Development and Control of a Shoulder Joint for Humanoid Robotics Application

by

Zhenis Otarbay

Submitted to the Department of Robotics and Mechatronics
in partial fulfillment of the requirements for the degree of

Master of Science in Robotics

at the

NAZARBAYEV UNIVERSITY

Apr 2020

Author .....
Department of Robotics and Mechatronics
Apr 29, 2020

Certified by .....
Michele Folgheraiter (Thesis supervisor)
Associate Professor
Thesis Supervisor

Certified by .....
Berdakh Abidullaev (co-supervisor)
Assistant professor
Thesis Supervisor

Certified by .....
Mohamad Mosadeghzad (co-supervisor)
Assistant professor
Thesis Supervisor

Accepted by .....
Vassilios D. Tourassis
Dean, School of Engineering and Digital Sciences

Abstract

Humanoid robotics represents an important sub-field of Robotics that has many applications, such as house assistance, surveillance, medical assistance, dangerous environments, repetitive works, and rehabilitation. Human joints are difficult to replicate in a humanoid robot. The available biological models may differ from the real-life robotic models. This work aims at modeling, building, and controlling a low-inertia, high-stiffness, tendon-driven shoulder joint. Force and position sensors are integrated for further study of the system dynamics. A Python-V-REP model of the joint is developed to study the joint motion and to optimize its design. Also, a Python interface is used to visualize the control mechanism and to represent the force sensor and magnetic encoder values that facilitate monitoring and controlling the actuators' status. We explored the possibility to use EEG (Electroencephalogram) signals to control the robotic joint, taking into account possible applications in telerobotics and as prosthesis. A parallel manipulator with three curved limbs can be used in a humanoid robots as a shoulder and wrist joints. Describing the kinematics of this type of platform is a challenging process. This work tried also to examine two methods to find the direct and inverse kinematics of the parallel manipulator. The successive screw method is applied to one limb of the manipulator with the assumption that the other two legs will follow the first one. The geometric method describes a change in the height of the manipulator where pulleys are located.

Acknowledgments

At first, I would like to thank my supervisor professor Michele Folgheraiter for his guidance and patience. Furthermore, I would like to thank professor Berdakh Abibullaev, professor Matteo Rubagotti, professor Mohammad Mosadeghzad, professor Atakan Varol, professor Almas Shintemirov, professor Tohid Alizadeh, professor Anara Sandygulova, professor Do Duc Ton, professor Zhanat Kappasov, instructor Altay Zhakatayev, instructor Iliyas Tursynbek for helping me and for teaching me many important subjects throughout the six years of study (4 years of Bachelor studies and 2 years of Masters study) in Nazarbayev University. Finally, I would like to thank to my colleagues Sharafatdin Yessirkepov, Darkhan Zholtayev, Zaksylyk Kazykenov, Alfarabi Imashev, Fahim Raza Sunasara, Asset Yskakov, Kim Junhui and Timur Ishuov.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction to the topic

Humanoid robotics aims at developing robotics systems that manipulate objects and move in the environment in a similar way that humans do. Their kinematic architecture is meant to partially reproduce the mobility of the human skeleton. In most of the systems developed so far, a sequence of revolute joints is used to reproduce the mobility of complex articulations present in the human body [25]. For example, the human shoulder represents a complex mechanism that can apply rotation and translations to the upper limb. In this thesis work, we want to develop a tendon actuated spherical joint that can reproduce the behavior of the humerus-glenoid cavity present in the human shoulder. We will also explore, with a feasibility study, the possibility to integrate EEG signals into the control architecture to predict the motion [17].

Parallel manipulators can withstand huge loads. One type of parallel manipulator is the Stewart platform. The Stewart platforms can be used to substitute shoulder and wrist in humanoid robots. This work will try to solve direct and inverse kinematics of the Stewart platform that has 12 rotational joints, 4 in each limb. Also, we will examine the problem of finding the relationship between the end-effector's position and orientation. We have also tried to verify direct and inverse kinematics using MATLAB and the simulation was done in V-REP. Successful implementation of the Stewart platform with 12 rotational joints would help to make the humanoid robot's hand robust.

