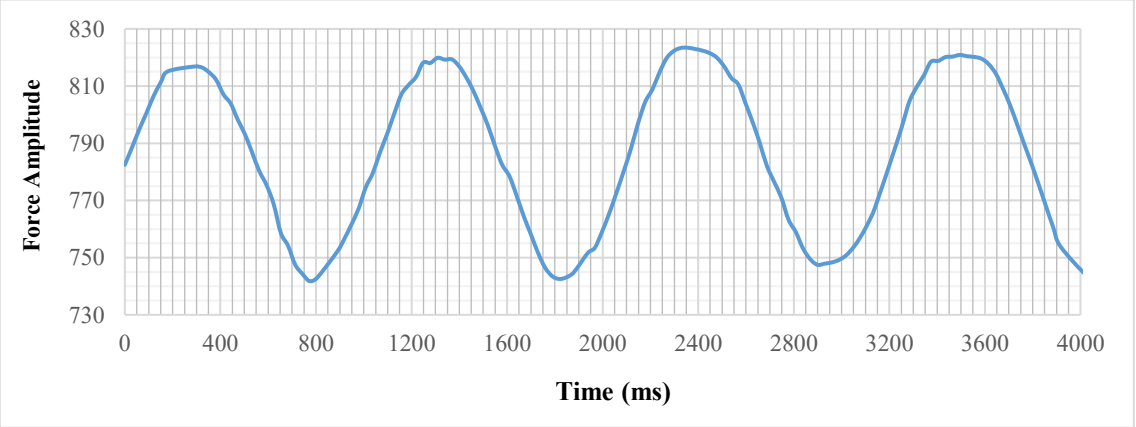


Figure 2.43 Steady oscillation test of force value for tuning motor speed on the middle finger. The ultimate gain (Ku) = 20 and Tu = 1.16 seconds

Table 0.7 estimated Kp, Ti and Td values for force impedance controller by Zeigler-Nicols PID tuning method based on ultimate oscillation period (Tu) and gain (Ku)

	Ku	Tu	Kp = 0.33 Ku	Ti = Tu/2	Td= Tu/3
Thumb motor	17	1.36	5.61	0.68	0.453
Index motor	60	1.03	19.80	0.515	0.343
Middle motor	20	1.16	6.6	0.58	0.386

Moreover, for tuning the PID values in order to control motor speed based on location feedback, the steady-state condition of each finger measured as shown in Figure 2.44, Figure 2.45 and Figure 2.46.

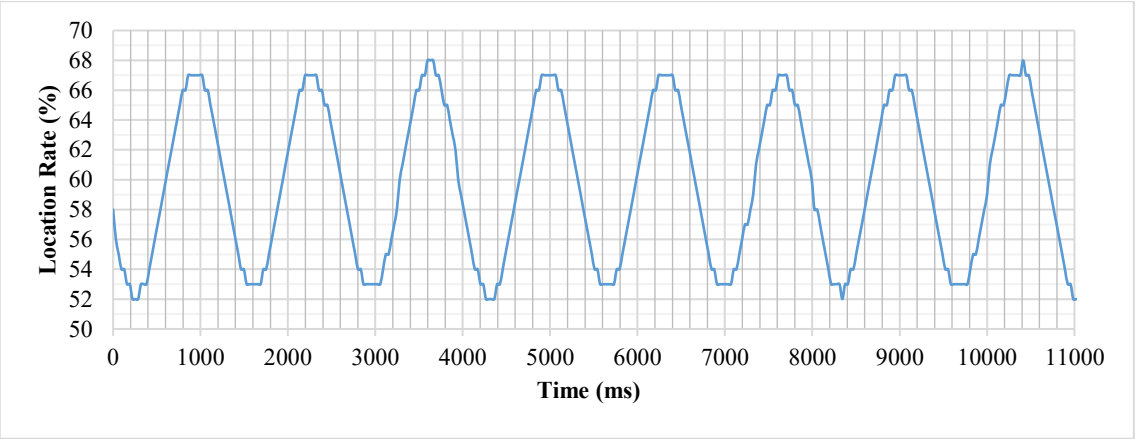


Figure 2.44 Steady oscillation test of location feedback for tuning motor speed on thumb finger. The ultimate gain (Ku) = 50 and Tu = 1.33 seconds

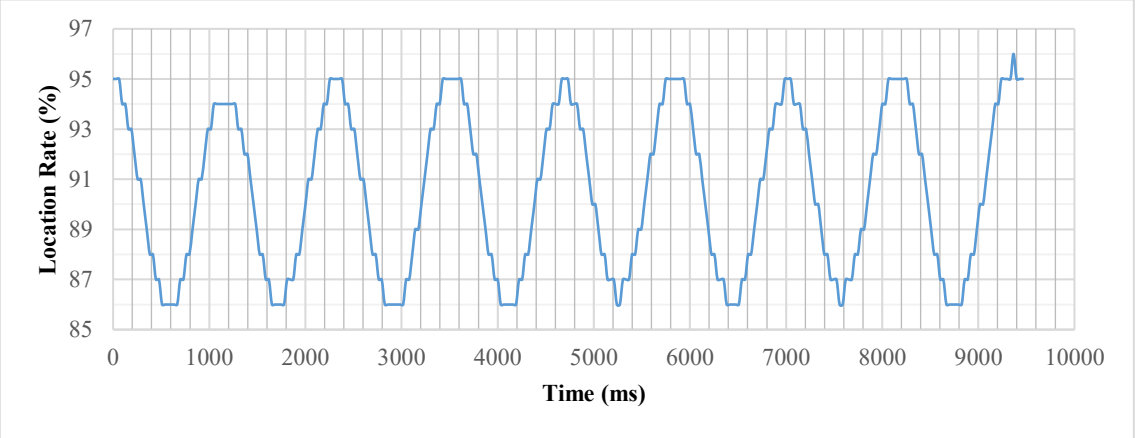


Figure 2.45 Steady oscillation test of location feedback for tuning motor speed on the index finger. The ultimate gain (Ku) = 80 and Tu = 1.14 seconds

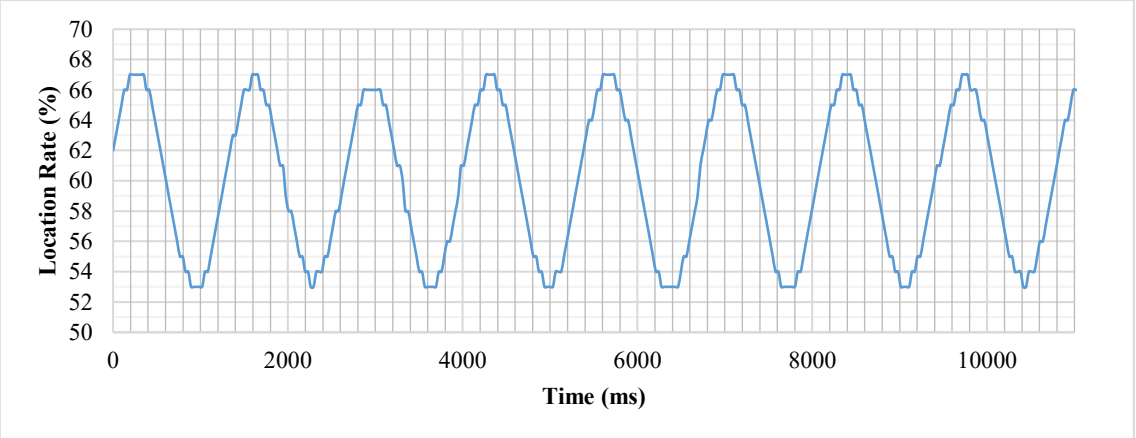


Figure 2.46 Steady oscillation test of location feedback for tuning motor speed on the middle finger. The ultimate gain (Ku) = 50 and Tu = 1.27 seconds

Table 2.8 Estimated Kp, Ti and Td values for location feedback controller by Zeigler-Nicols PID tuning method based on ultimate oscillation period (Tu) and gain (Ku)

	Ku	Tu	Kp = 0.33 Ku	Ti = Tu/2	Td = Tu/3
Thumb motor	50	1.33	16.5	0.665	0.443
Index motor	80	1.14	26.4	0.57	0.380
Middle motor	50	1.27	16.5	0.635	0.423

CHAPTER 3

EXPERIMENTS AND THE RESULTS

The completed version of the exoskeleton is available in Figure 3.1. An experiment was carried out using trajectory control with the previous PID values on a hand of the healthy person (Figure 3.3). The objective was to evaluate if the exoskeleton behavior interacts with the virtual hand. According to a couple of PID controllers that tuned before, we concentrate to get the behavior of exoskeleton during the mentioned modes.

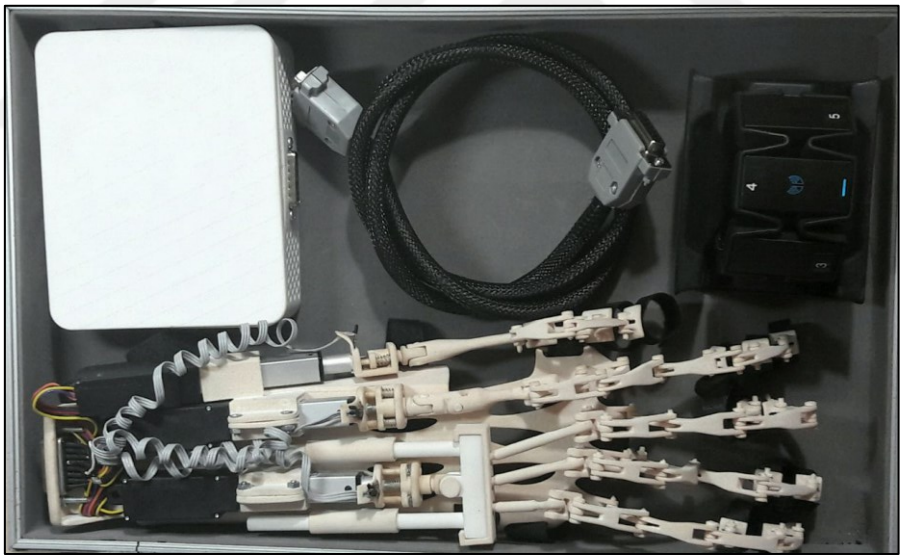


Figure 3.1 completed version of the exoskeleton which contains Controller case, cable, EMG armband, exoskeleton and the virtual environment

In the VE, when we select the EMG or EXO mode, the exoskeleton became passive to The VR hand. So, it gets the position rate of each finger as a set point value between 0 and 100 from the position of the related VR finger. In the current experiment which shown in Figure 3.2, Figure 3.4 and Figure 3.5, we assumed that each finger to catch the position rate of 70, which means all fingers of hand get the 70% of full flexion same as the fingers of the VR hand. Before running the actuator, all the fingers had the position rate of 32.

The Setting time for the middle finger actuator is 460ms, which is 750ms for the thumb finger actuator and 630ms for the middle finger actuator. The time difference compensated by the mechanical mechanism. Some vibrations observed at a lower speed for PWM duty cycle of the actuator which is because of mechanical frictions and mechanical clearances inside the gearbox of the actuators. These vibrations are not sensible by the operator. Moreover, their dead band values for all three actuators obtained ±1.5%.

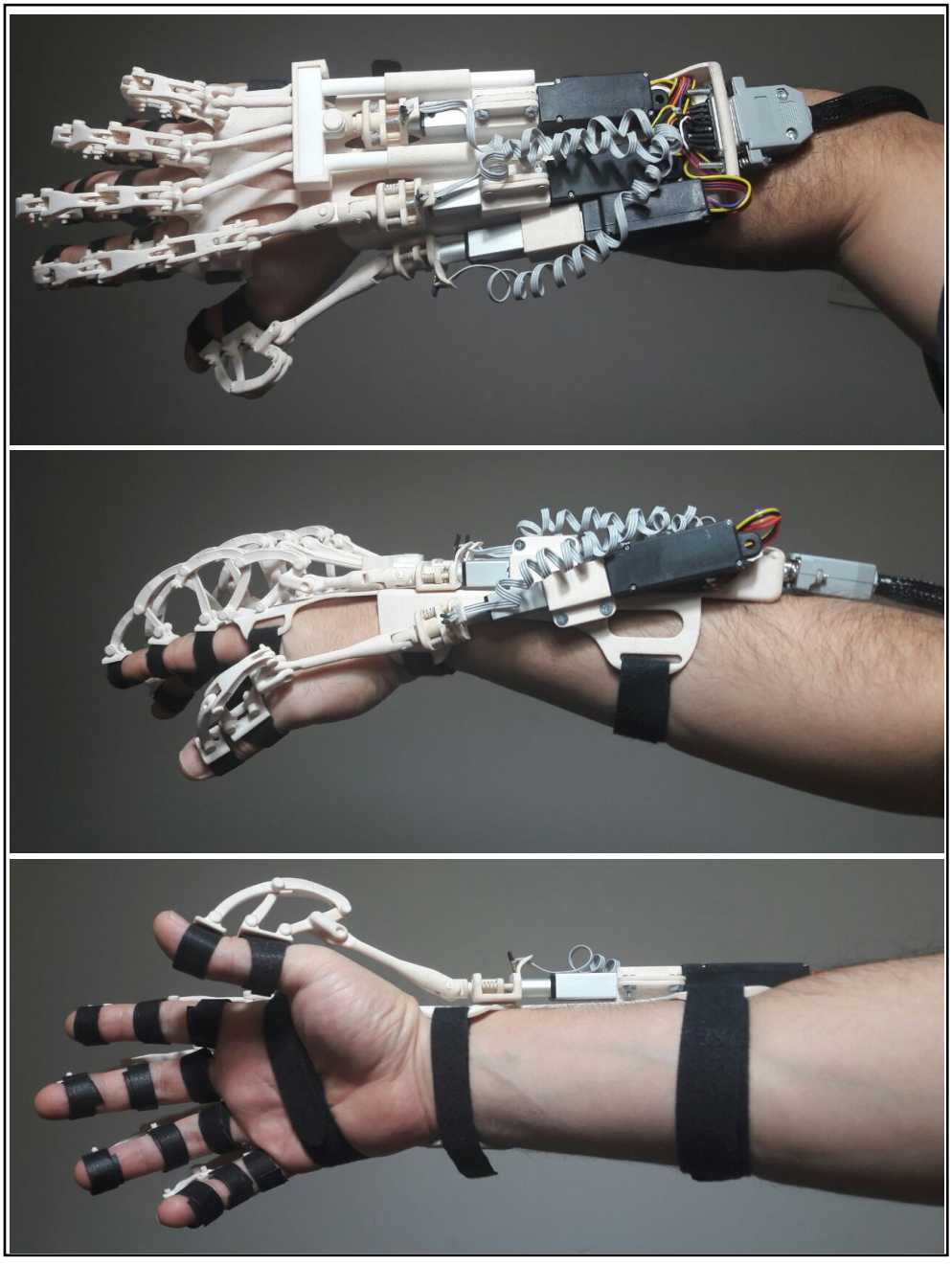


Figure 3.2 Different views of the exoskeleton mounted on hand.

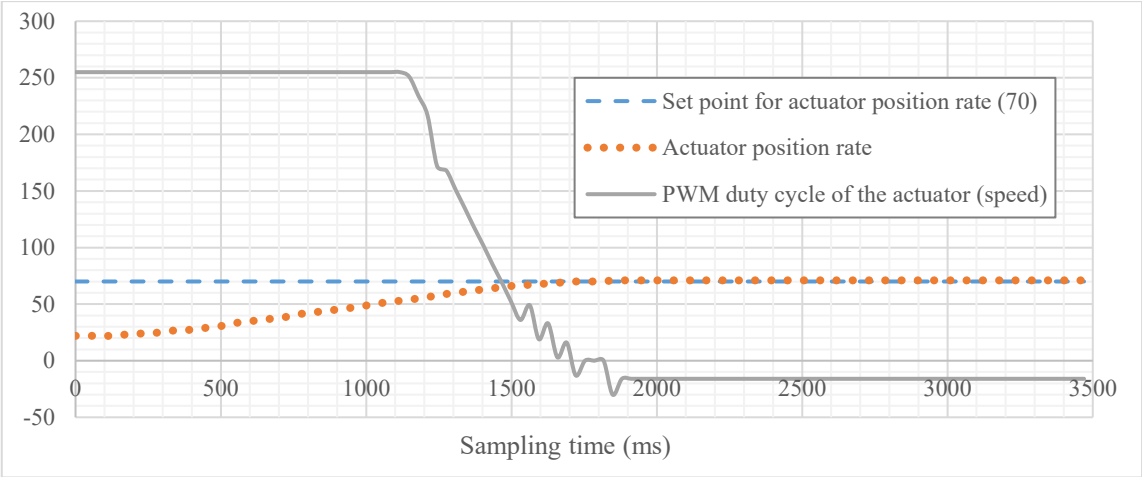


Figure 3.3 Response of the actuator on middle finger to the position rate change from 22 to 70. The set point values directed by the position of the middle finger of VR hand.

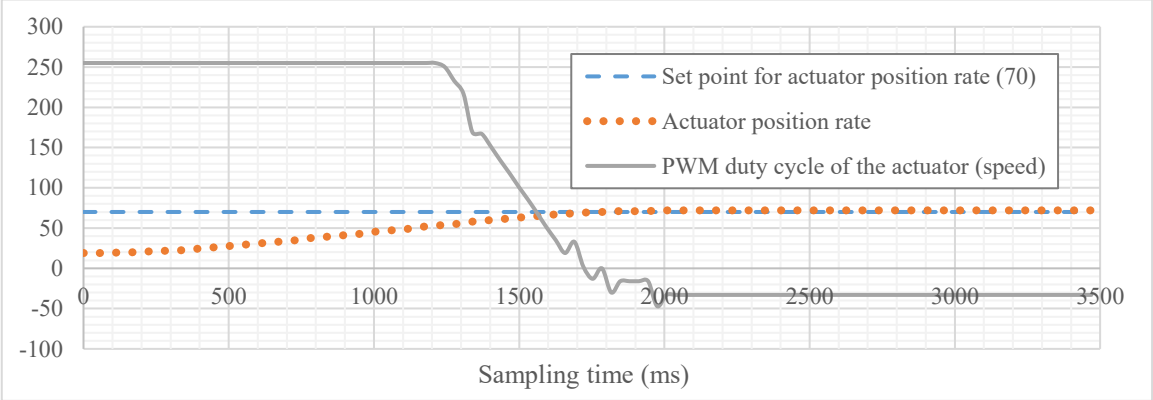


Figure 3.4 Response of the actuator on thumb finger to the position rate change from 19 to 70. The set point values directed by the position of the thumb finger of VR hand.

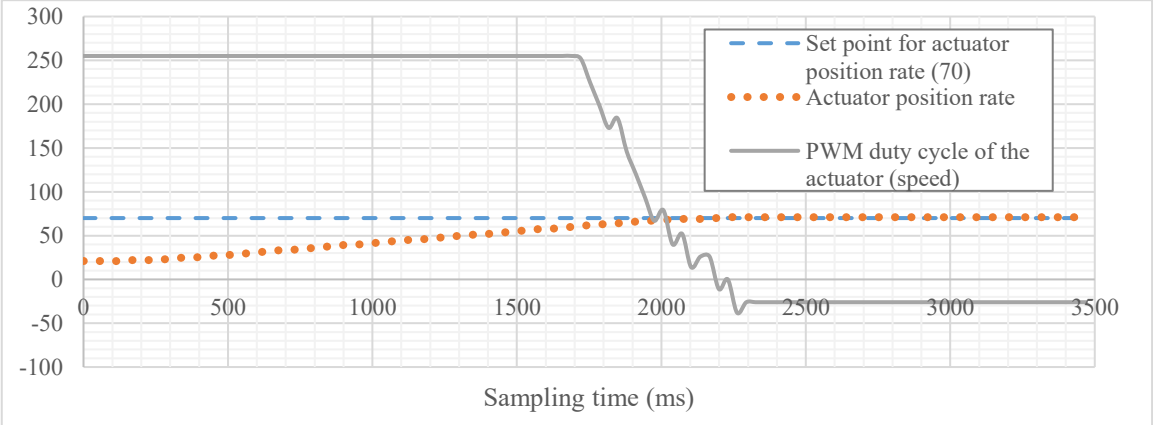


Figure 3.5 Response of the actuator on index finger to the position rate change from 21 to 70. The set point values directed by the position of the index finger of VR hand.