1.2 Importance of the topic

The problem we intend to solve is very important for future applications in the field of humanoid robotics. An actuated spherical joint will allow the robot to move more naturally and it also will simplify its design. Since the actuation system will be installed before the joint, the upper arm will be lighter and therefore capable to move faster and displace bigger payloads.

The possibility to integrate EEG signals in the control architecture of the artificial shoulder represents an important research question [2].

Using brain signals to assist to the control of the shoulder motion may be beneficial for many people who have lost their limbs and need a more natural control of their prosthetic arm [5] [4]. The system can also be used for teleoperation purposes, facilitating the operation of an exoskeleton (master-arm). Predicting the motion of the operator's arm could be used to reduce the stiffness of the exoskeleton and to allow a more natural operation [9].

1.3 Terminology

V-REP - is a software used to simulate the mechanical assembly to see how it performs and to analyze its movements before implementing the movements in real-life so that the shoulder is stressed bothered too much before its development is finished.

DOF - degree of freedom, characteristics of the movement of mechanical systems.

Quaternion (from Latin quaternion, four) - system of hyper-complex numbers, forming vector space with a dimension of four on the field of material numbers. It is used to decrease the calculation amount.

Forward kinematics - the process of determining parameters of connected objects (for example, kinematic pairs or kinematic chains) for approaching necessary positions, orientations, and the location of these objects.

Inverse kinematics - we have an end-effector in the case of a tendon-driven joint, and we aim to find the angles of the joints.

EEG - Electroencephalogram is a way of representing brain data.

UR5 - Universal robotics 5, a family of robot manipulators for research and industrial purposes.

PCB - printed circuit board.

KiCAD - software for designing models of PCB designs.

CNC machine - computer numerical control machine for engraving the models of PCB.

PWM signal - pulse width modulation signal.

DRV8835 - a motor driver to actuate and control the directions of 2 actuators.

1.4 Literature review

This literature review consists of four sections. In the first part, I will only talk about the design and control of shoulder joints. In the second part, I will discuss V-REP models that replicate human joints. In the third part, I will talk about the shoulder joints that are controlled by EEG signals from the brain. In the last section the robotic joint and systems which use Arduino will be discussed.

1.4.1 Control of shoulder joints survey

In this survey different designs will be compared and opposed to each other, to make 3 DOF tendon-driven shoulder joints robust. In each paper, there is some important information in torque, force, design specialties, and the most important findings.
Kim et al (2018) [7] emphasize their high speed, safe interaction with their robot with robotic joints. This is the most important side that will be analyzed in this paper that is how to make the robot joints move with high speed. In addition to it, low stiffness, and safe interaction will be considered. This paper contains quaternion representation, forward and inverse kinematics. The key point is to master them and to apply the knowledge to build a system. So, the mathematics of this paper is valuable. The authors brought their robot structure to real life and called it LIMS2-AMBIDEX.

The design of our shoulder joint initially was aimed to operate using tendons like [7] Kim and others humanoid arm's shoulder joint, however, we will also evaluate other actuation mechanisms.

Additionally, this paper uses the quaternion representation of the shoulder joint kinematics and implements the forward and inverse kinematics.

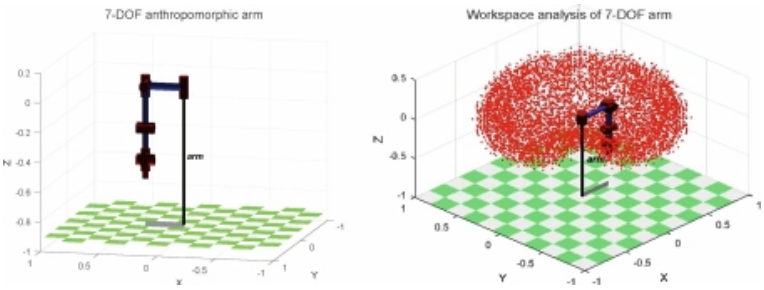


Figure 1-1: Modular design and replication human arm which is 7 DOF [6]

Ball et al (2017, August) emphasize the robotic joint's rehabilitation sides. And the design of the robotic joint is made especially safe and for rehabilitation purposes, rather than for experimentation of dangerous works, for example.

Ball et al (2007, September) also emphasize the rehabilitation side and present a robotic exoskeleton of 5 DOF and this designed mechanical joint is cable-driven. The value of these two Ball et al's papers in medical applications is important,the design of the system is appealing.
Ball et al (2007) represented a planar 3 DOF cable-driven robotic joint which is used to move shoulder and wrist for rehabilitation purposes.

Buongiorno et al (2018) [3] discuss the application of their tendon-driven differential transmission mechanism for the rehabilitation. It is important to know how differential transmission is used for this purpose. It is a bit unintuitive, but not rare in this domain. These authors developed a prototype which are worn by users.

Jiang et al (2019) present [6] their modular design and the replication of the human arm which is 7 DOF. It is more difficult to replicate a holonomic redundant system . So, it is particularly interesting to understand how this mechanism works. In the paper, it is unclear if they implement a real prototype.
Jiang et al (2019) also designed a tendon-driven 7-DOF cable-driven humanoid arm.

All the papers above represent application papers. In most of the them the concepts are proven in CAD.

Hand authors	mass	Nominal size	Number of actuators	Number of tendons/cables	Number of sensors	Number of DOF
Kim et al (2018)	Shoulder: 4.17 kg	Small size (not enough information)	~5+	~12 in one shoulder or wrist joint	~6 sensors for shoulder (10-15 sensors for whole robot)	3
Ball et al (2017, August)	No information	No information	~5+	4+	No information	3
Buongiorno et al (2018)	~4.7kg (WRES + driverbox)	Standard man's wrist side hand size	~3+	3+	6+	3
Jiang et al (2019)	No information	No information	~3+	6+	No information	7
Ball et al (2017, September)	115 kg (Actuators and exoskeleton)	exoskeleton	~5+	5+	3+	5
Folgheraiter and Otarbay* (planned to send for publishing in 2020-2021)	~2.4kg	Radius ~ 140mm and the height is about 300mm	4	3	7 (3 force, 4 magnetic encoders)	3

Figure 1-2: Survey results

Folgheraiter et al (2019) [21] focus on the implementation of Cartesian and joint space control modalities in a teach pendant. The paper is important as the propositions can be used to make the interface of our 3-DOF tendon-driven shoulder joint. This is our previous year's work and we want to add a page that controls our 3 DOF shoulder joint to the teach pendant GUI. Eventually, this pendant can control the whole NU-biped robot. This paper has its theoretical proofs, CAD model, and real device. Automation of the device was implemented using Python modules such as matplotlib, NumPy, scipy in the back end. And as a front end uses a Python Tkinter module.

Another method of assisting the human shoulder is using exoskeletons [28] is a survey paper where 71 neuro-controlled exoskeletons. The authors state that the exoskeletons mostly found usage in the military and for rehabilitation purposes. The most accepted exoskeletons are the passive ones. The people who mostly required such exoskeletons are suffering from cuts in the upper and lower limbs.

In [19] the authors present a functional system for an arm of a robot to increase the speed of the robot. The functional system looks like a PID control and the arm for which the authors try to develop its algorithm is actuated by stepper drives. They analyze

the transfer function of the system and they found an equation of the movement of the shoulder joint in the robot. They also present a regulator for the system which is based on quasi-optimal control. They also suggest an algorithm for quasi-optimal control which is similar to a control-block architecture where it has a beginning, end, derivative finding part, and optimal control part.