Figure 3.6, Figure 3.7 and Figure 3.8, present the results of an experiment to evaluate the response of impedance controller. They show how force sensor output variation from its set point can be compensated by changing actuator position rate. It means, during the

flexion/extension of the fingers, the variation of the force that applies to the force sensor, effects PWM duty cycle to run the actuator. So the actuator position rate changes and flows the trajectory of the fingers. During the extension a time delay is observable on actuator position. It occurs because of the backlash effects in the force sensing mechanism. Figure 3.6, shows the impedance control test during a flexion/extension of the thumb finger. Firstly, flexion applied at 6.5th second which decreased the force sensor output value. Depending the magnitude of the value, PID controller effects PWM duty cycle to run the actuator. Thus, the actuator moves forward from its primary position rate of 20 to 80 to flow the trajectory of the fingers. Also, extension applied at 16.3th second which increased the force sensor output value. The actuator moves backward from its primary position rate of 80 to 5 to flow the trajectory of the fingers. The backlash effects time delay ~450ms. Figure 3.7, shows the impedance control test during the flexion/extension of the index finger. Firstly, flexion applied at 4.0th second which decreased the force sensor output value. Depending the magnitude of the value, PID controller effects PWM duty cycle to run the actuator. Thus, the actuator moves forward from its primary position rate of 13 to 89 to flow the trajectory of the fingers. Also, extension applied at 11.5th second which increased the force sensor output value. The actuator moves backward from its primary position rate of 89 to 15 to flow the trajectory of the fingers. The backlash effects time delay ~350ms. Figure 3.8 shows the impedance control test during the flexion/extension of the middle finger. Firstly, flexion applied at 4.6th second which decreased the force sensor output value. Depending the magnitude of the value, PID controller effects PWM duty cycle to run the actuator. Thus, the actuator moves forward from its primary position rate of 12 to 77 to flow the trajectory of the fingers. Also, extension applied at 14.2th second which increased the force sensor output value. The actuator moves backward from its primary position rate of 77 to 25 to flow the trajectory of the fingers. The backlash effects time delay ~450ms.

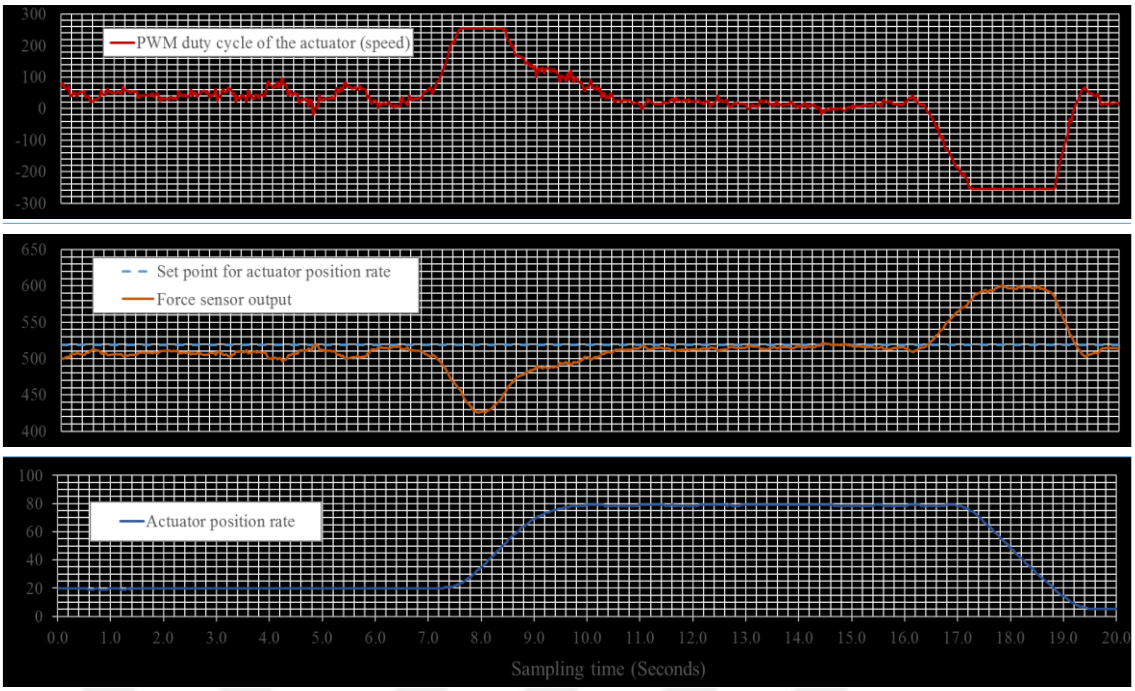


Figure 3.6 Impedance control expriment during the flexion/extension of the thumb finger. The flexion applied at 6.5th second and the extension applied at 16.3th second.

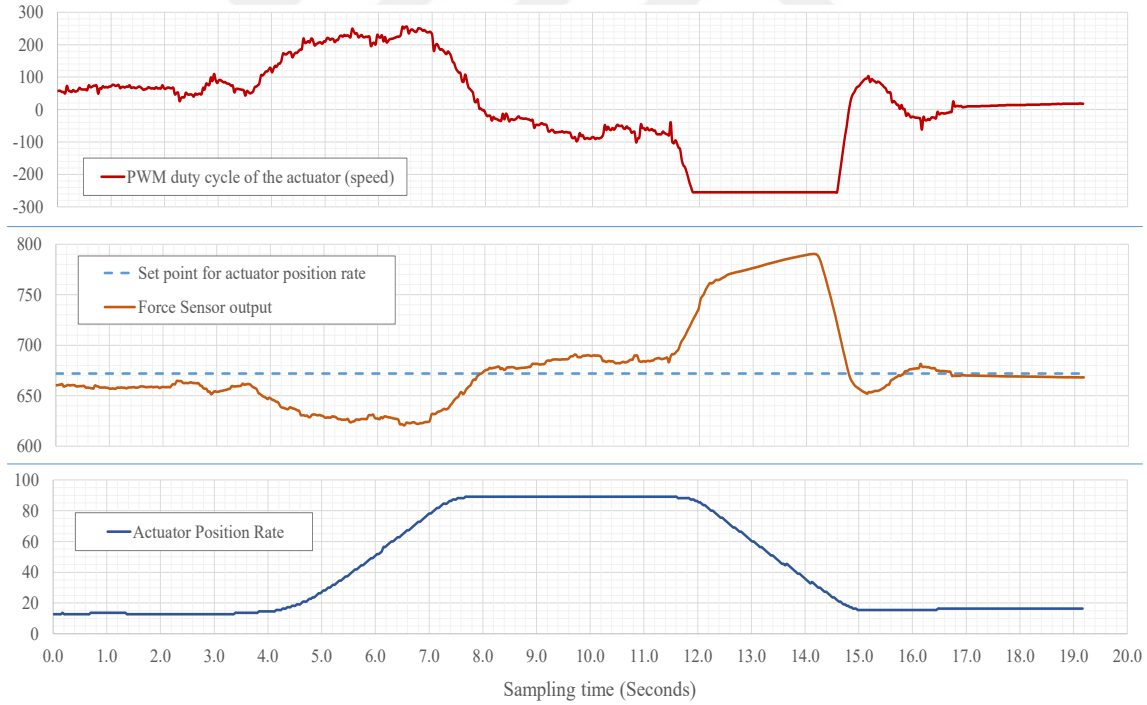


Figure 3.7 Impedance control expriment during the flexion/extension of the index finger. The flexion applied at 4.0th second and the extension applied at 11.5th second.

Figure 3.8 Impedance control expriment during the flexion/extension of the middle finger. The flexion applied at 4.6th second and the extension applied at 14.2th second.

CHAPTER 4

CONCLUSIONS AND FUTURE WORKS

4.1 Conclusions

A multi-functional human-machine interface was designed to assist fingers flexion and extension motions. The proposed exoskeleton system may help and improve the rehabilitation progress of hand fingers. This system was designed with 2 degrees of freedom (DOF) for the thumb finger and 3 DOF for the other fingers actuated via direct-driven linear DC actuators, which had PID position and force controllers. An electromyogram (EMG) armband with a built-in gyroscope and accelerometer unit supplies EMG and hand orientation signals to synchronize the exoskeleton hand and fingers with the ones designed in the three-dimensional virtual environment.

In addition, this exoskeleton system has also a novel force sensing mechanism designed to sense fingertip forces and also help the motors freely follow human finger movements with a minimal resistance in its virtual reality applications.

A contact detection method was developed and applied to sense the 3 dimensional (3D) objects in the virtual environment as force feedback. Fundamental experimental studies of the proposed exoskeleton mechanism on a healthy person were conducted to obtain position and force data for some virtual reality applications. The delay between the mechanical and the virtual system was determined as 13 milliseconds, which was not significantly felt by the user.

In each part of the project, we encountered a number of problems that require completion of further research. In mechanical parts, most problems due to friction between the joints.

Also, the clearance between the joints between links was another source of mechanical problems. Most of the virtual environment problems resulted from the latency of the computer processor. However, more filtering is also required to smooth noisy data which directly effects time delay. Using servo linear actuators, in spite of providing sufficient forces, it limits the flexion speed of fingers. Because of the financial constraints, it was required to continue the project with existing actuators.

Regarding the mentioned problems, the studies showed that the position and force feedback controllers suitably worked for both sensing the 3D virtual objects and for the mechanism's following the user fingers' movements with minimal resistance. In conclusion, such a mechanism with proposed special capabilities could be utilized for various applications such as finger rehabilitation, robotic hand telemanipulation, support devices for hand movements and also haptic training applications.

In stroked hands, wearable robots do the task for moving the patient's fingers in a normal pattern, usually in an assist-as-needed paradigm. This means that the exoskeleton should only apply the required force to complete the patient's movement. This approach can lead to an increase in physical effort while rehabilitation can be more effective in motor learning. Different types of the therapy can also lead to some faster recovery from fingers motor impairment. Which needs specific clinical studies.

4.2 Future Works

As a multi-functional human-machine interface, our project opens a variety of subjects for further researches. The project itself has lots of things to be developed.

As mentioned before, the mechanical mechanism would be redesigned to reduce frictions between links. I suggest using fine surface metal-plastic slip joints instead of plastic-plastic joints. It may reduce friction factor and clearances. We used 2.4GHz Intel CORE i7 as CPU and an Arduino mega for the controller unit. We suggest to use faster CPU for the future researches and use a controller unit which is able to do multi-thread functions. Furthermore, using more high-quality sensors and cables and different data transfer techniques may reduce the noises. It may help to need less smoothing procedure and less time delay. About the servo linear actuators, I would say more speed and less force would be more suitable for the mechanism. The control algorithm of the actuators is also needed to be optimized.

I suggest that the clustering of the EMG armband signals can be a separate field. It may help to recognize any flexion rate for each finger. The 3D environment could be developed to detect dynamic force feedback whid more integration with the hand. Augmented reality environment also can be added to this project.

furthermore, telemanipulation of a robotic hand with haptic sensation can be achievable. Which may be used to control a hand in a dangerous environment.

REFERENCES

[1] Romero, D., Stahre, J., Wuest, T., Norman, O., Bernus, P., Fast-Berglund, Å., and Gorecky, D., (2016). "Towards an Operator 4.0 Typology: A Human-Centric Perspective on the Fourth Industrial Revolution Technologies", International Conference on Computers & Industrial Engineering (CIE46) , 1-11.

[2] Dick, J. and Crapuchettes, B., (2000). Servo-assisted lower-body exoskeleton with a true running gait, DARPA Workshop on Exoskeletons for Human Performance Augmentation.

[3] Jansen, J.F., (2000). Exoskeleton for soldier enhancement systems feasibility study (No. ORNL/TM-2000/256), ORNL, 28 Sep 2000.

[4] Naidu, D., (1992). Scientific Fundamentals of Robotics 7: Biped Locomotion, Dynamics, Stability, Control and Application-M., Vukobratovic.

[5] Youngblut, C., Johnston, R.E., Nash, S.H., Wienclaw, R.A., and Will, C.A. (1996). Review of Virtual Environment Interface Technology (No. IDA-P-3186), Institute For Defense Analyses Alexandria Va.

[6] Bowman, D.A., and Hodges, L.F., (1995).User interface constraints for immersive virtual environment applications, Georgia Institute of Technology.

[7] Rizzo, A.S., and Kim, G.J., (2005). "A SWOT analysis of the field of virtual reality rehabilitation and therapy", Presence: Teleoperators & Virtual Environments, 14(2): 119-146.

[8] Srinivasan, M.A., and Basdogan, C., (1997). "Haptics in virtual environments: Taxonomy, research status and challenges", Computers & Graphics, 21(4): 393-404.

[9] Atlihan, M., Akdoğan, E., and Arslan, M.S., (2014). "Development of a therapeutic exercise robot for wrist and forearm rehabilitation", International Conference in Methods and Models in Automation and Robotics (MMAR), IEEE, 2014 Sep 2, 52-57.

[10] Ott, R., De Perrot, V., Thalmann, D., and Vexo, F., (2007). "MHaptic: a haptic manipulation library for generic virtual environments", International Conference in Cyberworlds, IEEE, 2007, 338-345.

[11] Zhou, M., and Ben-Tzvi, P., (2015). "RML glove—An exoskeleton glove mechanism with haptics feedback", IEEE/ASME Transactions on Mechatronics, 20: 641-652.

[12] Lelieveld, M., and Maeno, T., (2006). “Design and development of a 4 DOF portable haptic interface with multi-point passive force feedback for the index finger”, International Conference in Robotics and Automation, Proceedings 2006 IEEE, 3134-3139.

[13] Garcia-Hernandez, N., Sarakoglou, I., Tsagarakis, N., and Caldwell, D., (2014). Under-actuated hand exoskeleton with novel kinematics for potential use in rehabilitation, EuroHaptics, 2014.

[14] Avizzano, C.A., Bargagli, F., Frisoli, A., and Bergamasco, M., (2000). “The hand force feedback: analysis and control of a haptic device for the human-hand”, International Conference in Systems, Man, and Cybernetics, IEEE, 2(1):989-994.

[15] Amirabdollahian, F., Ates, S., Basteris, A., Cesario, A,. Buurke, J., Hermens, H., Hofs, D., Johansson, E., Mountain, G., and Nasr, N., (2014). “Design, development and deployment of a hand/wrist exoskeleton for home-based rehabilitation after stroke-SCRIPT project”, Robotica, 32: 1331-1346.

[16] Letier, P., Avraam, M., Horodinca, M., Veillerette, S., Verschueren, J.P., Fautre, T., Motard, E. Ilzkovitz, M., Schiele, A., and Preumont, A., EXOSTATION Phase B: 7-DOF Haptic Exoskeleton, Penn State University.

[17] Mulas, M., Folgheraiter, M., and Gini, G., (2005). “An EMG-controlled exoskeleton for hand rehabilitation”, 9th International Conference on Rehabilitation Robotics, ICORR 2005, IEEE, 371-374.

[18] Kiguchi, K., and Hayashi, Y., (2012). “An EMG-based control for an upper-limb power-assist exoskeleton robot”, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 42(4): 1064-1071.

[19] Accogli, A., Grazi, L., Crea, S., Panarese, A., Carpaneto, J., Vitiello, N., and Micera, S., (2017). “EMG-Based Detection of User’s Intentions for Human-Machine Shared Control of an Assistive Upper-Limb Exoskeleton”, Wearable Robotics: Challenges and Trends, Springer, 181-185.

[20] Sarasola-Sanz, A., López-Larraz, E., Irastorza-Landa, N., Klein, J., Valencia, D., Belloso, A., Morin, F., Spüler, M., Birbaumer, N., and Ramos-Murguialday, A., (2017). “An EEG-Based Brain-Machine Interface to Control a 7-Degrees of Freedom Exoskeleton for Stroke Rehabilitation”, Converging Clinical and Engineering Research on Neurorehabilitation II, Springer, 1127-1131.

[21] Veneman, J.F., Kruidhof, R., Hekman, E.E., Ekkelenkamp, R., Van Asseldonk, E.H., and Van Der Kooij, H., (2007). “Design and evaluation of the LOPES exoskeleton robot for interactive gait rehabilitation”, IEEE Transactions on Neural Systems and Rehabilitation Engineering, 15(3): 379-386.

[22] Zschorlich, V.R., (1989). “Digital filtering of EMG-signals”, Electromyogr Clin Neurophysiol, 29(2): 81-86.

[23] Park, S.H., Lee, S.P., (1998). “EMG pattern recognition based on artificial intelligence techniques”, IEEE transactions on Rehabilitation Engineering, 6(4): 400-405.

[24] Althloothi, S., Mahoor, M.H., Zhang, X., and Voyles, R.M., (2014). “Human activity recognition using multi-features and multiple kernel learning”, Pattern recognition, 47(5): 1800-1812.

[25] Plamondon, R., O'reilly, C., Galbally, J., Almaksour, A., and Anquetil, É., (2014). "Recent developments in the study of rapid human movements with the kinematic theory: Applications to handwriting and signature synthesis", Pattern Recognition Letters, 35: 225-235.

[26] Krebs, H.I., Volpe, B., Aisen, M., and Hogan, N., (2000). "Increasing productivity and quality of care: Robot-aided neuro-rehabilitation", Journal of rehabilitation research and development, 37(6): 639-652.

[27] Hidler, J., Nichols, D., Pelliccio, M., and Brady, K., (2005). "Advances in the understanding and treatment of stroke impairment using robotic devices", Topics in stroke rehabilitation, 12: 22-35.

[28] Banala, S.K., Kim, S.H., Agrawal, S.K., and Scholz, J.P., (2009). "Robot assisted gait training with active leg exoskeleton (ALEX)", IEEE Transactions on Neural Systems and Rehabilitation Engineering, 17(1): 2-8.

[29] Rohling, R., (1993). Optimized fingertip mapping and hand modeling for telemanipulation, PhD diss., McGill University.

[30] Stergiopoulos, P., Fuchs, P., and Laurgeau, C., (2003). Design of a 2-finger hand exoskeleton for VR grasping simulation, Eurohaptics, Dublin, Ireland: 80-93.

[31] ThalmicLabsInc., https://www.myo.com/techspecs , 15 March 2017.

[32] Jung, P.G., Lim, G., Kim, S., and Kong, K., (2015). "A wearable gesture recognition device for detecting muscular activities based on air-pressure sensors", IEEE Transactions on Industrial Informatics, 11(2): 485-494.

[33] Mannion, P., and Myo armband, Wearables design focuses on packaging: http://www.edn.com/design/systems-design/4442290/Myo-armband--earables-design-focuses-on-packaging , 05 May 2016.

[34] Chan, A.D., and Green, G.C., (2017). "Myoelectric control development toolbox", CMBES Proceedings, 30(1).

APPENDIX-A

3D PRINTING MATERIAL

e-Manufacturing Solutions

PA 2200 Top Quality 1.0
PA12 EOS GmbH - Electro Optical Systems

Product Texts

This whitish fine powder PA 2200 on the basis of polyamide 12 serves with its very well-balanced property profile a wide variety of applications. Laser-sintered parts made from PA 2200 possess excellent material properties:

- high strength and stiffness
- good chemical resistance
- excellent long-term constant behaviour
- high selectivity and detail resolution
- various finishing possibilities (e.g. metallisation, stove enamelling, vibratory grinding, tub colouring, bonding, powder coating, flocking)
- bio compatible according to EN ISO 10993-1 and USP/level VI/121 °C
- approved for food contact in compliance with the EU Plastics Directive 2002/72/EC (exception: high alcoholic foodstuff)

Typical applications of the material are fully functional plastic parts of highest quality. Due to the excellent mechanical properties the material is often used to substitute typical injection moulding plastics. The biocompatibility allows its use e.g. for prostheses, the high abrasion resistance allows e.g. the realisation of movable part connections.

60 µm layer thickness
The parameter set TopQuality is used for very small to medium-sized parts with extremely fine, fragile geometries and geometric elements and the strictest requirements in surface quality are best served by this parameter set. It applies a layer thickness of 60 µm, which is approximately the thickness of a grain of the plastic powder normally used today. The typical stair-step effect on upward and downward-pointing geometry elements can practically no longer be seen on TopQuality parts. The mechanical attributes of TopQuality parts are satisfyingly close to the levels of Performance parts.

Mechanical properties	Value	Unit	Test Standard
Izod Impact notched (23°C)	**4.4**	kJ/m²	ISO 180/1A
Shore D hardness (15s)	**75**	-	ISO 868

3D Data	Value	Unit	Test Standard
The properties of parts manufactured using additive manufacturing technology (e.g. laser sintering, stereolithography, Fused Deposition Modelling, 3D printing) are, due to their layer-by-layer production, to some extent direction dependent. This has to be considered when designing the part and defining the build orientation.			
Tensile Modulus (X Direction)	**1800**	MPa	ISO 527-1/-2
Tensile Modulus (Y Direction)	**1800**	MPa	ISO 527-1/-2
Tensile Modulus (Z Direction)	**1750**	MPa	ISO 527-1/-2
Tensile Strength (X Direction)	**52**	MPa	ISO 527-1/-2
Tensile Strength (Y Direction)	**52**	MPa	ISO 527-1/-2
Tensile Strength (Z Direction)	**52**	MPa	ISO 527-1/-2
Strain at break (X Direction)	**20**	%	ISO 527-1/-2
Strain at break (Y Direction)	**20**	%	ISO 527-1/-2
Strain at break (Z Direction)	**7**	%	ISO 527-1/-2
Charpy impact strength (+23°C, X Direction)	**53**	kJ/m²	ISO 179/1eU
Charpy notched impact strength (+23°C, X Direction)	**4.8**	kJ/m²	ISO 179/1eA
Flexural Modulus (23°C, X Direction)	**1500**	MPa	ISO 178

Thermal properties	Value	Unit	Test Standard
Melting temperature (10°C/min)	**176**	°C	ISO 11357-1/-3
Vicat softening temperature (50°C/h 50N)	**163**	°C	ISO 306

Other properties	Value	Unit	Test Standard
Density (lasersintered)	**930**	kg/m³	EOS Method

Characteristics

PA 2200 Top Quality 1.0
PA12 EOS GmbH - Electro Optical Systems

Processing

Laser Sintering

Delivery form

White

Chemical Resistance

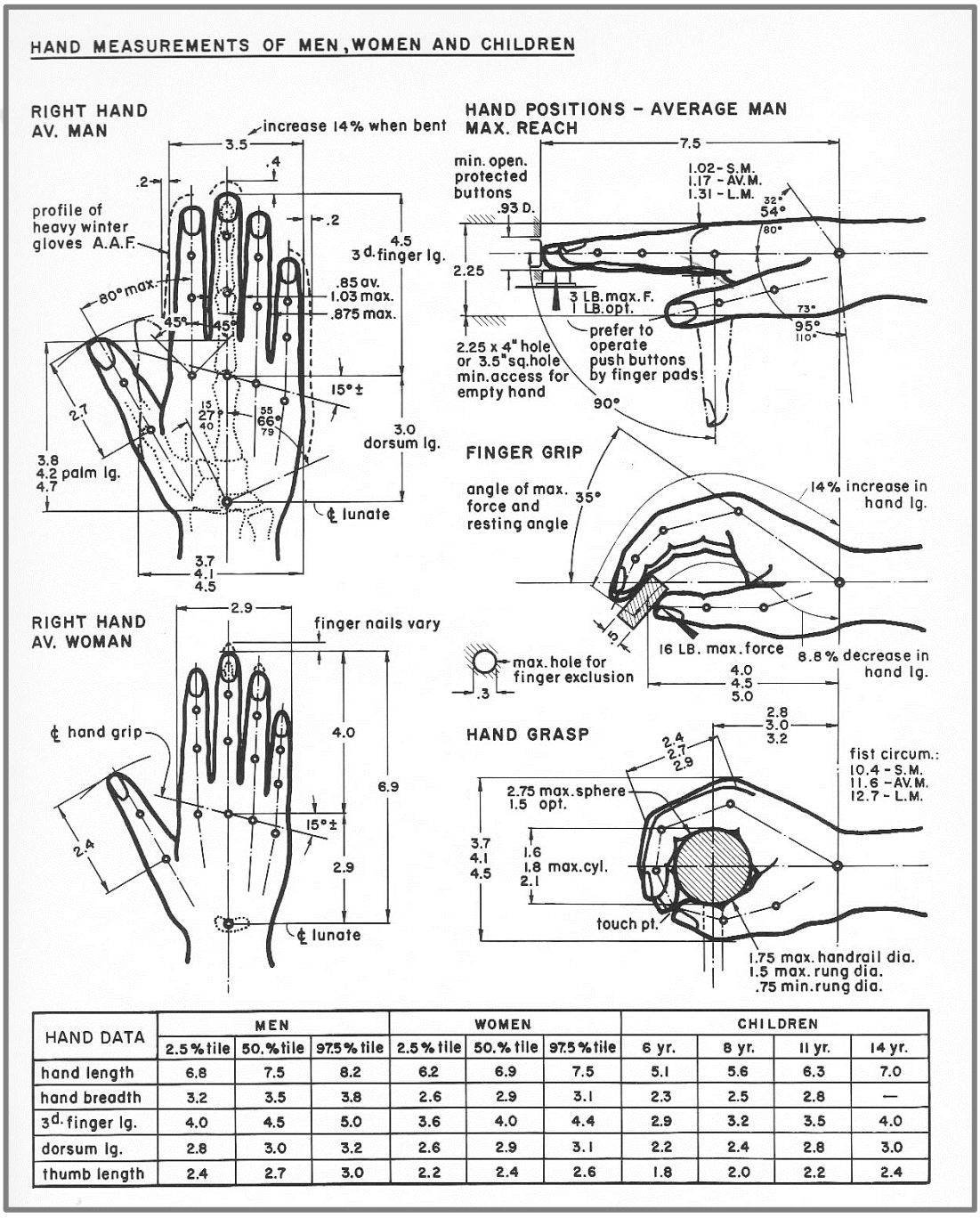
General Chemical Resistance

Ecological valuation

US Pharmacopeia Class VI Approved

APPENDIX-B

THE MEASURE OF MAN, HENRY DREYFUSS



HAND DATA	MEN			WOMEN			CHILDREN			
	2.5% tile	50.% tile	97.5% tile	2.5% tile	50.% tile	97.5% tile	6 yr.	8 yr.	11 yr.	14 yr.
hand length	6.8	7.5	8.2	6.2	6.9	7.5	5.1	5.6	6.3	7.0
hand breadth	3.2	3.5	3.8	2.6	2.9	3.1	2.3	2.5	2.8	—
3d. finger lg.	4.0	4.5	5.0	3.6	4.0	4.4	2.9	3.2	3.5	4.0
dorsum lg.	2.8	3.0	3.2	2.6	2.9	3.1	2.2	2.4	2.8	3.0
thumb length	2.4	2.7	3.0	2.2	2.4	2.6	1.8	2.0	2.2	2.4

APPENDIX-C

LINEAR MICRO ACTUATOR

100mm L16 – Actual Size

Miniature Linear Motion Series · L16

Actuonix Motion Devices' unique line of Miniature Linear Actuators enables a new generation of motion-enabled product designs, with capabilities that have never before been combined in a device of this size. These linear actuators are a superior alternative to designing your own push/pull mechanisms.

The L16 actuators are complete, self contained linear motion devices with position feedback for sophisticated position control capabilities, end of stroke limit switches for simple two position automation, or RC servo. Several gear ratios are available to give you varied speed/force configurations.

L16 Specifications

Gearing Option	35:1	63:1	150:1
Peak Power Point	50N @16mm/s	75N @10mm/s	175N @4mm/s
Peak Efficiency Point	24N @24mm/s	38N @15mm/s	75N @7mm/s
Max Speed (no load)	32mm/s	20mm/s	8mm/s
Max Force (lifted)	50N	100N	200N
Back Drive Force	31N	46N	102N
Stroke Option	50mm	100mm	140mm
Mass	56g	74g	84g
Repeatability (-P & LAC)	0.3mm	0.4mm	0.5mm
Max Side Load (extended)	40N	30N	20N
Closed Length (hole to hole)	118mm	168mm	208mm
Feedback Potentiometer	6kΩ±50%	11kΩ±50%	16kΩ±50%
Feedback Linearity	Less than 2.00%		
Input Voltage	"P", "S" Rated at 12VDC. "R" series rated at 6VDC		
Stall Current	650mA @ 12V		
Operating Temperature	-10°C to +50°C		
Audible Noise	60dB @ 45cm		
Ingress Protection	IP-54		
Mechanical Backlash	0.25mm		
Limit Switches	Max. Current Leakage: 8uA		
Maximum Static Force	250N		
Maximum Duty Cycle	20%		

Basis of Operation

The L16 is designed to push or pull a load along its full stroke length. The speed of travel is determined by the load applied. (See the Load Curves). Actuator speed can be reduced by lowering the drive voltage. When power is removed the actuator will hold its position, unless the applied load exceeds the back drive force. Repeated stalling or stalling for more than a few seconds will shorten the life of the actuator significantly. Stalling is when an actuator is pushing a load that it cannot move. Actuators should be tested in each specific application to determine their effective life under those loading conditions and environment.

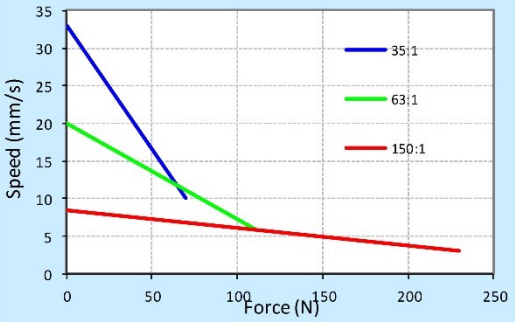
Applications

- → **Robotics**
- → **Consumer appliances**
- → **Toys**
- → **RC vehicles**
- → **Automotive**
- → **Industrial Automation**

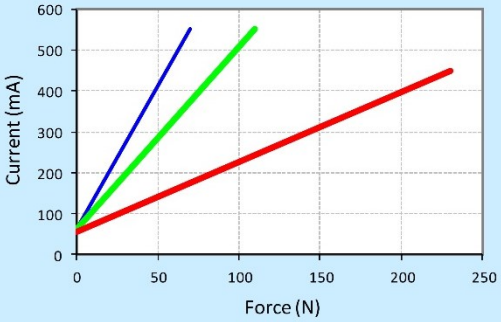
ACTUONIX MOTION DEVICES INC
Actuonix Motion Devices Inc
580 Starling Lane
Victoria, BC, V9E 2A9
Canada
1 (206) 347-9684 *phone*
1 (888) 225-9198 *toll-free*
1 (206) 347-9684 *fax*
sales@actuonix.com
www.actuonix.com

L16 Specifications

Load Curves



Current Curves



Model Selection

The L16 has 3 configuration choices: Stroke, Gear Ratio and Controller. L16 options are identified according to the following model numbering scheme:

L16-SS-GG-VV-C

Feature	*Options*
SS: Stroke	**50, 100, 140 (mm)**
GG: Gear reduction ratio (refer to load curves above)	**35, 63, 150** (lower ratios are faster but push less force, and vice versa)
VV: Voltage	**12** vdc or **6** vdc (-R only)
C: Controller	**P** Potentiometer Feedback **S** Limit Switches **R** RC Linear Servo

L16 Controller Options

Option S – End of Stroke Limit Switches

WIRING: (see last page for pin numbering)
1 - Red – Motor V+
2 – Black – Motor V- (Ground)

-S actuators are ideal for manually controlled applications and simple two position automated mechanisms. The –S actuators have limit switches that will turn off power to the motor when the actuator reaches within 0.5mm of the end of stroke. Internal diodes allow the actuator to reverse away from the limit switch. The limit switches cannot be moved once the actuator is manufactured. While voltage is applied to the motor power pins, (1 & 2) the actuator extends. Reverse the polarity and the actuator retracts. This can be accomplished manually with a DPDT switch or relay, or using an H-Bridge circuit. The –S model cannot be used with the LAC control board.

Option P – Potentiometer Position Feedback

WIRING: (see last page for pin numbering)
1 - Orange – Feedback Potentiometer negative reference rail
2 - Purple – Feedback Potentiometer wiper
3 - Red – Motor V+ (6V or 12V)
4 - Black – Motor V- (Ground)
5 - Yellow – Feedback Potentiometer positive reference rail

-P actuators are suited to automatically controlled positioning systems, but they can also be driven manually. The –P actuators have no built in controller, but do provide an analog position feedback signal that can be input to an external controller. While voltage is applied to the motor power pins, (3 & 4) the actuator extends. Reverse the polarity and the actuator retracts. This can be accomplished manually with a DPDT switch or relay, or using an H-Bridge circuit. Position of the actuator stroke can be monitored via the internal linear potentiometer. Provide any stable low and high reference voltage on pins 1 & 5, then read the position signal on pin 2. The voltage on pin 2 will vary linearly between the two reference voltages in proportion to the position of the actuator stroke.

The L16 –P actuator can be used as a linear servo by connecting the actuator to an external controller such as the LAC board offered by Actuonix. This control board reads the position signal from the L16, compares it with your input control signal then commands the actuator to move via an on-board H-bridge circuit. The LAC allows any one of the following control inputs: Analog 0-3.3V or 4-20mA, or Digital 0-5V PWM, 1-2ms Standard RC, or USB. The RC input effectively transforms your L16 into a linear servo, which is a direct replacement for any common hobby servo used in RC toys and robotics. Refer to the LAC datasheet for more details.

ACTUONIX MOTION DEVICES INC.
Actuonix Motion Devices Inc
580 Starling Lane
Victoria, BC, V9E 2A9
Canada
1 (206) 347-9684 *phone*
1 (888) 225-9198 *toll-free*
1 (206) 347-9684 *fax*
sales@actuonix.com
www.actuonix.com

Option R – RC Linear Servo

WIRING: (see last page for pin numbering)
1 - White – **RC input signal** (RC-servo compatible)
2 - Red – **Power** (+6 VDC)
3 - Black – **Ground**

Note: Reversing the polarity of pins 2 and 3 may permanently damage the actuator

-R actuators are ideally suited to use in robotics and radio control models. The –R actuators or 'linear servos' are a direct replacement for regular radio controlled hobby servos. The desired actuator position is input to the actuator on lead 1 as a positive 5 Volt pulse width signal. A 1.0 ms pulse commands the controller to fully retract the actuator, and a 2.0 ms pulse signals it to fully extend. If the motion of the actuator, or of other servos in your system, seems erratic, place a 1–4Ω resistor in series with the actuator's red V+ lead wire.

L16 –R Linear Servos are the only 6 volt models in the L16 range because they are designed to work with typical RC receivers and battery packs. Consequently they also are compatible with Arduino control boards, VEX Microcontrollers and many other similar boards designed for robotics.

Ordering

Small quantity orders can be placed directly online at www.actuonix.com. Purchase orders, volume quotes, and custom requests can be sent to sales@actuonix.com .

Each actuator ships with two mounting brackets, #8-32 mounting hardware and male connector pins. The cable length is approximately 300mm and connector is a 0.1" pitch female socket connector.

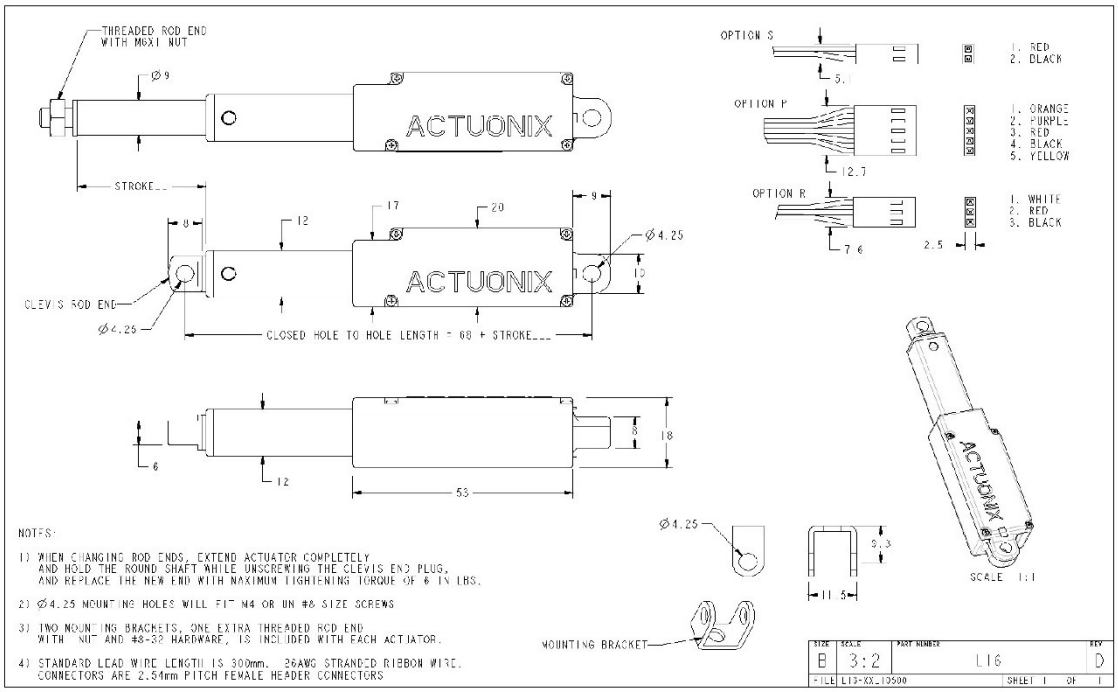
Custom Options

Contact Actuonix for quotes and lead-times on any non-standard customizations. Typically we can modify the stroke, limit switch positions, cable length and connector on a minimum order quantity of 500 pieces.

Any modifications requiring mold changes will involve higher MOQ and/or NRE fees.

Accessories

Visit www.actuonix.com for compatible accessories including cable extensions, DPDT switches and relays, position indicators and power supplies.



ACTUONIX MOTION DEVICES INC
Actuonix Motion Devices Inc
580 Starling Lane
Victoria, BC, V9E 2A9
Canada
1 (206) 347-9684 *phone*
1 (888) 225-9198 *toll-free*
1 (206) 347-9684 *fax*
sales@actuonix.com
www.actuonix.com

REV D September 2016

APPENDIX- D

RESISTANCE FORCE SENSOR

INTERLINK ELECTRONICS

FSR® 400 Series Data Sheet

Force Sensing Resistors®

Features and Benefits

- Actuation force as low as 0.2N and sensitivity range to 20N
- Cost effective
- Ultra thin
- Robust; up to 10M actuations
- Simple and easy to integrate

Description

Interlink Electronics FSR® 400 Series is part of the single zone Force Sensing Resistor® family. Force Sensing Resistors, or FSR's, are robust polymer thick film (PTF) devices that exhibit a decrease in resistance with increase in force applied to the surface of the sensor. This force sensitivity is optimized for use in human machine interface devices including automotive electronics, medical systems, industrial controls and robotics.

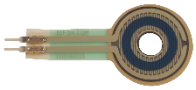
The FSR 400 Series sensors come in seven different models with four different connecting options. A battery operated demo is available. Call us for more information at +1 805-484-8855.

FSR® 400 Short
5mm Circle x 20mm

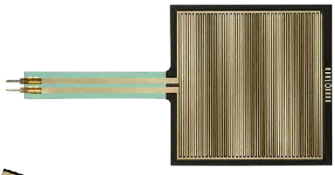
FSR® 400
5mm Circle x 38mm

FSR® 402 Short
13mm Circle x 25mm

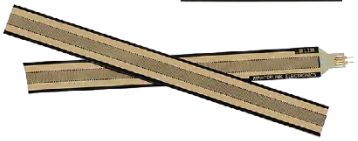
FSR® 402
13mm Circle x 56mm



FSR® 404
20mm Donut with 5.5mm hold



FSR® 406
38mm Square x 83mm



FSR® 408-xxx
10mm Wide x xxx mm strip
xxx = 50, 100, 200, 300, 400, 500mm

P/N: PDS-10004-C

Human-Machine Interface Solutions for a Connected World

INTERLINK ELECTRONICS

FSR® 400 Series Data Sheet

Force Sensing Resistors®

Device Characteristics

Actuation Force*	~0.2N min
Force Sensitivity Range*	~0.2N – 20N
Force Resolution	Continuous (analog)
Force Repeatability Single Part	+/- 2%
Force Repeatability Part to Part	+/- 6% (Single Batch)
Non-Actuated Resistance	>10 Mohms
Hysteresis	+10% Average (RF+ - RF-)/RF+
Device Rise Time	< 3 Microseconds
Long Term Drift	
1kg load, 35 days	< 5% log10(time)
Operating Temperature Performance	
Cold: -40ºC after 1 hour	-5% average resistance change
Hot: +85ºC after 1 hour	-15% average resistance change
Hot Humid: +85ºC 95RH after 1 hour	+10% average resistance change
Storage Temperature Performance	
Cold: -25ºC after 120 hours	-10% average resistance change
Hot: +85ºC after 120 hours	-%5 average resistance change
Hot Humid: +85ºC 95RH after 240 hours	+30% average resistance change
Tap Durability	
Tested to 10 Million actuations, 1kg, 4Hz	-10% average resistance change
Standing Load Durability	
2.5kg for 24 hours	-5% average resistance change
EMI	Generates no EMI
ESD	Not ESD Sensitive
UL	All materials UL grade 94 V-1 or better
RoHS	Compliant

Specifications are derived from measurements taken at 1000 grams, and are given as (one standard deviation/mean), unless otherwise noted.
*Typical value. Force dependent on actuation interface, mechanics, and measurement electronics.