A simplified optimal control to understand the foundation of usage of this kind of control was suggested by [31] where the paper's mathematical side is very important and could be used if we decided to improve the speed of our shoulder joint for a humanoid robot. In this work author tries to stabilize the speed of the angle of robot-manipulator. However, the work was designed to explain the concepts of the optimal control and it may not be so much precise compared to similar works. More accurate result can be obtained in a case if some random leaps are added to the phase trajectory. The results obtained are not so much new for this field, but just supplements to the theory of optimal control of stochastic systems of differential equations in cases of discontinuities of phase trajectories.

A paper written by [24] focuses on controlling a robot-manipulator both by traditional way and by applying fuzzy logic, also they present a Matlab simulation of this kind of robotic manipulator. This means that the suggested solution could be considered as a hybrid model. The necessity of such way presents a concrete scientific and practical interest as they state in the conclusion as long as compared with methods based on only traditional algorithms of control this method is more adaptive and allows to reach the required accuracy when changing the parameters of control, to shrink the volume of the produced calculations and to increase the speed of the control system.

Identifying deformations using Solidworks is also an important part when making a humanoid robotic joint. [12] present such analysis for 3 shoulders of a robotic manipulator which is intended to be similar to a FANUC R2000iB robot. So they design a model in Solidworks and test it for deformations. The authors also show and discuss the architecture of the control with a microcontroller which they developed in their project. In this paper additionally, the electric schematics can be found. Finally, they bring the experimental model to life and test it. One more important part is that they present a geometric method to find the inverse kinematics of their manipulator at the end.

The authors in [22] present a wheeled robot with a humanoid face. But, the shoulder

of the robots is differentiated from the humanoid shoulder. The height of the robot is 120 cm, for the face and emotions the authors use LED. They optimize the printing process using 3D extruder for this robot. The robot is adapted for a presentation and initially was dedicated for presentably assisting purposes. Authors are mostly concerned with designing the heads, faces, and the shoulder of the robot.

Summary Six publication were compared and contrasted 1-2. Ball et al [1] presented relatively robust and rehabilitation aimed exoskeletons. Tendon based actuation is comprehensive and a bit more robust for shoulder or wrist joint substitute purposes. Like some of the works we reviewed here, we are also going to apply the simulation techniques for the shoulder joint before we test each of the movements in the real prototype.

1.4.2 V-REP models that replicate humanoid joints

[27] discussed a spherical joint actuated with 3 motors, 6 gears, and 6 revolute joints. The design is unique and it has simulation results such as angle and unique unit vectors. Link collisions are detected by built-in CoppelliaSim and the V-REP is connected with MATLAB to implement the model control. The V-REP model was verified using a physical model in real life.

[20] describes a soft robotic limb that is bent when a light source is received by the sensor. This soft robotic limb is also tendon driven and it also has both V-REP and physical model verification. The mathematical part is useful, especially the V-REP programming part was helpful to adapt it for our shoulder joint.

For the V-REP model workspace [29] analyzed important kinematic calculations. They suggest some systems of equations to determine the workspace of the robot. According to the authors, the obtained equations allow not only to determine the coordinates of the output but also inversely to determine the constructive characteristics with the given parameters of the workspace of the delta-robot.

Adaptive control for robot manipulator is presented by [23] where the author focuses on Puma-560 and inverted pendulum. State-space is used in dynamic control and in stabilizing the system. He also suggests a PID control for this system and the graphical output after applying the PID control. According to the author compared to the neuro-controller the adaptive control in a wide range of mass and length of the shoulder joint allows some installed accuracy, by giving required change of the angle of the manipulator

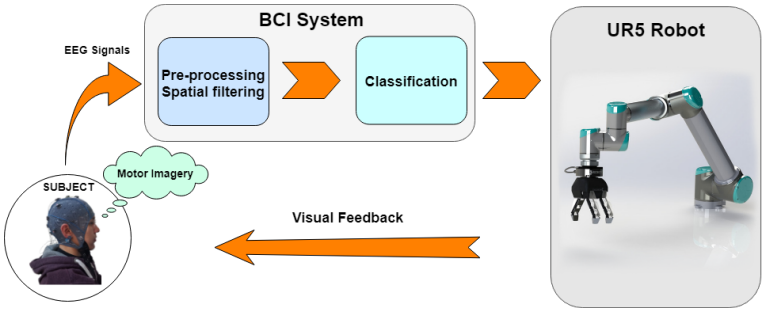


Figure 1-3: BCI-controlled-UR-manipulator [10]

as output .