FSR® 400 Series Data Sheet

Force Sensing Resistor®

Connector Information

Bare Tail

Female Tin Contacts
PN: TE 2-487406-4

Female Tin Contacts with 2 Pin Housing
PN: TE 2-487406-4
PN: TE 487378-1

Solder Tabs
PN: TE 1-88997-2

Other Available Part Numbers:
Hardware Development Kit, PN 54-76247

Application Information

For specific application needs please contact Interlink Electronics support team. An Integration Guide and Hardware Development Kit (HDK) are also available. FSR's are two-wire devices with a resistance that depends on applied force. Below is a force vs. resistance graph that illustrates a typical FSR® response characteristic. Please note that the graph values are reference only and actual values are dependent upon actuation system mechanics and sensor geometry.

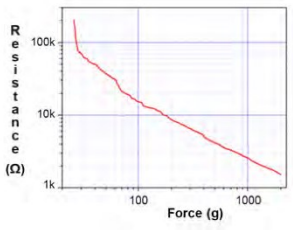
For simple force-to-voltage conversion, the FSR device is tied to a measuring resistor in a voltage divider (see figure below) and the output is described by the following equation.

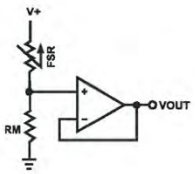
$$V_{OUT} = \frac{R_M V +}{(R_M + R_{FSR})}$$

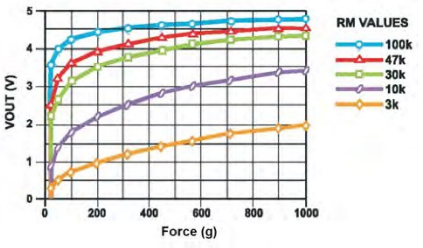
In the configuration shown, the output voltage increases with increasing force. If R_{FSR} and R_M are swapped, the output swing will decrease with increasing force. The measuring resistor, R_M, is chosen to maximize the desired force sensitivity range and to limit current. Depending on the impedance requirements of the measuring circuit, the voltage divider could be followed by an op-amp.

A family of force vs. V_{OUT} curves is shown on the graph below for a standard FSR in a voltage divider configuration with various R_M resistors. A V+ of 5V was used for these examples. Please note that the graph values are for reference only and will vary between different sensors and applications.

Refer to the FSR Integration Guide for more integration methods and techniques.







INTERLINK ELECTRONICS

FSR® Model 400 Short Tail

Force Sensing Resistor®

Model 400 Short Tail:

Active Area: Ø5.62mm
Nominal Thickness: 0.30mm
Switch Travel: 0.05mm

Available Part Numbers:

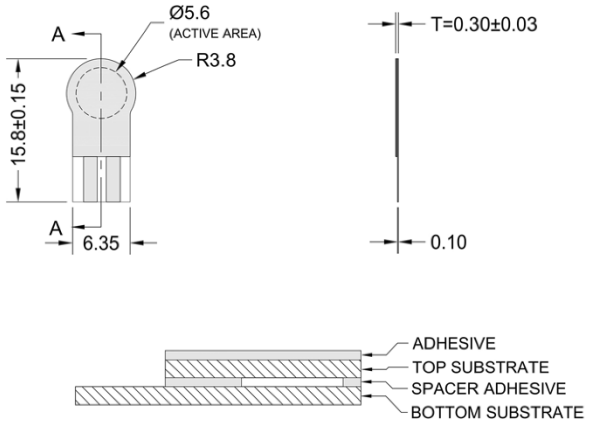
PN: 34-47021 Model 400 Short Tail
- No contacts or solder tabs

PN: 34-00005 Model 400 Short Tail
- with female contacts

PN: 34-00006 Model 400 Short Tail
- with female contacts and housing

PN: 34-00004 Model 400 Short Tail
- with solder tabs

Sensor Mechanical Data



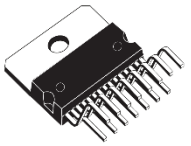
SECTION A-A
LAYER STACK-UP

Exploded View

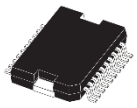
L298N MOTOR DRIVER

DUAL FULL-BRIDGE DRIVER

- OPERATING SUPPLY VOLTAGE UP TO 46 V
- TOTAL DC CURRENT UP TO 4 A
- LOW SATURATION VOLTAGE
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)



Multiwatt15



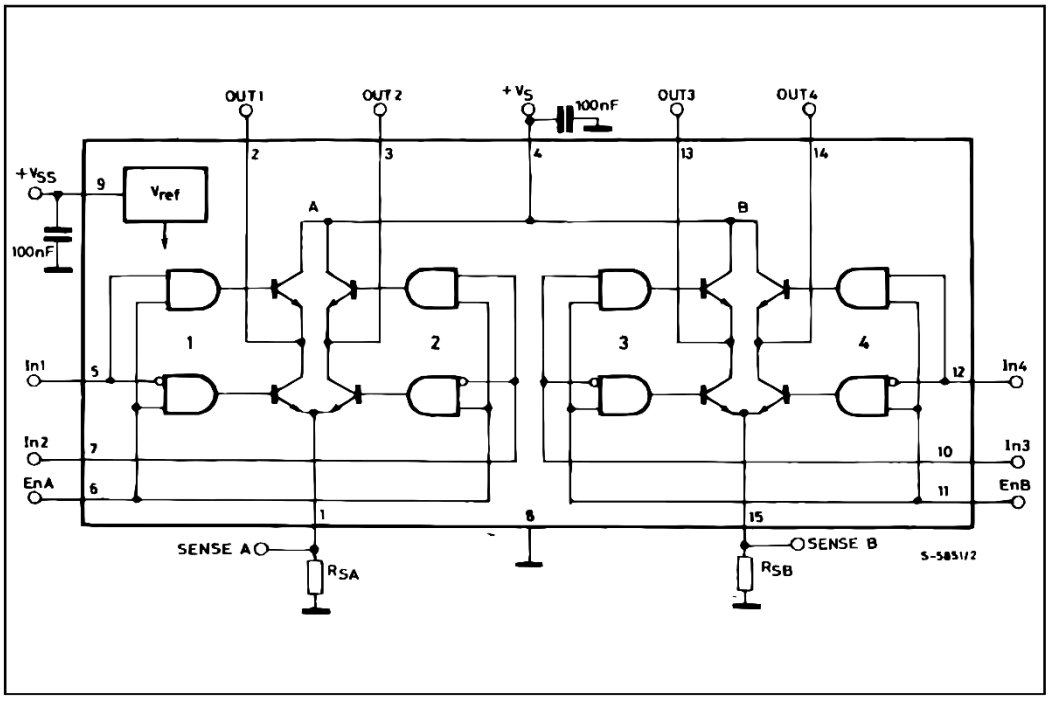
PowerSO20

ORDERING NUMBERS : L298N (Multiwatt Vert.)
L298HN (Multiwatt Horiz.)
L298P (PowerSO20)

DESCRIPTION

The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

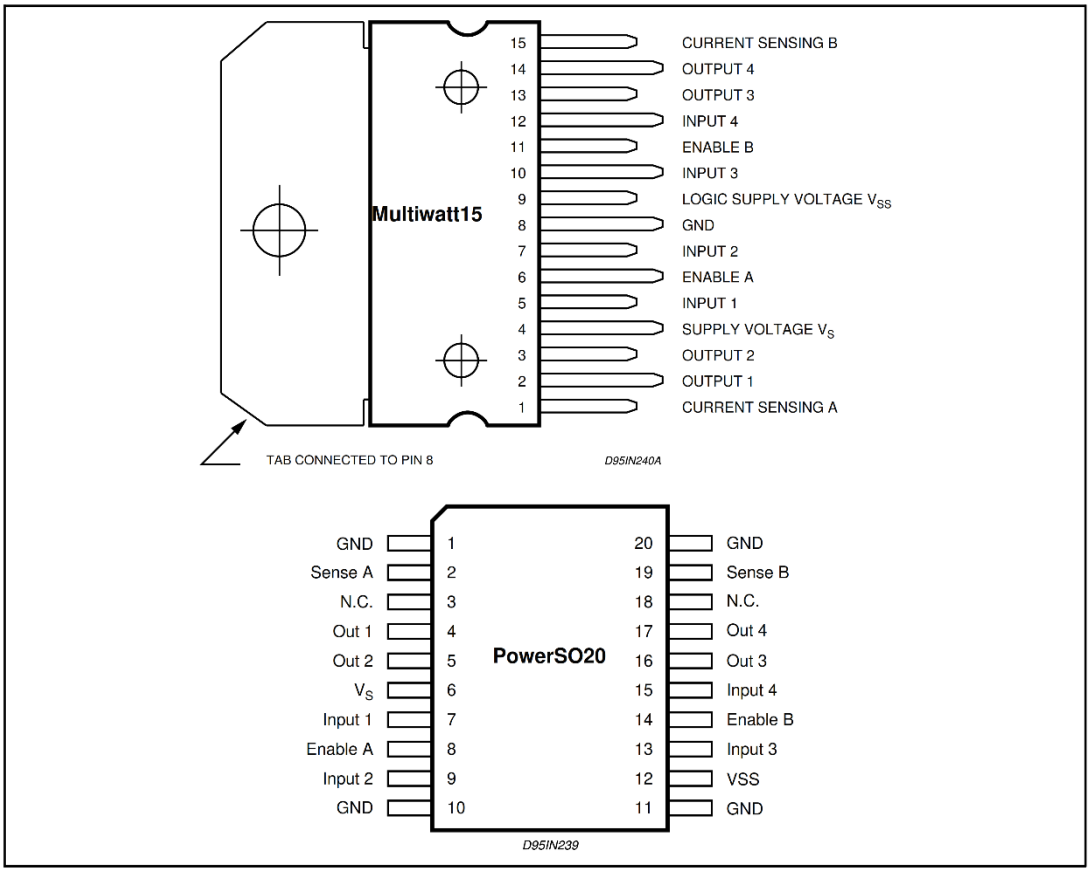
BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Power Supply	50	V
V_{SS}	Logic Supply Voltage	7	V
V_I,V_{en}	Input and Enable Voltage	–0.3 to 7	V
I_O	Peak Output Current (each Channel) – Non Repetitive (t = 100µs) –Repetitive (80% on –20% off; t_{on} = 10ms) –DC Operation	 3 2.5 2	 A A A
V_{sens}	Sensing Voltage	–1 to 2.3	V
P_{tot}	Total Power Dissipation (T_{case} = 75°C)	25	W
T_{op}	Junction Operating Temperature	–25 to 130	°C
T_{stg}, T_j	Storage and Junction Temperature	–40 to 150	°C

PIN CONNECTIONS (top view)



THERMAL DATA

Symbol	Parameter		PowerSO20	Multiwatt15	Unit
$R_{th\ j\text{-}case}$	Thermal Resistance Junction-case	Max.	–	3	°C/W
$R_{th\ j\text{-}amb}$	Thermal Resistance Junction-ambient	Max.	13 (*)	35	°C/W

(*) Mounted on aluminum substrate

PIN FUNCTIONS (refer to the block diagram)

MW.15	PowerSO	Name	Function
1;15	2;19	Sense A; Sense B	Between this pin and ground is connected the sense resistor to control the current of the load.
2;3	4;5	Out 1; Out 2	Outputs of the Bridge A; the current that flows through the load connected between these two pins is monitored at pin 1.
4	6	V_S	Supply Voltage for the Power Output Stages. A non-inductive 100nF capacitor must be connected between this pin and ground.
5;7	7;9	Input 1; Input 2	TTL Compatible Inputs of the Bridge A.
6;11	8;14	Enable A; Enable B	TTL Compatible Enable Input: the L state disables the bridge A (enable A) and/or the bridge B (enable B).
8	1,10,11,20	GND	Ground.
9	12	VSS	Supply Voltage for the Logic Blocks. A100nF capacitor must be connected between this pin and ground.
10; 12	13;15	Input 3; Input 4	TTL Compatible Inputs of the Bridge B.
13; 14	16;17	Out 3; Out 4	Outputs of the Bridge B. The current that flows through the load connected between these two pins is monitored at pin 15.
–	3;18	N.C.	Not Connected

ELECTRICAL CHARACTERISTICS (V_S = 42V; V_{SS} = 5V, T_j = 25°C; unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_S	Supply Voltage (pin 4)	Operative Condition	V_{IH} +2.5		46	V
V_{SS}	Logic Supply Voltage (pin 9)		4.5	5	7	V
I_S	Quiescent Supply Current (pin 4)	V_{en} = H; I_L = 0 V_i = L V_i = H		13 50	22 70	mA mA
		V_{en} = L V_i = X			4	mA
I_{SS}	Quiescent Current from V_{SS} (pin 9)	V_{en} = H; I_L = 0 V_i = L V_i = H		24 7	36 12	mA mA
		V_{en} = L V_i = X			6	mA
V_{iL}	Input Low Voltage (pins 5, 7, 10, 12)		–0.3		1.5	V
V_{iH}	Input High Voltage (pins 5, 7, 10, 12)		2.3		VSS	V
I_{iL}	Low Voltage Input Current (pins 5, 7, 10, 12)	V_i = L			–10	µA
I_{iH}	High Voltage Input Current (pins 5, 7, 10, 12)	Vi = H ≤ V_{SS} –0.6V		30	100	µA
V_{en} = L	Enable Low Voltage (pins 6, 11)		–0.3		1.5	V
V_{en} = H	Enable High Voltage (pins 6, 11)		2.3		V_{SS}	V
I_{en} = L	Low Voltage Enable Current (pins 6, 11)	V_{en} = L			–10	µA
I_{en} = H	High Voltage Enable Current (pins 6, 11)	V_{en} = H ≤ V_{SS} –0.6V		30	100	µA
$V_{CEsat\ (H)}$	Source Saturation Voltage	I_L = 1A I_L = 2A	0.95	1.35 2	1.7 2.7	V V
$V_{CEsat\ (L)}$	Sink Saturation Voltage	I_L = 1A (5) I_L = 2A (5)	0.85	1.2 1.7	1.6 2.3	V V
V_{CEsat}	Total Drop	I_L = 1A (5) I_L = 2A (5)	1.80		3.2 4.9	V V
V_{sens}	Sensing Voltage (pins 1, 15)		–1 (1)		2	V

ELECTRICAL CHARACTERISTICS (continued)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
T_1 (V_i)	Source Current Turn-off Delay	0.5 V_i to 0.9 I_L (2); (4)		1.5		µs
T_2 (V_i)	Source Current Fall Time	0.9 I_L to 0.1 I_L (2); (4)		0.2		µs
T_3 (V_i)	Source Current Turn-on Delay	0.5 V_i to 0.1 I_L (2); (4)		2		µs
T_4 (V_i)	Source Current Rise Time	0.1 I_L to 0.9 I_L (2); (4)		0.7		µs
T_5 (V_i)	Sink Current Turn-off Delay	0.5 V_i to 0.9 I_L (3); (4)		0.7		µs
T_6 (V_i)	Sink Current Fall Time	0.9 I_L to 0.1 I_L (3); (4)		0.25		µs
T_7 (V_i)	Sink Current Turn-on Delay	0.5 V_i to 0.9 I_L (3); (4)		1.6		µs
T_8 (V_i)	Sink Current Rise Time	0.1 I_L to 0.9 I_L (3); (4)		0.2		µs
fc (V_i)	Commutation Frequency	I_L = 2A		25	40	KHz
T_1 (V_{en})	Source Current Turn-off Delay	0.5 V_{en} to 0.9 I_L (2); (4)		3		µs
T_2 (V_{en})	Source Current Fall Time	0.9 I_L to 0.1 I_L (2); (4)		1		µs
T_3 (V_{en})	Source Current Turn-on Delay	0.5 V_{en} to 0.1 I_L (2); (4)		0.3		µs
T_4 (V_{en})	Source Current Rise Time	0.1 I_L to 0.9 I_L (2); (4)		0.4		µs
T_5 (V_{en})	Sink Current Turn-off Delay	0.5 V_{en} to 0.9 I_L (3); (4)		2.2		µs
T_6 (V_{en})	Sink Current Fall Time	0.9 I_L to 0.1 I_L (3); (4)		0.35		µs
T_7 (V_{en})	Sink Current Turn-on Delay	0.5 V_{en} to 0.9 I_L (3); (4)		0.25		µs
T_8 (V_{en})	Sink Current Rise Time	0.1 I_L to 0.9 I_L (3); (4)		0.1		µs

1) 1)Sensing voltage can be –1 V for t ≤ 50 µsec; in steady state V_{sens} min ≥ – 0.5 V.
2) See fig. 2.
3) See fig. 4.
4) The load must be a pure resistor.

Figure 1 : Typical Saturation Voltage vs. Output Current.

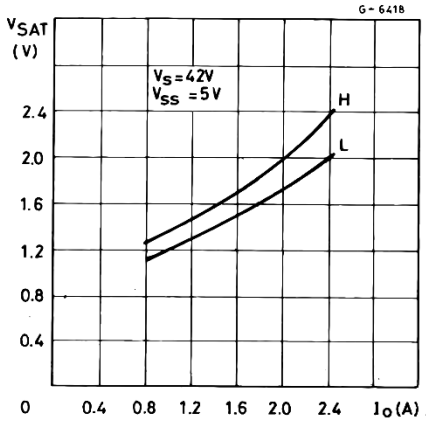
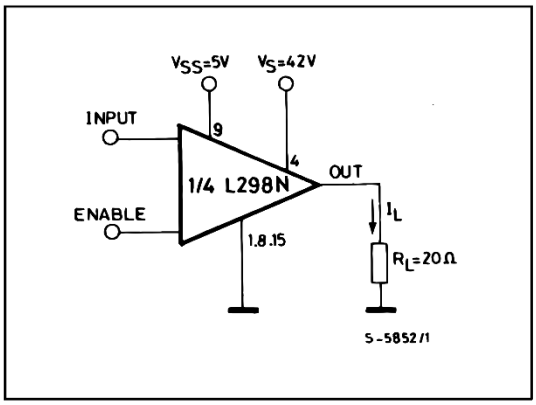


Figure 2 : Switching Times Test Circuits.



Note : For INPUT Switching, set EN = H
For ENABLE Switching, set IN = H

Figure 3 : Source Current Delay Times vs. Input or Enable Switching.

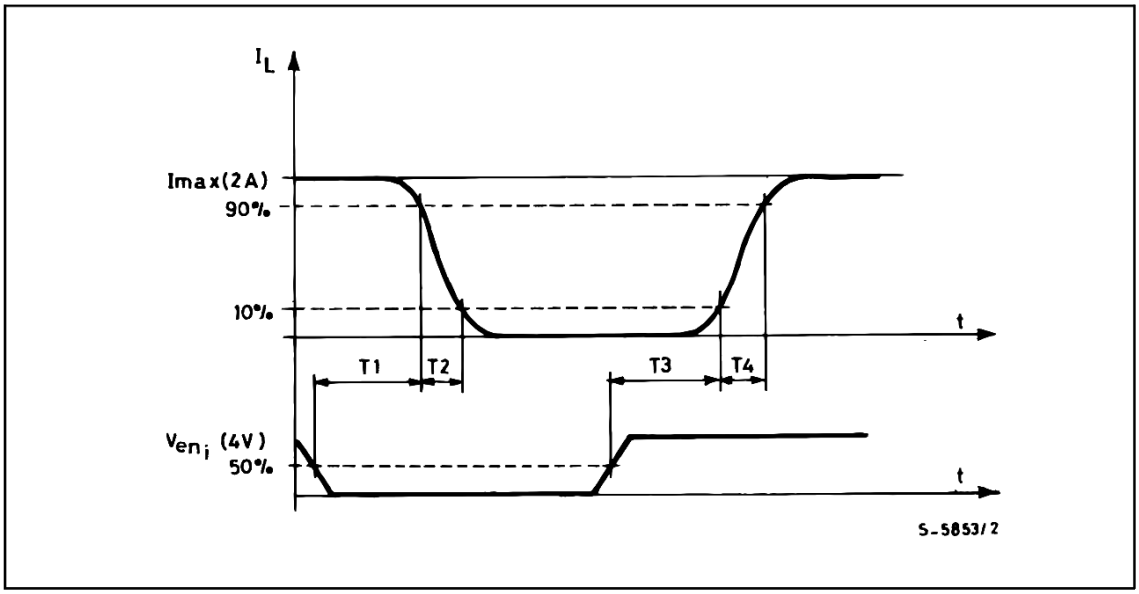
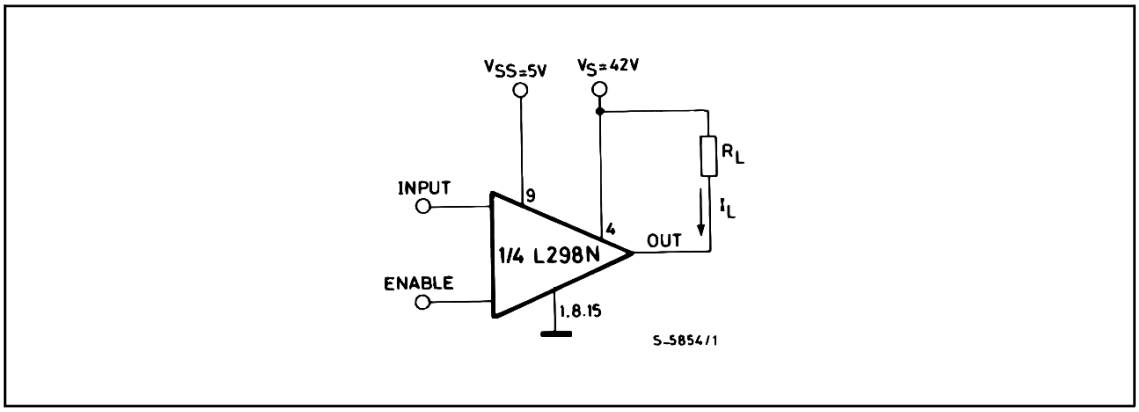


Figure 4 : Switching Times Test Circuits.



Note : For INPUT Switching, set EN = H
For ENABLE Switching, set IN = L

Figure 5 : Sink Current Delay Times vs. Input 0 V Enable Switching.

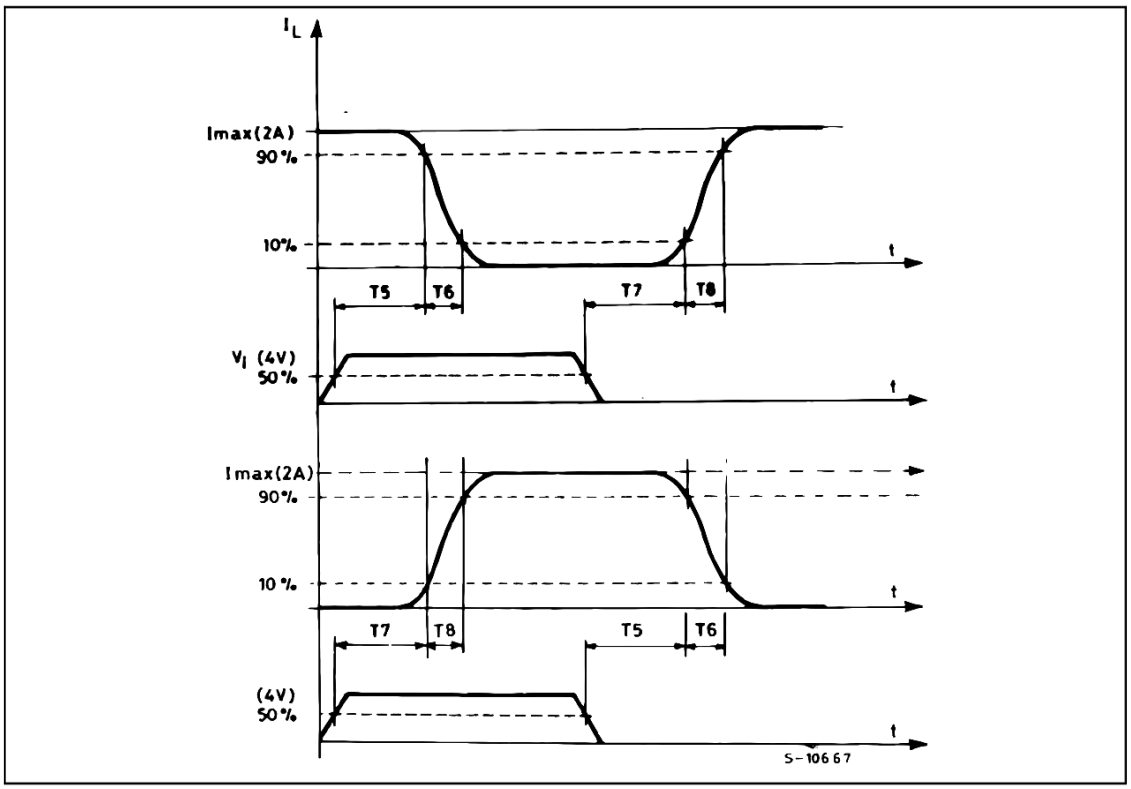
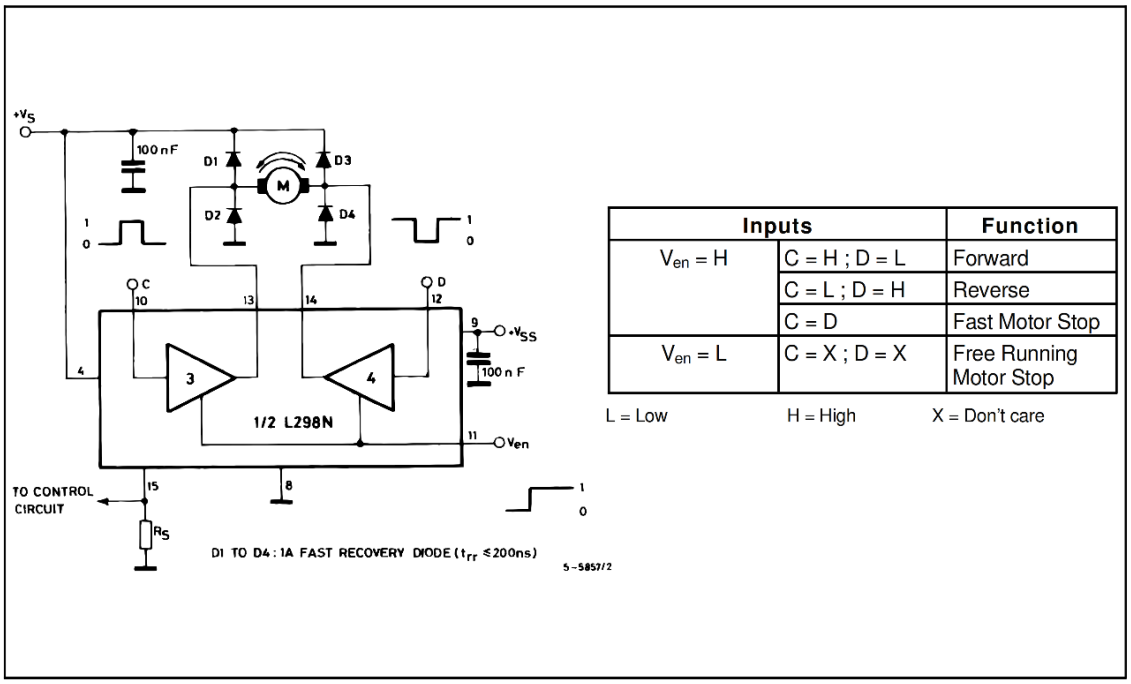


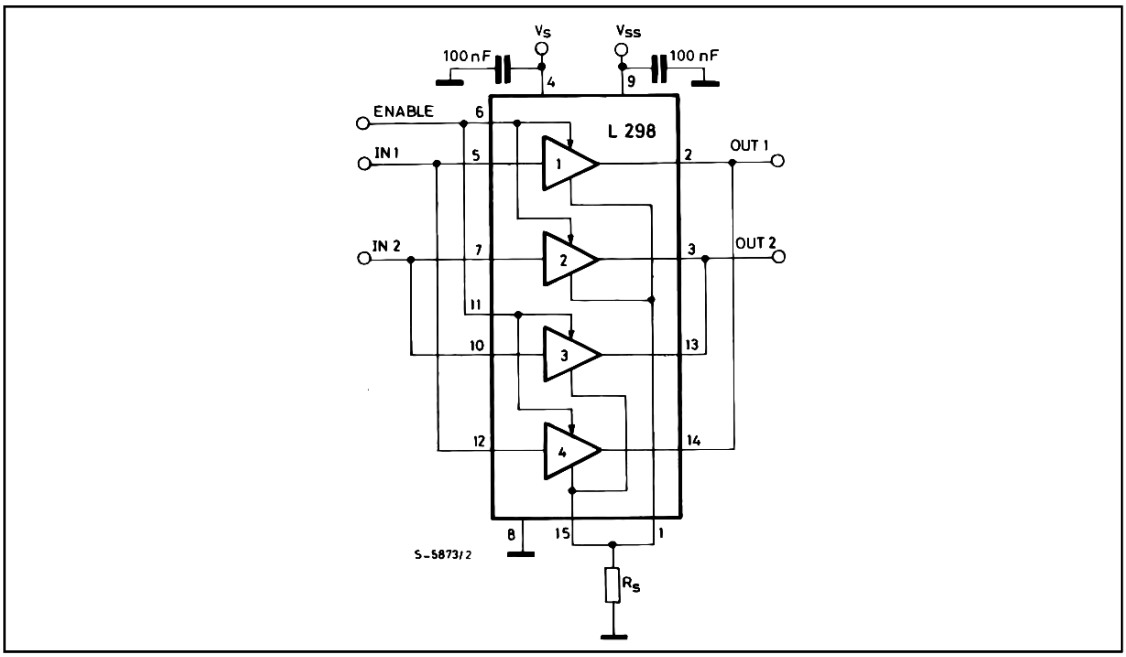
Figure 6 : Bidirectional DC Motor Control.



Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

Figure 7 : For higher currents, outputs can be paralleled. Take care to parallel channel 1 with channel 4 and channel 2 with channel 3.



APPLICATION INFORMATION (Refer to the block diagram)

1.1. POWER OUTPUT STAGE

The L298 integrates two power output stages (A ; B). The power output stage is a bridge configuration and its outputs can drive an inductive load in common or differenzial mode, depending on the state of the inputs. The current that flows through the load comes out from the bridge at the sense output : an external resistor (R_{SA} ; R_{SB}.) allows to detect the intensity of this current.

1.2. INPUT STAGE

Each bridge is driven by means of four gates the input of which are In1 ; In2 ; EnA and In3 ; In4 ; EnB. The In inputs set the bridge state when The En input is high ; a low state of the En input inhibits the bridge. All the inputs are TTL compatible.

2. SUGGESTIONS

A non inductive capacitor, usually of 100 nF, must be foreseen between both Vs and Vss, to ground, as near as possible to GND pin. When the large capacitor of the power supply is too far from the IC, a second smaller one must be foreseen near the L298.

The sense resistor, not of a wire wound type, must be grounded near the negative pole of Vs that must be near the GND pin of the I.C.

Each input must be connected to the source of the driving signals by means of a very short path.

Turn-On and Turn-Off : Before to Turn-ON the Supply Voltage and before to Turn it OFF, the Enable input must be driven to the Low state.

3. APPLICATIONS

Fig 6 shows a bidirectional DC motor control Schematic Diagram for which only one bridge is needed. The external bridge of diodes D1 to D4 is made by four fast recovery elements (trr $\leq$ 200 nsec) that must be chosen of a VF as low as possible at the worst case of the load current.

The sense output voltage can be used to control the current amplitude by chopping the inputs, or to provide overcurrent protection by switching low the enable input.

The brake function (Fast motor stop) requires that the Absolute Maximum Rating of 2 Amps must never be overcome.

When the repetitive peak current needed from the load is higher than 2 Amps, a paralleled configuration can be chosen (See Fig.7).

An external bridge of diodes are required when inductive loads are driven and when the inputs of the IC are chopped ; Shottky diodes would be preferred.

This solution can drive until 3 Amps In DC operation and until 3.5 Amps of a repetitive peak current.

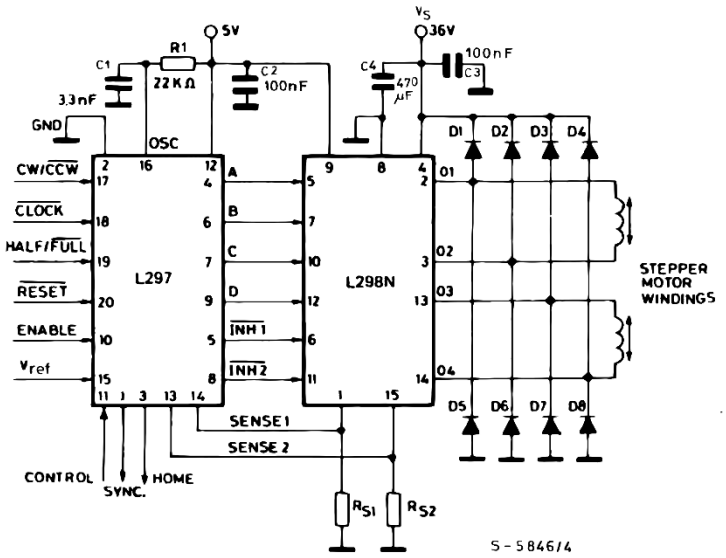
On Fig 8 it is shown the driving of a two phase bipolar stepper motor ; the needed signals to drive the inputs of the L298 are generated, in this example, from the IC L297.

Fig 9 shows an example of P.C.B. designed for the application of Fig 8.

Fig 10 shows a second two phase bipolar stepper motor control circuit where the current is controlled by the I.C. L6506.

Figure 8 : Two Phase Bipolar Stepper Motor Circuit.

This circuit drives bipolar stepper motors with winding currents up to 2 A. The diodes are fast 2 A types.



$R_{S1} = R_{S2} = 0.5\ \Omega$

D1 to D8 = 2 A Fast diodes { $V_F \leq 1.2$ V @ I = 2 A
trr ≤ 200 ns

Figure 9 : Suggested Printed Circuit Board Layout for the Circuit of fig. 8 (1:1 scale).

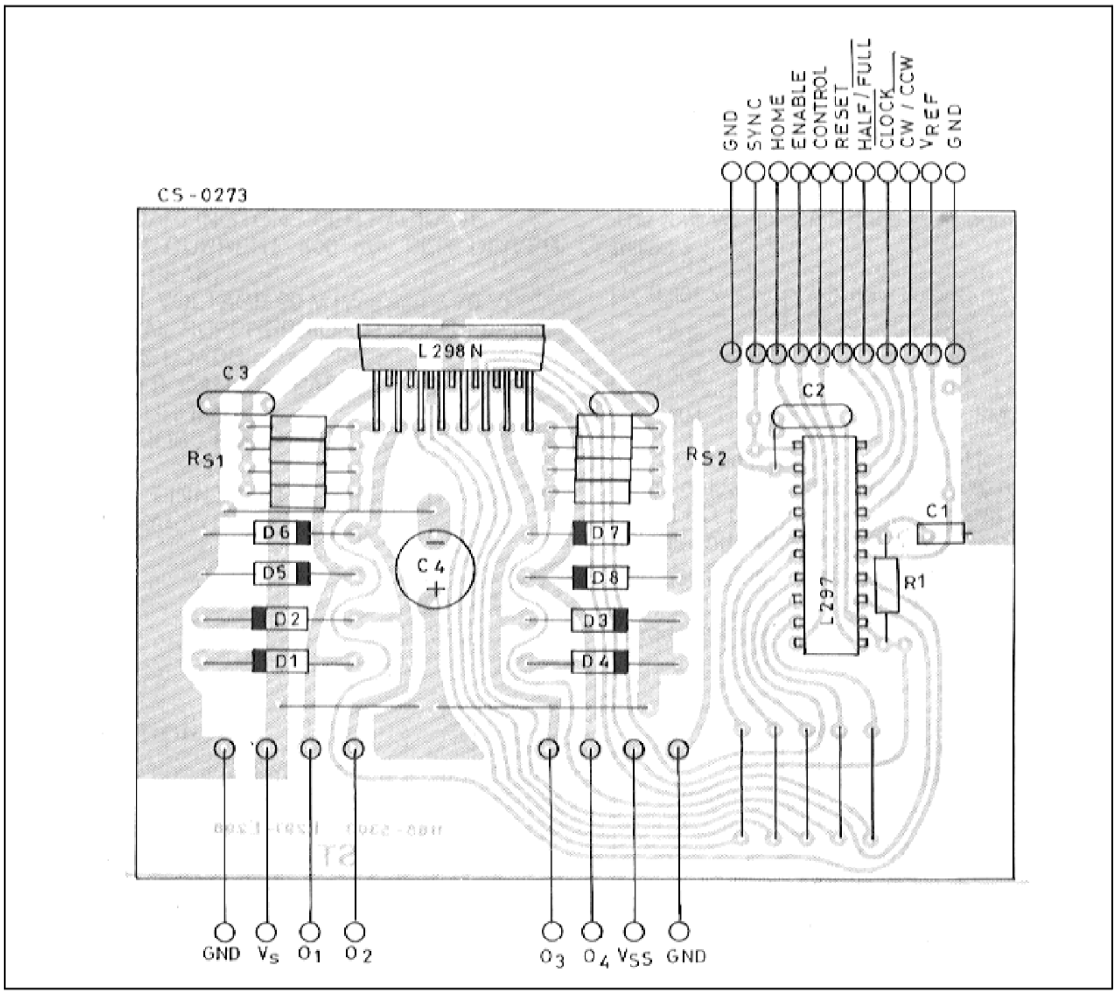
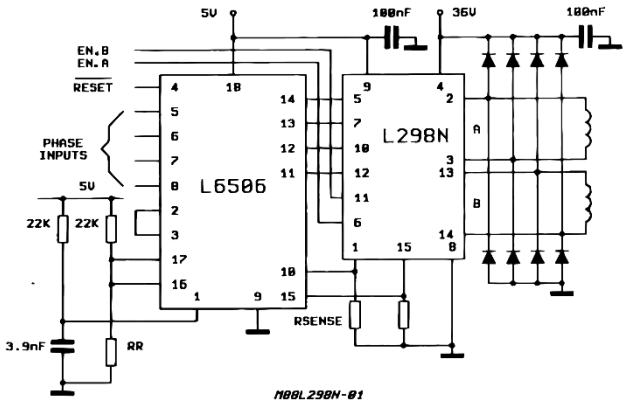


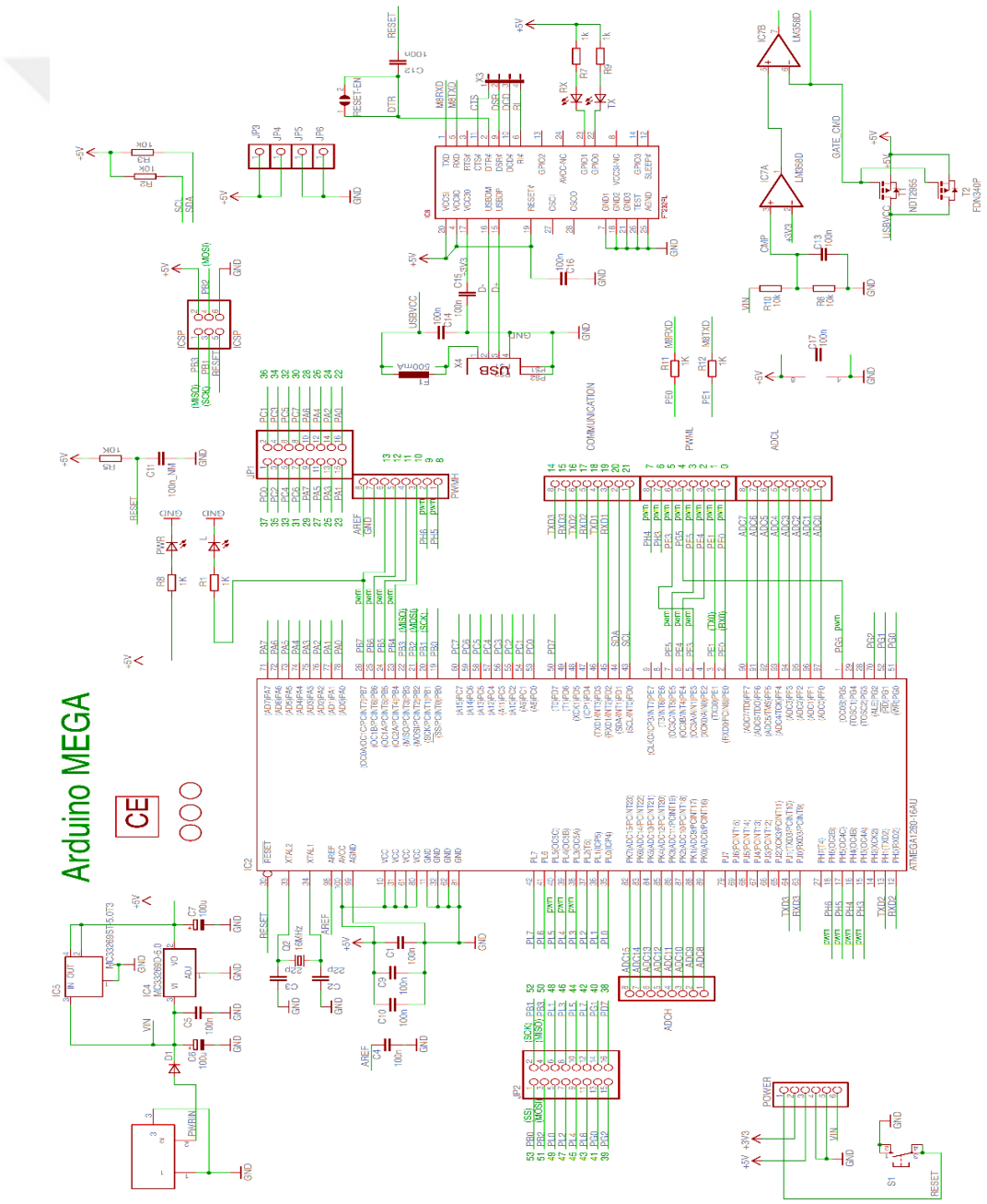
Figure 10 : Two Phase Bipolar Stepper Motor Control Circuit by Using the Current Controller L6506.