1.4.3 Shoulder or robotic grippers that are controlled by EEG signals from the brain survey

In the [8] work written by Abidullaev (2020) and colleagues, the robotic arm UR5 is controlled using EEG signals acquired from the brain. The accuracy of classification to control left and right or up and down movements achieved for 7 different subjects is about 75.9% on average. This is a bit lower than what is required for real-time control and also makes the subject tired, so this makes this control mechanism not so practical.

Luu et al [18] present EEG signals from the brain to make a walking avatar. For that, they get EEG signals from the cortical part of the brain. The used dataset is EEG obtained from 64 channels for 100 Hz. It is useful for my research even though it is for walking, the general structure uses the EEG signal from the brain.
The following is an application of EEG robotic arm control using a direct brain interface.

Fu et al [15] presents an EEG control mechanism for shoulder-elbow joint by focusing on ERD for stroke survivors to assist them. One of the important parts of this research is noise rejection which we can apply in my research. They reject noises from the scalp and facial muscles.

In [13], the authors implemented EEG integrated with 9 DOF shoulder joints. I reviewed the work of Folgheraiter et. al. implementing a 9 DOF exoskeleton that controls the control system is modulated with the EEG signals. These are used to predict 300 ms in advance the intention of movement of the user and trigger the exoskeleton actuation system. This design suggests a robust electronics and safe design for its software. The voltage and the power of the hardware are also safe, thus it is highly unlikely to harm

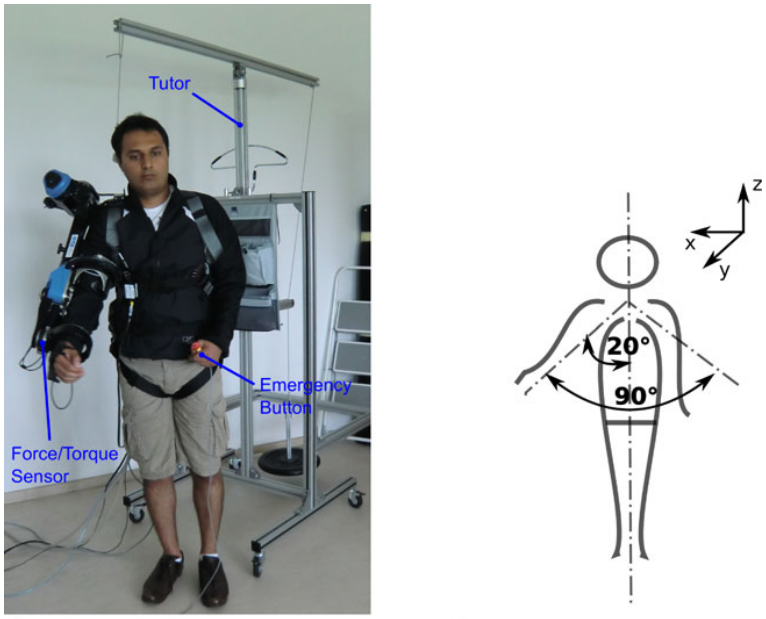


Figure 1-4: Folgheraiter et al EEG based BCI to predict motion [21]

the human subject. This research represents teleoperation-based software-hardware parts and a movement prediction technique which is one of the important parts of the research that is related to my thesis work.

Bhagat et al [10] show a feasibility study of an asynchronous EEG-based BMI for a chronic stroke. Although this method is slightly different from the method we are going to use, but it is more or less related to its general structure. This research uses combined EEG and EMG to assist the subjects' movements. Fixation is also used in this research to determine the subjects' intentions.

1.4.4 Robotic joints and systems using Arduino and teensy microcontroller

[11] examines the ways to control the robot with different software. The robot's system can overload and run for a long time, so the programming language should be as compact as possible. To do this, the language focuses on the limb.

[30] examines the ways of using robotic systems in welding. The researcher found 4 different Arduino systems and used one of them to build his robot for welding.

[14] examines a rehabilitation system with the help of robots. Evaluates the effec-

tiveness of this method. This robotic system including 4 different methods which are called designed as brain-computer exoskeleton brushes is more efficient than traditional methods of rehabilitation.

There are 4 different stimulation frequencies that were used in the experiment of SSVEP based BCI in [16]. Since the objective of the original paper, by [16], was aiming for the versatile algorithm. [16] gets participants through the offline event. The result of the paper suggested very high accuracy, 256 sampling rate may be good for the points. Like "We get high accuracy with even 256 Sample rate". For 53 subjects, the experiment had about a general average of 95.5%.

[24] focuses on controlling a robot-manipulator both, by traditional and by applying fuzzy logic, also they presented a Matlab simulation of this kind of robotic manipulator. That means they suggested a solution that could be considered as a hybrid model. The necessity of such way presents a concrete scientific and practical interest as they stated in the conclusion that as long as it is compared to methods based on only traditional control algorithms this method is more adaptive and allows to reach the required accuracy when changing the parameters of control, to shrink the volume of the produced calculations and to increase the speed of the control system.

Identifying deformations using Solidworks is also an important part when making a humanoid robotic joint. [12] present such analysis for 3 DOFs shoulders of a robotic manipulator which is intended to be similar to the FANUC R2000iB robot.

So they design up a model in Solidworks and tested it for deformations.

The authors also show and discuss the architecture of the control system with a microcontroller which they developed in their project. In addition to it, the electric schematics can also be found in this paper. Another important part is that they present a geometric method to find the inverse kinematics of their manipulator.

The authors in [22] present a wheeled robot with a humanoid face. But, the shoulder of the robots is different from the human shoulder. The height of the robot is 120 cm, for its face and emotions the authors use LED. They optimize the printing process using 3D extruder for this robot. The robot is adapted for presentation and initially was dedicated for assisting purposes. Authors are mostly concerned with designing the heads, faces, and the shoulder of the robot.

For the V-REP model workspace, [29] gives some idea and also the kinematic cal-

culations. They suggest some system of equations to determine the workspace of the robot. According to the authors, the obtained equations allow not only to determine coordinates of the output but also inversely to determine the constructive characteristics with the given parameters of the workspace of the delta-robot.

1.5 Knowledge gap (What is missing?)

Most of the researches that we listed in the literature review are not employing a spherical joint in the shoulder, but a sequence of revolute joints [25]. The payload is generally limited, especially in tendon-driven systems. Different systems are also not accurate or report just simulated models. Very few examples are available of integration of EEG signals in the control architecture. This research aims to improve the design of Kim et al's (2018) that is based on the usage of 3D printed parts.
The mechanical design and the electronics part represent the core of my thesis.

1.6 Problem statement

Part of our research is focusing on measuring the shoulder activity in a dynamic interface. We can measure three dynamic force sensor values and four magnetic encoder values where each of the magnetic encoders are attached to the actuators and each of the force sensors are attached to the shoulder parts where we can measure the forces to scheme or CAD of the force and position sensors. After this, we will focus on the V-REP simulation part to investigate how the shoulder joint performs in the virtual simulation so that to spare the real prototype.
Figure 1-5 represents the change of L in the shoulder joint for the older version of the prototype:

Direct and inverse kinematics of the Stewart platform with rotational joints and curved limbs need to be solved and verified algebraically and geometrically using MATLAB, the CAD, and simulation platforms.