R_R and R_{sense} depend from the load current

APPENDIX-F

ARDUINO MEGA MICROCONTROLLERSCHEMATIC



APPENDIX-G

TECHNICAL DRAWINGS OF THE MECHANICAL PARTS

APPENDIX-H

C# SCRIPTS FOR COMUNICATION WITH MYO ARMBAND

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;

namespace Thalmic.Myo
{
   public class Myo //: MonoBehaviour
   {
      #region MYOConnect
      private readonly Hub _hub;
      private IntPtr _handle;
      private bool streamEmg;
      public static Myo Instance;

      public static calibratorClass calibClass;
      public static calibratorClass fist_calibration_data;
      public static calibratorClass rest_calibration_data;

      internal Myo(Hub hub, IntPtr handle)
      {
         System.Diagnostics.Debug.Assert(handle != IntPtr.Zero, "Cannot construct Myo instance with
null pointer.");
         _hub = hub;
         _handle = handle;
         // Thread threadd = new Thread(new ThreadStart(emglistener));
         // threadd.Start();
         calibClass = new calibratorClass();
         fist_calibration_data = new calibratorClass();
         rest_calibration_data = new calibratorClass();
      }
      public event EventHandler<MyoEventArgs> Connected;
      public event EventHandler<MyoEventArgs> Disconnected;
      public event EventHandler<ArmSyncedEventArgs> ArmSynced;
      public event EventHandler<MyoEventArgs> ArmUnsynced;
      public event EventHandler<PoseEventArgs> PoseChange;
```

```
        public event EventHandler<OrientationDataEventArgs> OrientationData; //Orientation is the
direction the arm is pointing in space
        public event EventHandler<AccelerometerDataEventArgs> AccelerometerData; //the accelerometer
measures the force of acceleration applied to the armband
        public event EventHandler<GyroscopeDataEventArgs> GyroscopeData; //The gyroscope measures
angular velocity (ie, how fast is your arm is rotating, in radians per second)
        public event EventHandler<RssiEventArgs> Rssi;
        public event EventHandler<MyoEventArgs> Unlocked;
        public event EventHandler<MyoEventArgs> Locked;
        public event EventHandler<EmgDataEventArgs> EmgData;
        internal Hub Hub { get { return _hub; } }
        internal IntPtr Handle { get { return _handle; } }

        public object EmgTxt { get; private set; }

        public void Vibrate(VibrationType type) { libmyo.vibrate(_handle, (libmyo.VibrationType)type,
IntPtr.Zero); }
        public void RequestRssi() { libmyo.request_rssi(_handle, IntPtr.Zero); }
        public Result SetStreamEmg(StreamEmg type) { streamEmg = true; return
(Result)libmyo.set_stream_emg(_handle, (libmyo.StreamEmg)type, IntPtr.Zero); }
        public void Unlock(UnlockType type) { libmyo.myo_unlock(_handle, (libmyo.UnlockType)type,
IntPtr.Zero); }
        public void Lock() { libmyo.myo_lock(_handle, IntPtr.Zero); }
        public void NotifyUserAction() { libmyo.myo_notify_user_action(_handle,
libmyo.UserActionType.Single, IntPtr.Zero); }
        #endregion MYOConnect

        #region MYOsensors
        public int[] emgData = new int[7];

        public static Boolean isPressed = false;
        public static Boolean isMax = false;
        public static Boolean isMin = false;

        static long lastSampleInMilliseconds = 0;
        public static float final_EMG = 0;
        public int movingAveragesQTY = 30;
        public static float A_min = 1.0f;
        public static float A_max = 80.0f;

        public static float[] Calbrating_fist = new float[8] { 0, 0, 0, 0, 0, 0, 0, 0 };
        public static float[] Calbrating_rest = new float[8] { 0, 0, 0, 0, 0, 0, 0, 0 };

        public static float Truncate(float value, int digits)
        {
            double mult = Math.Pow(10.0, digits);
            double result = Math.Truncate(mult * value) / mult;
            return (float)result;
        }
        // Handling All Device Events
        internal void HandleEvent(libmyo.EventType type, DateTime timestamp, IntPtr evt)
        {
            bool outputEmgData = false;
```

```
        switch (type)
        {
            case libmyo.EventType.Emg:
                outputEmgData = true;
                SetEmgData(evt, timestamp);
                break;
            case libmyo.EventType.Connected:
                if (Connected != null)
                {
                    Connected(this, new MyoEventArgs(this, timestamp));
                }
                break;

            case libmyo.EventType.Disconnected:
                if (Disconnected != null)
                {
                    Disconnected(this, new MyoEventArgs(this, timestamp));
                }
                break;

            case libmyo.EventType.ArmSynced:
                if (ArmSynced != null)
                {
                    Arm arm = (Arm)libmyo.event_get_arm(evt);
                    XDirection xDirection = (XDirection)libmyo.event_get_x_direction(evt);
                    ArmSynced(this, new ArmSyncedEventArgs(this, timestamp, arm, xDirection));   //
khodam kardam
                }
                break;

            case libmyo.EventType.ArmUnsynced:
                if (ArmUnsynced != null)
                {
                    ArmUnsynced(this, new MyoEventArgs(this, timestamp));
                }
                break;

            case libmyo.EventType.Orientation:
                if (AccelerometerData != null)
                {
                    float x = (18 / 3) * libmyo.event_get_accelerometer(evt, 0);
                    float y = (18 / 3) * libmyo.event_get_accelerometer(evt, 1);
                    float z = (18 / 3) * libmyo.event_get_accelerometer(evt, 2);
                    var accelerometer = new Vector3(x, y, z);
                    AccelerometerData(this, new AccelerometerDataEventArgs(this, timestamp,
accelerometer));
                }
                if (GyroscopeData != null)
                {
                    float x = (18 / 3) * libmyo.event_get_gyroscope(evt, 0);
                    float y = (18 / 3) * libmyo.event_get_gyroscope(evt, 1);
                    float z = (18 / 3) * libmyo.event_get_gyroscope(evt, 2);
                    var gyroscope = new Vector3(x, y, z);
```

```
            GyroscopeData(this, new GyroscopeDataEventArgs(this, timestamp, gyroscope));
        }
        if (OrientationData != null)
        {
            float x = (18 / 3) * libmyo.event_get_orientation(evt, libmyo.OrientationIndex.X);
            float y = (18 / 3) * libmyo.event_get_orientation(evt, libmyo.OrientationIndex.Y);
            float z = (18 / 3) * libmyo.event_get_orientation(evt, libmyo.OrientationIndex.Z);
            float w = (18 / 3) * libmyo.event_get_orientation(evt, libmyo.OrientationIndex.W);
            var orientation = new Quaternion(x, y, z, w);
            OrientationData(this, new OrientationDataEventArgs(this, timestamp, orientation));
        }
        break;

    case libmyo.EventType.Pose:
        if (PoseChange != null)
        {
            var pose = (Pose)libmyo.event_get_pose(evt);
            PoseChange(this, new PoseEventArgs(this, timestamp, pose));
        }
        break;

    case libmyo.EventType.Rssi:
        if (Rssi != null)
        {
            var rssi = libmyo.event_get_rssi(evt);
            Rssi(this, new RssiEventArgs(this, timestamp, rssi));
        }
        break;
    case libmyo.EventType.Unlocked:
        if (Unlocked != null)
        {
            Unlocked(this, new MyoEventArgs(this, timestamp));
        }
        break;
    case libmyo.EventType.Locked:
        if (Locked != null)
        {
            Locked(this, new MyoEventArgs(this, timestamp));
        }
        break;
  }
  if (!outputEmgData && streamEmg) { SetEmgData(evt, timestamp); }
}
#endregion MYOsensors

//  public static int [] emg1 ;

#region vaihd

public static Boolean fistPressed = false;
public static Boolean restPressed = false;

public int counter = 1;
```

```
// Listening Myo Device Signals
protected void SetEmgData(IntPtr evt, DateTime timestamp)
{
   long milliseconds = (DateTime.Now.Ticks / TimeSpan.TicksPerMillisecond);
   if (lastSampleInMilliseconds < (milliseconds - 50))
   {
      int[] emg1 =
      {
       libmyo.event_get_emg(evt, 0),
       libmyo.event_get_emg(evt, 1),
       libmyo.event_get_emg(evt, 2),
       libmyo.event_get_emg(evt, 3),
       libmyo.event_get_emg(evt, 4),
       libmyo.event_get_emg(evt, 5),
       libmyo.event_get_emg(evt, 6),
       libmyo.event_get_emg(evt, 7)
      };

      int[] emg =
      {
       Math.Abs( libmyo.event_get_emg(evt, 0)),
       Math.Abs( libmyo.event_get_emg(evt, 1)),
       Math.Abs( libmyo.event_get_emg(evt, 2)),
       Math.Abs( libmyo.event_get_emg(evt, 3)),
       Math.Abs( libmyo.event_get_emg(evt, 4)),
       Math.Abs( libmyo.event_get_emg(evt, 5)),
       Math.Abs( libmyo.event_get_emg(evt, 6)),
       Math.Abs( libmyo.event_get_emg(evt, 7))
      };
      this.Emglistener(emg);
      float[] MA_EMG = new float[8];
      MA_EMG = movingAvarageSamplingQueueEmg;

      // total data to calibClass
        calibClass.ListAdd(new float[24] {
         emg1[0], emg1[1], emg1[2], emg1[3], emg1[4], emg1[5], emg1[6], emg1[7],
         emg[0], emg[1], emg[2], emg[3], emg[4], emg[5], emg[6], emg[7],
         MA_EMG[0], MA_EMG[1], MA_EMG[2], MA_EMG[3], MA_EMG[4], MA_EMG[5],
MA_EMG[6], MA_EMG[7] });

       Myo.saveToFile(Myo.calibClass, "all");
      // Calibration Buttons
      if (Myo.fistPressed == true)
      {
         fist_calibration_data.ListAdd(new float[8] { MA_EMG[0], MA_EMG[1], MA_EMG[2],
MA_EMG[3], MA_EMG[4], MA_EMG[5], MA_EMG[6], MA_EMG[7] });
      }

      if (Myo.restPressed == true)
      {
         rest_calibration_data.ListAdd(new float[8] { MA_EMG[0], MA_EMG[1], MA_EMG[2],
MA_EMG[3], MA_EMG[4], MA_EMG[5], MA_EMG[6], MA_EMG[7] });
      }
```

```
        this.SNR(MA_EMG);

        try
        {
            if (sumOfTotalEMG <= A_min)
            {
                sumOfTotalEMG = A_min + 1;
            }
            else if (sumOfTotalEMG >= A_max)
            {
                sumOfTotalEMG = A_max - 1;
            }
            final_EMG = (100 * (sumOfTotalEMG - A_min)) / (A_max - A_min);
        }
        catch
        {
            UnityEngine.Debug.Log("error catch");
            final_EMG = 50;
        }
        lastSampleInMilliseconds = milliseconds;
    }
}

// sampling Queue of EMG signals

public float[] movingAvarageSamplingQueueEmg = new float[8] { 0, 0, 0, 0, 0, 0, 0, 0 };
//creat new arrey used in emglistener()
Queue<int> samplingQueueemg0 = new Queue<int>();
Queue<int> samplingQueueemg1 = new Queue<int>();
Queue<int> samplingQueueemg2 = new Queue<int>();
Queue<int> samplingQueueemg3 = new Queue<int>();
Queue<int> samplingQueueemg4 = new Queue<int>();
Queue<int> samplingQueueemg5 = new Queue<int>();
Queue<int> samplingQueueemg6 = new Queue<int>();
Queue<int> samplingQueueemg7 = new Queue<int>();

List<String> TextdataRaw = new List<String>();
public float FrequentlyUsedEmg = 0;
//bool isMean = true;
float[] MASEMG = new float[8];

// Moving Average Of Signals
public void Emglistener(int[] emg)
{
    float[] OldValue = new float[8];

    //deport oldest signal of arrey
    if (samplingQueueemg0.Count >= movingAveragesQTY)
        OldValue[0] = samplingQueueemg0.Dequeue();
    if (samplingQueueemg1.Count >= movingAveragesQTY)
        OldValue[1] = samplingQueueemg1.Dequeue();
    if (samplingQueueemg2.Count >= movingAveragesQTY)
        OldValue[2] = samplingQueueemg2.Dequeue();
```

```
    if (samplingQueueemg3.Count >= movingAveragesQTY)
        OldValue[3] = samplingQueueemg3.Dequeue();
    if (samplingQueueemg4.Count >= movingAveragesQTY)
        OldValue[4] = samplingQueueemg4.Dequeue();
    if (samplingQueueemg5.Count >= movingAveragesQTY)
        OldValue[5] = samplingQueueemg5.Dequeue();
    if (samplingQueueemg6.Count >= movingAveragesQTY)
        OldValue[6] = samplingQueueemg6.Dequeue();
    if (samplingQueueemg7.Count >= movingAveragesQTY)
        OldValue[7] = samplingQueueemg7.Dequeue();

    // import new signal to arrey
    samplingQueueemg0.Enqueue(emg[0]);
    samplingQueueemg1.Enqueue(emg[1]);
    samplingQueueemg2.Enqueue(emg[2]);
    samplingQueueemg3.Enqueue(emg[3]);
    samplingQueueemg4.Enqueue(emg[4]);
    samplingQueueemg5.Enqueue(emg[5]);
    samplingQueueemg6.Enqueue(emg[6]);
    samplingQueueemg7.Enqueue(emg[7]);

    movingAvarageSamplingQueueEmg[0] = (float)samplingQueueemg0.Average();
    movingAvarageSamplingQueueEmg[1] = (float)samplingQueueemg1.Average();
    movingAvarageSamplingQueueEmg[2] = (float)samplingQueueemg2.Average();
    movingAvarageSamplingQueueEmg[3] = (float)samplingQueueemg3.Average();
    movingAvarageSamplingQueueEmg[4] = (float)samplingQueueemg4.Average();
    movingAvarageSamplingQueueEmg[5] = (float)samplingQueueemg5.Average();
    movingAvarageSamplingQueueEmg[6] = (float)samplingQueueemg6.Average();
    movingAvarageSamplingQueueEmg[7] = (float)samplingQueueemg7.Average();

    double A = samplingQueueemg0.First();
}

public static bool Noisy1 = false;
public static bool Noisy2 = false;
public static bool Noisy3 = false;
public static bool Noisy4 = false;
public static bool Noisy5 = false;
public static bool Noisy6 = false;
public static bool Noisy7 = false;
public static bool Noisy8 = false;
public float sumOfTotalEMG = 0;
public static float[] totalEMG = new float[8] { 0, 0, 0, 0, 0, 0, 0, 0 };
private readonly calibratorClass calibrationDate;

public void SNR(float[] MA_EMG)
{
    Queue<float> totalEMG = new Queue<float>();
    int Zero = 0;
    if (Noisy1 == false)
    {
        totalEMG.Enqueue(MA_EMG[0]);
    }
```

```
else
{
   totalEMG.Enqueue(Zero);
}

if (Noisy2 == false)
{
   totalEMG.Enqueue(MA_EMG[1]);
}
else
{
   totalEMG.Enqueue(Zero);
}
if (Noisy3 == false)
{
   totalEMG.Enqueue(MA_EMG[2]);
}
else
{
   totalEMG.Enqueue(Zero);
}
if (Noisy4 == false)
{
   totalEMG.Enqueue(MA_EMG[3]);
}
else
{
   totalEMG.Enqueue(Zero);
}
if (Noisy5 == false)
{
   totalEMG.Enqueue(MA_EMG[4]);
}
else
{
   totalEMG.Enqueue(Zero);
}
if (Noisy6 == false)
{
   totalEMG.Enqueue(MA_EMG[5]);
}
else
{
   totalEMG.Enqueue(Zero);
}
if (Noisy7 == false)
{
   totalEMG.Enqueue(MA_EMG[6]);
}
else
{
   totalEMG.Enqueue(Zero);
}
```

```
        if (Noisy8 == false)
        {
            totalEMG.Enqueue(MA_EMG[7]);
        }
        else
        {
            totalEMG.Enqueue(Zero);
        }

        sumOfTotalEMG = totalEMG.Sum();
    }

    public static void saveToFile(calibratorClass calibrationDate, string filename)
    {
        string path = @"D:\Exo\" + filename +".txt";
        using (StreamWriter sw = (File.Exists(path)) ? File.AppendText(path) : File.CreateText(path))
        {
            var data = calibrationDate.Ls.ToArray();
            foreach (float[] dt in data)
            {
                string line = String.Join(", ", dt.Select(p => p.ToString()).ToArray());
                sw.WriteLine(line);
            }
        }
    }
}

#endregion vahid

#region MYOoptions
public enum Arm
{
    Right,
    Left,
    Unknown
}

public enum XDirection
{
    TowardWrist,
    TowardElbow,
    Unknown
}

public enum Result
{
    Success,
    Error,
    ErrorInvalidArgument,
    ErrorRuntime
}

public enum VibrationType
```

```
    {
        Short,
        Medium,
        Long
    }
    public enum StreamEmg
    {
        Disabled,
        Enabled
    }
    public enum UnlockType
    {
        Timed = 0,  ///< Unlock for a fixed period of time.
        Hold = 1    ///< Unlock until explicitly told to re-lock.
    }
    #endregion MYOoptions
}
```

APPENDIX-I

C# SCRIPTS FOR COMUNICATION WITH MICROCONTROLLER

```
using UnityEngine;
using UnityEngine.UI;
using Thalmic.Myo;
using System;
using System.IO.Ports;
using System.Globalization;
using System.Collections;

public class Oriant : MonoBehaviour
{

    #region
    public Myo Myo_finalData;

    public Transform ThumbR0;
    public Transform ThumbR1;
    public Transform ThumbR2;

    public Transform IndexR0;
    public Transform IndexR1;
    public Transform IndexR2;

    public Transform MiddleR0;
    public Transform MiddleR1;
    public Transform MiddleR2;

    public Transform RingR0;
    public Transform RingR1;
    public Transform RingR2;

    public Transform LittleR0;
    public Transform LittleR1;
    public Transform LittleR2;

    public static float MotorThumb;
```

```
public static float MotorIndex;
public static float MotorMiddle;

public Slider Motor1;
public Slider Motor2;
public Slider Motor3;

public Text Thumbtxt;
public Text Indextxt;
public Text Middletxt;

public Toggle EMG;
public Toggle SLIDER;
public Toggle EXO;

#endregion
public SerialPort stream = new SerialPort("COM4", 250000);
private string[] numbers;
private void Start()
{
   try
   {
      stream.Open();
      Debug.Log("arduino Stream is open ");
   }
   catch (Exception e)
   {
      Debug.Log("Could not open serial port: " + e.Message);
   }
}

void Update()
{
    if (SLIDER.isOn == true) //gets finger value from Slider
   {
      //for the Index finger
      MotorThumb = Motor1.value;
      Thumbtxt.text = MotorThumb.ToString();
      //for the Thumb finger
      MotorIndex = Motor2.value;
      Indextxt.text = MotorIndex.ToString();
      //for the Middle finger
      MotorMiddle = Motor3.value;
      Middletxt.text = MotorMiddle.ToString();
   }
   if (EMG.isOn == true) //gets finger value from Slider
   {
      float aaaaa =  Myo.Truncate(Myo.final_EMG,2);
      //for the Thumb finger
      Thumbtxt.text = aaaaa.ToString();
      MotorThumb = aaaaa;
      //for the Index finger
      Indextxt.text = aaaaa.ToString();
```

```
            MotorIndex = aaaaa;
            //for the Middle finger
            Middletxt.text = aaaaa.ToString();
            MotorMiddle = aaaaa;
        }
        #region to VR hand
        //for the Thumb finger
        double MotorThumb1 = (MotorThumb - 0) * (35 - 3) / (100 -0) + 3;
        ThumbR0.transform.localEulerAngles = new UnityEngine.Vector3(0f, 0f, (float)MotorThumb1);
        ThumbR1.transform.localEulerAngles = new UnityEngine.Vector3(0f, 0f, (float)(31.12 *
Math.Pow((double)MotorThumb1, 0.1104)));
        ThumbR2.transform.localEulerAngles = new UnityEngine.Vector3(0f, 0f, (float)((3.47 *
MotorThumb1) - 5.89));
        IndexR0.transform.localEulerAngles = new UnityEngine.Vector3(0f, 0f, -MotorIndex);
        IndexR1.transform.localEulerAngles = new UnityEngine.Vector3(0f, 0f, (float)(-(1.64
*(double)MotorIndex + 6.18) ));
        IndexR2.transform.localEulerAngles = new UnityEngine.Vector3(0f, 0f, - MotorIndex);
        MiddleR0.transform.localEulerAngles = new UnityEngine.Vector3(0f, 0f, -MotorMiddle);
        MiddleR1.transform.localEulerAngles = new UnityEngine.Vector3(0f, 0f, -(4 / 3) * MotorMiddle);
        MiddleR2.transform.localEulerAngles = new UnityEngine.Vector3(0f, 0f, -(2 / 2) * MotorMiddle);
        //for the Ring finger
        RingR0.transform.localEulerAngles = (MiddleR0.transform.localEulerAngles) * (11 / 10);
        RingR1.transform.localEulerAngles = (MiddleR1.transform.localEulerAngles) * (11 / 10);
        RingR2.transform.localEulerAngles = (MiddleR2.transform.localEulerAngles) * (11 / 10);
        //for the Little finger
        LittleR0.transform.localEulerAngles = (RingR0.transform.localEulerAngles) * (11 / 10);
        LittleR1.transform.localEulerAngles = (RingR1.transform.localEulerAngles) * (11 / 10);
        LittleR2.transform.localEulerAngles = (RingR2.transform.localEulerAngles) * (11 / 10);
        #endregion to VR hand

        if (stream != null)
        {
            if (stream.IsOpen)
            {
                if (SLIDER.isOn == true || EMG.isOn == true )
                {
                    stream.WriteLine("<" +"0"+ "," + Thumbtxt.text + "," + Indextxt.text + "," + Middletxt.text +
"," + ballCol.ThumbCollide + "," + ballCol.IndexCollide + "," + ballCol.MiddleCollide + ">");
                    stream.Close();

                } else if (EXO.isOn == true)
                {
                    stream.WriteLine("<" +"1"+ "," + Thumbtxt.text + "," + Indextxt.text + "," + Middletxt.text +
"," + ballCol.ThumbCollide + "," + ballCol.IndexCollide + "," + ballCol.MiddleCollide + ">");

                    string _distance = stream.ReadLine();
                    numbers = _distance.Split(';');

                    float Motor_1_PositionSensor = float.Parse(numbers[0],
CultureInfo.InvariantCulture.NumberFormat);

                    float Motor_2_PositionSensor = float.Parse(numbers[1],
CultureInfo.InvariantCulture.NumberFormat);
```

```
                float Motor_3_PositionSensor = float.Parse(numbers[2],
CultureInfo.InvariantCulture.NumberFormat);
                Debug.Log(Motor_3_PositionSensor);

                //for the Thumb finger
                Thumbtxt.text = Motor_1_PositionSensor.ToString();
                MotorThumb = Motor_1_PositionSensor;
                //for the Index finger
                Indextxt.text = Motor_3_PositionSensor.ToString();
                MotorIndex = Motor_3_PositionSensor;
                //for the Middle finger
                Middletxt.text = Motor_2_PositionSensor.ToString();
                MotorMiddle = Motor_2_PositionSensor;

                stream.Close();
            }
        }
        else
        {
            stream.Open();
            stream.ReadTimeout = 16;
        }
    }
    else
    {
        if (stream.IsOpen)
        {
            Debug.Log("port is open");
        }
        else
        {
          Debug.Log("port == null");
        }
    }
  }
}
```

APPENDIX-J

SCRIPTS OF THE MICROCONTROLLER

```
//////////////////.....//////    ////////////////// recive data from c#
const byte numChars = 32;
char receivedChars[numChars];
char tempChars[numChars];        // temporary array for use when parsing
// variables to hold the parsed data
char messageFromPC[numChars] = {0};
int modeByCsharp = 1 ;
int ThumbPosByCsharp = 50;
int IndexPosByCsharp = 50;
int MiddlePosByCsharp = 50;
int mode;
boolean newData = false;
//////////////////////////////////////////// Thumb
#define Thumb_Velocity_PWM_ENB         11
#define Thumb_Driction_digital_EN4     12
#define Thumb_Driction_digital_EN3     13
#define Thumb_Position_Analog          A2
#define Thumb_ForceSensor_Analog       A3
//////////////////////////////////////////// Index
#define Index_Velocity_PWM_ENB         9
#define Index_Driction_digital_EN4     38
#define Index_Driction_digital_EN3     34
#define Index_Position_Analog          A1
#define Index_ForceSensor_Analog       A5
////////////////////////////////////////////   Middle
#define Middle_Velocity_PWM_ENB        8
#define Middle_Driction_digital_EN4    32
#define Middle_Driction_digital_EN3    36
#define Middle_Position_Analog         A0
#define Middle_ForceSensor_Analog      A4
//////////////////////////////////////////// Thumb Position
int Averaging_Thumb_Position_Value;
float _Averaging_Thumb_Position_Value = 0 ;
const int numReadingsPosition_Thumb = 16;
float readingsPosition_Thumb[numReadingsPosition_Thumb];      // the readings from the analog input
int readIndexPosition_Thumb = 0;                              // the index of the current reading
```

```
float totalPosition_Thumb = 0;                  // the running total
float Averaging_Thumb_Position = 0;             // the average
////////////////////////////////////////// Index Position
int Averaging_Index_Position_Value;
float _Averaging_Index_Position_Value = 0 ;
const int numReadingsPosition_Index = 16;
float readingsPosition_Index[numReadingsPosition_Index];      // the readings from the analog input
int readIndexPosition_Index = 0;                // the index of the current reading
float totalPosition_Index = 0;                  // the running total
float Averaging_Index_Position = 0;             // the average
////////////////////////////////////////// Middle Position
int Averaging_Middle_Position_Value;
float _Averaging_Middle_Position_Value = 0 ;
const int numReadingsPosition_Middle = 16;
float readingsPosition_Middle[numReadingsPosition_Middle];    // the readings from the analog input
int readIndexPosition_Middle = 0;               // the index of the current reading
float totalPosition_Middle = 0;                 // the running total
float Averaging_Middle_Position = 0;            // the average
////////////////////////////////////////// Thumb ForceSensor
float Averaging_Thumb_ForceSensor_Value;
float _Averaging_Thumb_ForceSensor_Value = 0 ;
const int numReadingsForce_Thumb = 20;
float readingsForce_Thumb[numReadingsForce_Thumb];          // the readings from the analog input
int readIndexForce_Thumb = 0;                   // the index of the current reading
float totalForce_Thumb = 0;                     // the running total
float Averaging_Thumb_ForceSensor = 0;          // the average
////////////////////////////////////////// Index ForceSensor
float Averaging_Index_ForceSensor_Value;
float _Averaging_Index_ForceSensor_Value = 0 ;
const int numReadingsForce_Index = 20;
float readingsForce_Index[numReadingsForce_Index];          // the readings from the analog input
int readIndexForce_Index = 0;                   // the index of the current reading
float totalForce_Index = 0;                     // the running total
float Averaging_Index_ForceSensor = 0;          // the average
////////////////////////////////////////// Middle ForceSensor
float Averaging_Middle_ForceSensor_Value;
float _Averaging_Middle_ForceSensor_Value = 0 ;
const int numReadingsForce_Middle = 20;
float readingsForce_Middle[numReadingsForce_Middle];        // the readings from the analog input
int readIndexForce_Middle = 0;                  // the index of the current reading
float totalForce_Middle = 0;                    // the running total
float Averaging_Middle_ForceSensor = 0;         // the average
////////////////////////////////////////// force feedback from VR (contacts)
int ThumbCollide = 7;
int IndexCollide = 7;
int MiddleCollide = 7;
int Tcollider = 0;
int Icollider = 0;
int Mcollider = 0;
//////////////////////////PID Speed
int ATP = 300;
int AMP = 300;
int AIP = 300;
```

```
float Thumb_error = 0;
float Middle_error = 0;
float Index_error = 0;

float Thumb_integral = 0;
float Middle_integral = 0;
float Index_integral = 0;

int Thumb_output_dutyCycle = 0;
int Middle_output_dutyCycle = 0;
int Index_output_dutyCycle = 0;

double Thumb_Kp = 16.5;
double Thumb_Ki = 0.665;
double Thumb_Kd = 0.443;

double Index_Kp = 26.4;
double Index_Ki = 0.57;
double Index_Kd = 0.380;

double Middle_Kp = 16.5;
double Middle_Ki = 0.635;
double Middle_Kd = 0.423;

const float DeadBand = 2.85; //1.5;

////////////////////////////////////////// force PID     //https://arduinoplusplus.wordpress.com/2017/06/21/pid-
    control-experiment-tuning-the-controller/
int Thumb_force_dead_band = 4;
int Middle_force_dead_band = 4;
int Index_force_dead_band = 4;

int Thumb_force_Setpoint = 450;
int Middle_force_Setpoint = 745;
int Index_force_Setpoint = 740;

float Thumb_Force_error = 0;
float Middle_Force_error = 0;
float Index_Force_error = 0;

float Thumb_Force_integral = 0;
float Middle_Force_integral = 0;
float Index_Force_integral = 0;

double Thumb_Force_Kp = 4;
double Thumb_Force_Ki = 0;
double Thumb_Force_Kd = 0.1;

double Index_Force_Kp = 5;
double Index_Force_Ki = 0;
double Index_Force_Kd = 0.1;
```

```
double Middle_Force_Kp = 6.;
double Middle_Force_Ki = 0;
double Middle_Force_Kd = 0.2;
unsigned long time1 = 0;
////////////////////////////////////////////////////////////////////////////////////////////Data streaming from C#
bool EXO_mode = false;
bool EMGandSLIDER_mode = false;
//////////////////////////////////////////////////////////////////////////////////////////////  setup
void setup()
{
 Serial.begin(250000);

 TCCR4B = TCCR4B & B11111000 | B00000001;
 TCCR2B = TCCR2B & B11111000 | B00000001;
 TCCR1B = TCCR1B & B11111000 | B00000001;

 pinMode(Thumb_Velocity_PWM_ENB, OUTPUT);
 pinMode(Thumb_Driction_digital_EN3, OUTPUT);
 pinMode(Thumb_Driction_digital_EN4, OUTPUT);
 pinMode(Thumb_Position_Analog, INPUT);
 pinMode(Thumb_ForceSensor_Analog, INPUT);

 pinMode(Index_Velocity_PWM_ENB, OUTPUT);
 pinMode(Index_Driction_digital_EN3, OUTPUT);
 pinMode(Index_Driction_digital_EN4, OUTPUT);
 pinMode(Index_Position_Analog, INPUT);
 pinMode(Index_ForceSensor_Analog, INPUT);

 pinMode(Middle_Velocity_PWM_ENB, OUTPUT);
 pinMode(Middle_Driction_digital_EN3, OUTPUT);
 pinMode(Middle_Driction_digital_EN4, OUTPUT);
 pinMode(Middle_Position_Analog, INPUT);
 pinMode(Middle_ForceSensor_Analog, INPUT);

 Thumb_force_Setpoint = analogRead(Thumb_ForceSensor_Analog);
 Middle_force_Setpoint = analogRead(Middle_ForceSensor_Analog);
 Index_force_Setpoint = analogRead(Index_ForceSensor_Analog);

 int thisReading_Thumb = 0;
 int _thisReading_Thumb = 0;
 int thisReading_Index = 0;
 int _thisReading_Index = 0;
 int thisReading_Middle = 0;
 int _thisReading_Middle = 0;

 //////<< Thumb queue >>\\\\\\
 for ( thisReading_Thumb = 0; thisReading_Thumb < numReadingsPosition_Thumb;
    thisReading_Thumb++)
  readingsPosition_Thumb[thisReading_Thumb] = 0;
 for (int _thisReading_Thumb = 0; _thisReading_Thumb < numReadingsForce_Thumb;
    _thisReading_Thumb++)
  readingsForce_Thumb[_thisReading_Thumb] = 0;
```

```
  //////<< Index queue >>\\\\\\
  for ( thisReading_Index = 0; thisReading_Index < numReadingsPosition_Index; thisReading_Index++)
    readingsPosition_Index[thisReading_Index] = 0;
  for (int _thisReading_Index = 0; _thisReading_Index < numReadingsForce_Index;
      _thisReading_Index++)
    readingsForce_Index[_thisReading_Index] = 0;
  //////<< Middle queue >>\\\\\\
  for ( thisReading_Middle = 0; thisReading_Middle < numReadingsPosition_Middle;
      thisReading_Middle++)
    readingsPosition_Middle[thisReading_Middle] = 0;
  for (int _thisReading_Middle = 0; _thisReading_Middle < numReadingsForce_Middle;
      _thisReading_Middle++)
    readingsForce_Middle[_thisReading_Middle] = 0;
}
////////////////////////////////////////////////////////////////////////////////////////// Loop
void loop()
{
  recvWithStartEndMarkers();

  if (newData == true)
  {
    strcpy(tempChars, receivedChars);
    parseData(); // split the data into its parts
    newData = false;
  }
  showParsedData();
  if (Serial.available() > 0) {
    Serial.print(ATP);
    Serial.print(";");
    Serial.print(AMP);
    Serial.print(";");
    Serial.println(AIP);
  }
  float Thumb_PositionSensor = analogRead(Thumb_Position_Analog);
  float Thumb_ForceSensor = analogRead(Thumb_ForceSensor_Analog);
  float Index_PositionSensor = analogRead(Index_Position_Analog);
  float Index_ForceSensor = analogRead(Index_ForceSensor_Analog);
  float Middle_PositionSensor = analogRead(Middle_Position_Analog);
  float Middle_ForceSensor = analogRead(Middle_ForceSensor_Analog);
  MovingAveraeForceSensor();
  MovingAveragePositionSensor();
  ////////////////////////////////////////////////////////////////////////// mapping
  int Averaging_Thumb_Position = abs(totalPosition_Thumb / numReadingsPosition_Thumb) ;
  int Averaging_Middle_Position =  abs(totalPosition_Middle / numReadingsPosition_Middle) ;
  int Averaging_Index_Position =  abs(totalPosition_Index / numReadingsPosition_Index) ;
  ATP = map(Averaging_Thumb_Position, 0, 1023, -5, 105);
  AMP = map(Averaging_Middle_Position, 0, 1023, -5, 105);
  AIP = map(Averaging_Index_Position, 0, 1023, -5, 105);
  ////////////////////////////////////////////////////////////////////////// drive with EMG or slider mode
  if (EXO_mode == true)
  {
    EMGandSLIDER_mode == false;
    int Thumb_Force_input = Thumb_force_Setpoint;
```

```
int Middle_Force_input = Middle_force_Setpoint; //band 710 to 670
int Index_Force_input = Index_force_Setpoint; //band 710 to 670

float Thumb_Force_error_last = Thumb_Force_error;
float Middle_Force_error_last = Middle_Force_error;
float Index_Force_error_last = Index_Force_error;

//Proportional Control
Thumb_Force_error = Thumb_Force_input  - Averaging_Thumb_ForceSensor;
Middle_Force_error = Middle_Force_input  - Averaging_Middle_ForceSensor;
Index_Force_error = Index_Force_input  - Averaging_Index_ForceSensor;

unsigned long time0 = time1;
time1 = millis();
unsigned long deltatime = time1 - time0;

//intigrate
Thumb_Force_integral  = ((Thumb_Force_error + Thumb_Force_error_last) / 2) * (((double)
 (deltatime)) / 1000);
Middle_Force_integral = ((Middle_Force_error + Middle_Force_error_last) / 2) * (((double)
 (deltatime)) / 1000);
Index_Force_integral = ((Index_Force_error + Index_Force_error_last) / 2) * (((double) (deltatime)) /
 1000);
//derivative
float Thumb_Force_derivative = (Thumb_Force_error - Thumb_Force_error_last)  / (((double)
 (deltatime)) / 1000);
float Middle_Force_derivative = (Middle_Force_error - Middle_Force_error_last) / (((double)
 (deltatime)) / 1000);
float Index_Force_derivative = (Index_Force_error - Index_Force_error_last) / (((double) (deltatime)) /
 1000);
Thumb_output_dutyCycle = ((Thumb_Force_Kp * Thumb_Force_error) + (Thumb_Force_Ki *
 Thumb_Force_integral)) + (Thumb_Force_Kd * Thumb_Force_derivative);
if (Thumb_output_dutyCycle > 255  )
{
  Thumb_output_dutyCycle = 255 ;
}
else if ( Thumb_output_dutyCycle < -255 )
{
  Thumb_output_dutyCycle = -255;
}
else if (abs(Thumb_output_dutyCycle) < ((int)(35)))
{
  analogWrite(Thumb_output_dutyCycle, 0);
  digitalWrite(Thumb_Driction_digital_EN4, LOW);
  digitalWrite(Thumb_Driction_digital_EN3, LOW);
}
Middle_output_dutyCycle = ((Middle_Force_Kp * Middle_Force_error) + (Middle_Force_Ki *
 Middle_Force_integral)) + (Middle_Force_Kd * Middle_Force_derivative)
if (Middle_output_dutyCycle > 255  )
{
  Middle_output_dutyCycle = 255 ;
}
else if ( Middle_output_dutyCycle < -255 )
```

```
{
  Middle_output_dutyCycle = -255;
}
else if (abs(Middle_output_dutyCycle) < ((int)(35)))
{
  analogWrite(Middle_Velocity_PWM_ENB, 0);
  digitalWrite(Middle_Driction_digital_EN4, LOW);
  digitalWrite(Middle_Driction_digital_EN3, LOW);
}
Index_output_dutyCycle = ((Index_Force_Kp * Index_Force_error) + (Index_Force_Ki *
 Index_Force_integral)) + (Index_Force_Kd * Index_Force_derivative);
if (Index_output_dutyCycle > 255  )
{
  Index_output_dutyCycle = 255 ;
}
else if ( Index_output_dutyCycle < -255 )
{
  Index_output_dutyCycle = -255;
}
else if (abs(Index_output_dutyCycle) < ((int)(35)))
{
  analogWrite(Index_output_dutyCycle, 0);
  digitalWrite(Index_Driction_digital_EN4, LOW);
  digitalWrite(Index_Driction_digital_EN3, LOW);
}
int Thumb_Force_derection;
int Middle_Force_derection;
int Index_Force_derection;
////////////////////////////////////////////////////////////////////// drive thumb
if ( abs(Thumb_Force_error) <= Thumb_force_dead_band || Tcollider == 1   )
{
  analogWrite(Thumb_Velocity_PWM_ENB, 0);
  digitalWrite(Thumb_Driction_digital_EN4, LOW);
  digitalWrite(Thumb_Driction_digital_EN3, LOW);
}
else
{
  if  ( Thumb_Force_error  < 0 || AIP > 80 || Tcollider == 2 ) ////////////////////
  {
    analogWrite(Thumb_Velocity_PWM_ENB,  abs(Thumb_output_dutyCycle) );
    digitalWrite(Thumb_Driction_digital_EN4, HIGH);
    digitalWrite(Thumb_Driction_digital_EN3, LOW);
  }
  else if (Thumb_Force_error > 0 || AIP < 10   )
  {
    analogWrite(Thumb_Velocity_PWM_ENB, abs(Thumb_output_dutyCycle));
    digitalWrite(Thumb_Driction_digital_EN3, HIGH);
    digitalWrite(Thumb_Driction_digital_EN4, LOW);
  }
}
////////////////////////////////////////////////////////////////////// drive middle
if ( abs(Middle_Force_error) <= Middle_force_dead_band  || Mcollider == 1   )
{
```