The main problem is to identify the relationship between the end-effector's position and orientation and the sensors length. The challenge of solving the dynamics problem is to make sure of its correctness and its coherence with the theoretical calculations.

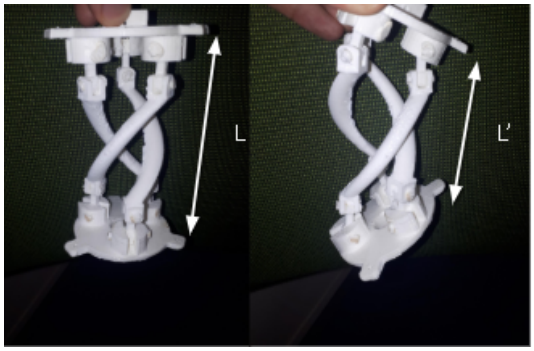


Figure 1-5: Change of L in ball-and-socket joint

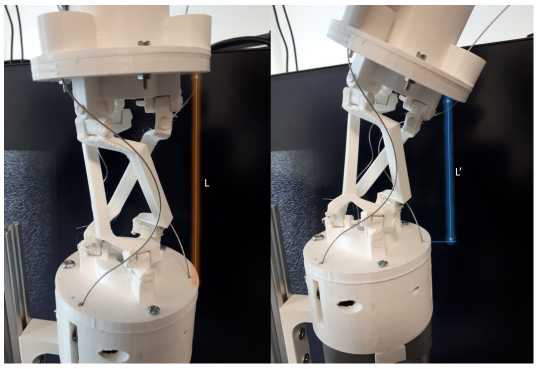


Figure 1-6: the Stewart platform with rotational joints

1.7 Potential impact

It may bring many benefits to our country in terms of science, economics, technology, and society. This research project allowed to look deeper to shoulder joint construction. Through it the effective design, brain-computer interaction, connections to device was researched. With this work robot start to have new challenges and chances, new features and programs. Limb can roll on both sides and be fixed under some degree, it controlled by touch-pad and program was made with simple control. This work prepared me for more difficult tasks.

Chapter 2

Design and implementation

As a preliminary work, firstly I have tried to make calculations for the shoulder joint and find the distance change between the fixed and moving part of the shoulder joint. Then, I made a control mechanism where I coded in Teensy 3.6 and Python the simulated model for the older version of the shoulder joint. Then, I have made a KiCAD model to engrave the PCB for the shoulder joint.
I designed a V-REP model for both of the two versions of the hardware and implemented the required constraints.

2.1 Mechanical design (Computer-Aided Design)

The model of the shoulder joint was drawn, designed, realized and assembled. Figure 2-1 represents the cad model of our shoulder joint. Solidworks was used to design the prototype and to conduct the first simulations. This way, It was possible to do a detailed design with reduced number of iterations. After the VREP model was implemented, the constraints such as max force, motor speed, torque were integrated in the V-REP model. The angular constraints or the maximum and minimum angles reach of the bend are pitch - 90 degrees and roll - 60 degrees. These are the angles at which the shoulder can rotate. 2-1 shows the views of shoulder joint assembly in Solidworks with 180 degrees curved legs $h_{leg} = 60mm$. Each leg makes 180 degrees of a curve from the base to reach the platform. The three limbs of the shoulder joint are separated at 120 degrees from each other.

The symmetric orientation of the platform compared to the base makes 60 degrees.

Parts of the SPM	Material type	Approximate in axis, mm dimensions			Quantity
		X axis	Y axis	Z axis	
Base part	PLA	140	140	8	1
Curved leg	Metal	40	10	90	3
Universal joint (plastic)	Metal	10	10	30	6
Tendon wire	platic tendon	1	1	150	3
Circular Platform	PLA	140	140	50	1
Manipulator hanger	PLA	65	20	110	1
Aluminum extrusion	Aliminuim	300	20	20	4
Clamps	PLA	10	60	50	2
motor pack	PLA	140	140	45	1
pulley	Stainless steel	10	10	3	3
Shaft pack	PLA	25	25	30	3
Motor holder	PLA	140	140	3	1
Motor	metal	38	38	75	4

Table 2.1: Parts of SPM

The 2.1 consists of shoulder joint part materials, approximate dimensions, and their quantity. The CAD model is as good as the actual 3d printed shoulder joint.