```
  analogWrite(Middle_Velocity_PWM_ENB, 0);
  digitalWrite(Middle_Driction_digital_EN4, LOW);
  digitalWrite(Middle_Driction_digital_EN3, LOW);
 }
 else
 {
  if  ( Middle_Force_error < 0 || AMP > 90 || Mcollider == 2   )
  {
   analogWrite(Middle_Velocity_PWM_ENB,  abs(Middle_output_dutyCycle) );
   digitalWrite(Middle_Driction_digital_EN4, HIGH);
   digitalWrite(Middle_Driction_digital_EN3, LOW);
  }
  else if (Middle_Force_error > 0 || AMP < 10   )
  {
   analogWrite(Middle_Velocity_PWM_ENB, abs(Middle_output_dutyCycle) );
   digitalWrite(Middle_Driction_digital_EN3, HIGH);
   digitalWrite(Middle_Driction_digital_EN4, LOW);
  }
 }
 ////////////////////////////////////////////////////////////////drive Index
 if ( abs(Index_Force_error) <= Index_force_dead_band  || Icollider == 1   )
 {
  analogWrite(Index_Velocity_PWM_ENB, 0);
  digitalWrite(Index_Driction_digital_EN4, HIGH);
  digitalWrite(Index_Driction_digital_EN3, LOW);
 }
 else
 {
  if  ( Index_Force_error  < 0 || AIP > 90 || Icollider == 2   )
  {
   analogWrite(Index_Velocity_PWM_ENB,  abs(Index_output_dutyCycle) );
   digitalWrite(Index_Driction_digital_EN4, HIGH);
   digitalWrite(Index_Driction_digital_EN3, LOW);
  }
  else if (Index_Force_error > 0 || AIP < 10   )
  {
   analogWrite(Index_Velocity_PWM_ENB, abs(Index_output_dutyCycle));
   digitalWrite(Index_Driction_digital_EN3, HIGH);
   digitalWrite(Index_Driction_digital_EN4, LOW);
  }
  else
  {
   analogWrite(Index_Velocity_PWM_ENB, 0);
   analogWrite(Middle_Velocity_PWM_ENB, 0);
   analogWrite(Thumb_Velocity_PWM_ENB, 0);
  }
 }
 ////////////////////////////////////////////////////////////////END////////////////////////////////////////////////////////////////////////////////
  drive with EXO mode
}
if (EMGandSLIDER_mode == true )
{
 EXO_mode == false;
```

```
int Thumb_input = ThumbPosByCsharp;       //////////<<<<<<<<<get from C#
int Middle_input = MiddlePosByCsharp;
int Index_input = IndexPosByCsharp;
float Thumb_error_last = Thumb_error;
float Middle_error_last = Middle_error;
float Index_error_last = Index_error;
//Proportional Control
Thumb_error = Thumb_input - ATP;
Middle_error = Middle_input - AMP;
Index_error = Index_input - AIP;
unsigned long time0 = time1;
time1 = millis();
unsigned int deltatime = (time1 - time0);
//derivative
float Thumb_derivative = (Thumb_error - Thumb_error_last) / (((double) (deltatime)) / 1000);
float Middle_derivative = (Middle_error - Middle_error_last) / (((double) (deltatime)) / 1000);
float Index_derivative = (Index_error - Index_error_last) / (((double) (deltatime)) / 1000);
//intigrate
Thumb_integral = ((Thumb_error + Thumb_error_last) / 2) * (((double) (deltatime)) / 1000);
Middle_integral  = ((Middle_error + Middle_error_last) / 2) * (((double) (deltatime)) / 1000);
Index_integral  = ((Index_error + Index_error_last) / 2) * (((double) (deltatime)) / 1000);
//PID output ;     dutyCycle duty is from -100 (100% backward) to +100 (100% forward)
int Thumb_output_dutyCycle = ((Thumb_Kp * Thumb_error) + (Thumb_Ki * Thumb_integral)) +
 (Thumb_Kd * Thumb_derivative);
if (Thumb_output_dutyCycle > 255  )
{
 Thumb_output_dutyCycle = 255 ;
}
else if ( Thumb_output_dutyCycle < -255 )
{
 Thumb_output_dutyCycle = -255;
}
else if (abs(Thumb_output_dutyCycle) < ((int)(255 / 6)))
{
 analogWrite(Thumb_output_dutyCycle, 0);
 digitalWrite(Thumb_Driction_digital_EN4, LOW);
 digitalWrite(Thumb_Driction_digital_EN3, LOW);
}
int Middle_output_dutyCycle = ((Middle_Kp * Middle_error) + (Middle_Ki * Middle_integral)) +
 (Middle_Kd * Middle_derivative);
if (Middle_output_dutyCycle > 255  )
{
 Middle_output_dutyCycle = 255 ;
}
else if ( Middle_output_dutyCycle < -255 )
{
 Middle_output_dutyCycle = -255;
}
else if (abs(Middle_output_dutyCycle) < ((int)(255 / 6)))
{
 analogWrite(Middle_Velocity_PWM_ENB, 0);
 digitalWrite(Middle_Driction_digital_EN4, LOW);
 digitalWrite(Middle_Driction_digital_EN3, LOW);
```

```
}
int Index_output_dutyCycle = ((Index_Kp * Index_error) + (Index_Ki * Index_integral)) + (Index_Kd
 * Index_derivative);
if (Index_output_dutyCycle > 255  )
{
 Index_output_dutyCycle = 255 ;
}
else if ( Index_output_dutyCycle < -255 )
{
 Index_output_dutyCycle = -255;
}
else if (abs(Index_output_dutyCycle) < ((int)(255 / 6)))
{
 analogWrite(Index_output_dutyCycle, 0);
 digitalWrite(Index_Driction_digital_EN4, LOW);
 digitalWrite(Index_Driction_digital_EN3, LOW);
}
////////////////////////////////////////////////////////////// drive thumb
if ( abs(Thumb_error) <= DeadBand ) // || abs( Averaging_Thumb_ForceSensor -
 Thumb_force_Setpoint) < Thumb_force_dead_band )
{
 analogWrite(Thumb_Velocity_PWM_ENB, 0);
}
else
{
 if  (Thumb_error < 0 || ATP > 90 )
  {
   //analogWrite(Thumb_Velocity_PWM_ENB, map(abs(Thumb_output_dutyCycle) , 0 , 100 , 0 ,
 255));
   analogWrite(Thumb_Velocity_PWM_ENB, abs(Thumb_output_dutyCycle));
   digitalWrite(Thumb_Driction_digital_EN3, LOW);
   digitalWrite(Thumb_Driction_digital_EN4, HIGH);
  }
 else if (Thumb_error > 0 || ATP < 10 )
  {
   analogWrite(Thumb_Velocity_PWM_ENB, abs(Thumb_output_dutyCycle));
   digitalWrite(Thumb_Driction_digital_EN4, LOW);
   digitalWrite(Thumb_Driction_digital_EN3, HIGH);
  }
}
//////////////////////////////////////////////////////////////drive middle
if (abs(Middle_error) <= DeadBand  ) //|| abs(Averaging_Middle_ForceSensor -
 Middle_force_Setpoint) < Middle_force_dead_band )
{
 analogWrite(Middle_Velocity_PWM_ENB, 0);
}
else
{
 if  (Middle_error < 0 || AMP > 90)
  {
   analogWrite(Middle_Velocity_PWM_ENB, abs(Middle_output_dutyCycle));
   digitalWrite(Middle_Driction_digital_EN4, HIGH);
   digitalWrite(Middle_Driction_digital_EN3, LOW);
```

```
    }
    else if (Middle_error >  0 || AMP < 10 )
    {
     analogWrite(Middle_Velocity_PWM_ENB, abs(Middle_output_dutyCycle));
     digitalWrite(Middle_Driction_digital_EN3, HIGH);
     digitalWrite(Middle_Driction_digital_EN4, LOW);
    }
   }
   ////////////////////////////////////////////////////////////////drive Index
   if (abs(Index_error) <= DeadBand   )// || abs(Averaging_Index_ForceSensor - Index_force_Setpoint) <
    Index_force_dead_band )
   {
    analogWrite(Index_Velocity_PWM_ENB, 0);
   }
   else
   {
    if  (Index_error < 0 || AIP > 90)
    {
     analogWrite(Index_Velocity_PWM_ENB, abs(Index_output_dutyCycle));
     digitalWrite(Index_Driction_digital_EN4, HIGH);
     digitalWrite(Index_Driction_digital_EN3, LOW);
    }
    else if (Index_error > 0 || AIP < 10)
    {
     analogWrite(Index_Velocity_PWM_ENB, abs(Index_output_dutyCycle));
     digitalWrite(Index_Driction_digital_EN3, HIGH);
     digitalWrite(Index_Driction_digital_EN4, LOW);
    }
   }
   ////////////////////////////////////////////////////////////////END////////////////////////////////////////////////////////////////////////
    drive with EMG or slider mode
  }
  delay(10);
  Serial.flush();
}
/////////////////////////////////////////////////////////////////////////////// Data transmit scripts
void recvWithStartEndMarkers() {
  static boolean recvInProgress = false;
  static byte ndx = 0;
  char startMarker = '<';
  char endMarker = '>';
  char rc;

  while (Serial.available() > 0 && newData == false) {
   rc = Serial.read();
   if (recvInProgress == true) {
    if (rc != endMarker) {
     receivedChars[ndx] = rc;
     ndx++;
     if (ndx >= numChars) {
      ndx = numChars - 1;
     }
    }
```

```
      else {
        receivedChars[ndx] = '\0'; // terminate the string
        recvInProgress = false;
        ndx = 0;
        newData = true;
      }
    }
    else if (rc == startMarker) {
      recvInProgress = true;
    }
  }
}
void parseData() {      // split the data into its parts
  ////////////////////////////////////////// vr fb
  int ThumbCollideOld = ThumbCollide;
  int IndexCollideOld = IndexCollide;
  int MiddleCollideOld = MiddleCollide;
  //////////////////////////////////////////vr fb
  char * strtokIndx; // this is used by strtok() as an index

  strtokIndx = strtok(tempChars, ","); // this continues where the previous call left off
  modeByCsharp = atoi(strtokIndx);     // convert this part to an integer

  strtokIndx = strtok(NULL, ","); // this continues where the previous call left off
  ThumbPosByCsharp = atoi(strtokIndx);     // convert this part to an integer

  strtokIndx = strtok(NULL, ",");
  IndexPosByCsharp = atoi(strtokIndx);     // convert this part to an integer

  strtokIndx = strtok(NULL, ",");
  MiddlePosByCsharp = atoi(strtokIndx);     // convert this part to an integer

  strtokIndx = strtok(NULL, ",");
  ThumbCollide = atoi(strtokIndx);     // detects virtual collides with thunb

  strtokIndx = strtok(NULL, ",");
  IndexCollide = atoi(strtokIndx);     // detects virtual collides with Index

  strtokIndx = strtok(NULL, ",");
  MiddleCollide = atoi(strtokIndx);     // detects virtual collides with Middle

  if ( modeByCsharp == 0 )
  {
    EXO_mode = false;
    EMGandSLIDER_mode = true;
  } else
  {
    EMGandSLIDER_mode = false;
    EXO_mode = true;
  }
  ////////////////////// thumb VR FB
  if ( ThumbCollide == 6 )   {
    Tcollider = 2;//  go back
```

```
 }
 else if ( ThumbCollide == 7 && ThumbCollideOld == 6  )   {
  Tcollider = 1;// Stop
 }
 else   {
  Tcollider = 0;//do nothing
 }
 ////////////////////// Index VR FB
 if ( IndexCollide == 6 )        {
  Icollider = 2;
 }  //  go back        }
 else if ( IndexCollide == 7 && IndexCollideOld == 6  )        {
  Icollider = 1;          // Stop
 }
 else        {
  Icollider = 0;         //do nothing
 }
 ////////////////////// Middle VR FB
 if ( MiddleCollide == 6 )        {
  Mcollider = 2;          //  go back
 }
 else if ( MiddleCollide == 7 && MiddleCollideOld == 6  )        {
  Mcollider = 1;       // Stop
 }
 else        {
  Mcollider = 0;        //do nothing
 }
}
void showParsedData()
{
 delay(15);
}
void MovingAveraeForceSensor() {
 ////////////////////////////////////////////////////////////////////////////////////////////// Thumb ForceSensor
 totalForce_Thumb = totalForce_Thumb - readingsForce_Thumb[readIndexForce_Thumb];         //
    subtract the last reading
 readingsForce_Thumb[readIndexForce_Thumb] = analogRead(Thumb_ForceSensor_Analog);        //
    read from the sensor
 totalForce_Thumb = totalForce_Thumb + readingsForce_Thumb[readIndexForce_Thumb];         // add
    the reading to the total
 readIndexForce_Thumb = readIndexForce_Thumb + 1;                                         // advance to the next
    position in the array
 if (readIndexForce_Thumb >= numReadingsForce_Thumb)                                      // if we're at the end
    of the array...
 {
  readIndexForce_Thumb = 0;                                                               // ...wrap around to the beginning:
 }
 Averaging_Thumb_ForceSensor = abs(totalForce_Thumb / numReadingsForce_Thumb);
 ////////////////////////////////////////////////////////////////////////////////////////////// Index ForceSensor
 totalForce_Index = totalForce_Index - readingsForce_Index[readIndexForce_Index];         // subtract the
    last reading
 readingsForce_Index[readIndexForce_Index] = analogRead(Index_ForceSensor_Analog);        // read
    from the sensor
```

```
  totalForce_Index = totalForce_Index + readingsForce_Index[readIndexForce_Index];          // add the
    reading to the total
  readIndexForce_Index = readIndexForce_Index + 1;                                          // advance to the next
    position in the array
  if (readIndexForce_Index >= numReadingsForce_Index)                                       // if we're at the end of
    the array...
  {
   readIndexForce_Index = 0;                                                                // ...wrap around to the beginning:
  }
  Averaging_Index_ForceSensor = abs(totalForce_Index / numReadingsForce_Index);
  ////////////////////////////////////////////////////////////////////////////////////////// Middle ForceSensor
  totalForce_Middle = totalForce_Middle - readingsForce_Middle[readIndexForce_Middle];      // subtract
    the last reading
  readingsForce_Middle[readIndexForce_Middle] = analogRead(Middle_ForceSensor_Analog);      // read
    from the sensor
  totalForce_Middle = totalForce_Middle + readingsForce_Middle[readIndexForce_Middle];      // add the
    reading to the total
  readIndexForce_Middle = readIndexForce_Middle + 1;                                        // advance to the next
    position in the array
  if (readIndexForce_Middle >= numReadingsForce_Middle)                                     // if we're at the end
    of the array...
  {
   readIndexForce_Middle = 0;                                                               // ...wrap around to the beginning:
  }
  Averaging_Middle_ForceSensor = abs(totalForce_Middle / numReadingsForce_Middle);
}
void MovingAveragePositionSensor() {
  ////////////////////////////////////////////////////////////////////////////////////////// Thumb PositionSensor
  totalPosition_Thumb = totalPosition_Thumb - readingsPosition_Thumb[readIndexPosition_Thumb];  //
    subtract the last reading
  readingsPosition_Thumb[readIndexPosition_Thumb] = analogRead(Thumb_Position_Analog);        //
    read from the sensor
  totalPosition_Thumb = totalPosition_Thumb + readingsPosition_Thumb[readIndexPosition_Thumb];
    // add the reading to the total
  readIndexPosition_Thumb = readIndexPosition_Thumb + 1;                                    // advance to the
    next position in the array
  if (readIndexPosition_Thumb >= numReadingsPosition_Thumb)                                 // if we're at the
    end of the array...
  {
   readIndexPosition_Thumb = 0;                                                             // ...wrap around to the
    beginning:
  }
  ////////////////////////////////////////////////////////////////////////////////////////// Index PositionSensor
  totalPosition_Index = totalPosition_Index - readingsPosition_Index[readIndexPosition_Index];  //
    subtract the last reading
  readingsPosition_Index[readIndexPosition_Index] = analogRead(Index_Position_Analog);        // read
    from the sensor
  totalPosition_Index = totalPosition_Index + readingsPosition_Index[readIndexPosition_Index];        //
    add the reading to the total
  readIndexPosition_Index = readIndexPosition_Index + 1;                                    // advance to the
    next position in the array
  if (readIndexPosition_Index >= numReadingsPosition_Index)                                 // if we're at the
    end of the array...
```

```
 {
  readIndexPosition_Index = 0;                                    // ...wrap around to the beginning:
 }
 ////////////////////////////////////////////////////////////////////////////////////////// Middle PositionSensor
 totalPosition_Middle = totalPosition_Middle - readingsPosition_Middle[readIndexPosition_Middle];  //
    subtract the last reading
 readingsPosition_Middle[readIndexPosition_Middle] = analogRead(Middle_Position_Analog);        //
    read from the sensor
 totalPosition_Middle = totalPosition_Middle + readingsPosition_Middle[readIndexPosition_Middle];
    // add the reading to the total
 readIndexPosition_Middle = readIndexPosition_Middle + 1;                                       // advance to the
    next position in the array
 if (readIndexPosition_Middle >= numReadingsPosition_Middle)                                    // if we're at the
    end of the array...
 {
  readIndexPosition_Middle = 0;                                   // ...wrap around to the
    beginning:
 }
}
```

CURRICULUM VITAE

PERSONAL INFORMATION

Name Surname : Mir Vahid AHMADIPOURI NAEIM

Date of birth and place : Sep 20 1985 , Tabriz - Iran

Foreign Languages : Persian , English

E-mail : v.ahmadipour@live.com

EDUCATION

Degree	Department	University	Date of Graduation
Undergraduate	Manufacturing & Production	Tabriz Technical Institute	2009
High School	Mathematics and Physics	Gazi Tabatabayi (Tabriz, Iran)	2004

WORK EXPERIENCE

Year	Corporation/Institute	Enrollment
2013	Iran Tractor Manufacturing Company	R&D Engineer
2012	Sahand Screws Co.	QC Engineer
2010	Tabriz Petrochemical Complex	Project Manager
2011	Toseye Sanaye Karbordi Inc.	CAD Designer
2010	Mobtakeran Toseye Pooya Inc.	CAD Designer
2006	Iran Tractor Manufacturing Co.	CAD Designer

PUBLISHMENTS

Conference Papers

1. Ahmadipourinaeim M., Yilmaz C., "Determining the Relation between Flexion Angles of Finger Joints to be Used in a Virtual 3D Hand with EMG Signal Interactions", IMSMATEC'18 International Conference on Material Science, Mechanical and Automotive Engineerings and Technology, İZMİR, TURKEY, 2018, vol.1, no.1, pp.35-40

Projects

1. Research on Adaptive Control Systems and Their Application in Machinery, Tabriz College Technical, Tabriz, Iran (2009)
2. Design & manufacturing a machine to cut the shell & tubes of heat exchangers, Tabriz Petrochemical Complex (TPC), Tabriz, Iran, (2011-2012)
3. Semi-automatic machine of rug burnishing (Iran Patent Org., Number: 43529)
4. Accurate Hydraulic Feed System with Electromagnetically Control (Iran Patent Org., Number: 58856)
5. Real Time Viscosity Monitoring & Control System by Measuring the Current (Iran Patent Org., Number: 61386)
6. Adaptive controlled Band saw cutting machine to cut shell & Tube Bundle of Heat Exchangers with diameter under 2meters (Iran Patent Org., Number: 75960)
7. Lagari Team member from Yildiz Technical University for CanSat Competition 2016 (2015-2016)

Funding and Awards

1. Semi-automatic machine of rug burnishing (Iran Patent Org., Number: 43529)
2. Funding award by National Elites Foundation of Iran for “Scientific Innovation Certificate for Real Time Viscosity” project
3. Funding award by National Elites Foundation of Iran for “Real Time Viscosity Monitoring & Control System by Measuring the Current” project
4. Funding award by National Elites Foundation of Iran for “Adaptive controlled Band saw cutting machine to cut shell & Tube Bundle of Heat Exchangers with diameter under 2meters” project