Figure 2-2 demonstrates how tendon actuation in our shoulder joint works. The yellow one which is the tendon is rotated to the head of the actuator. So, when an actuator is rotated clockwise and anticlockwise, the tendon is tightened and loosened.

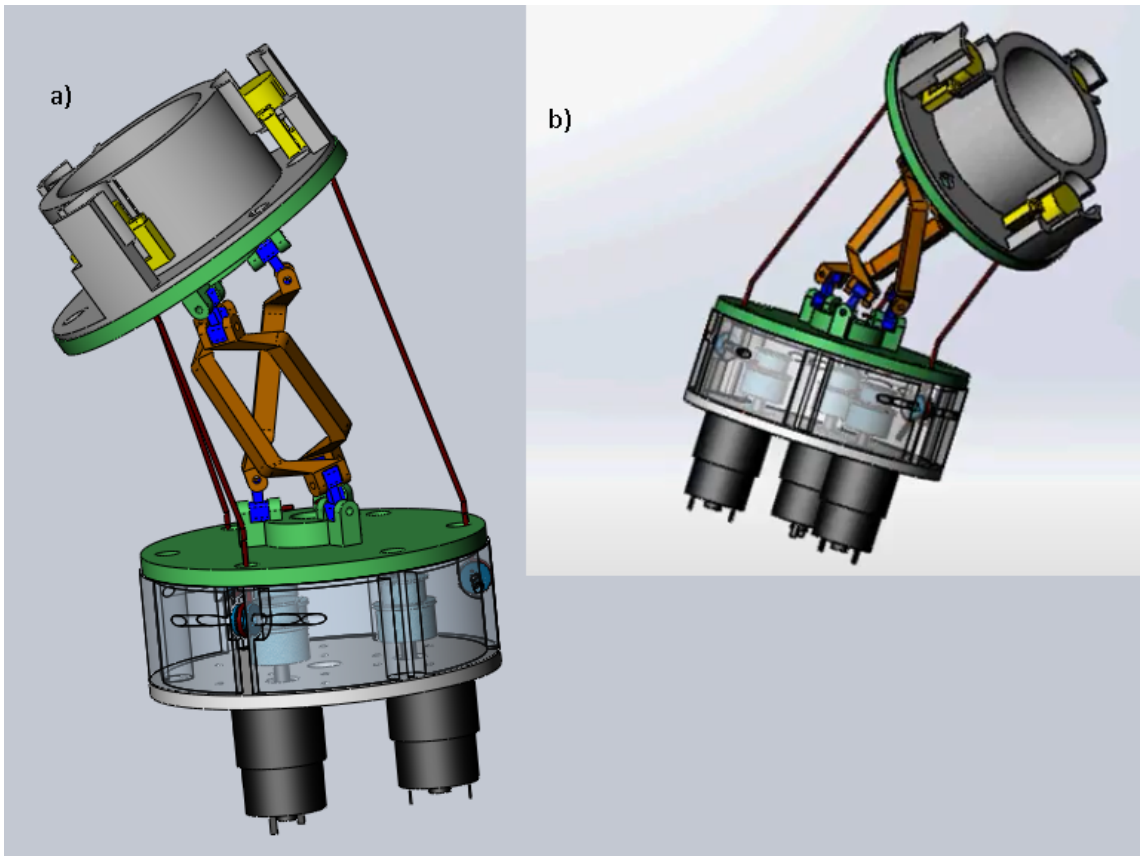


Figure 2-1: a) Views of the shoulder joint in Solidworks that resemble the functionality of the articulatio humeri assembly, b) Shoulder joint bent to the target position.

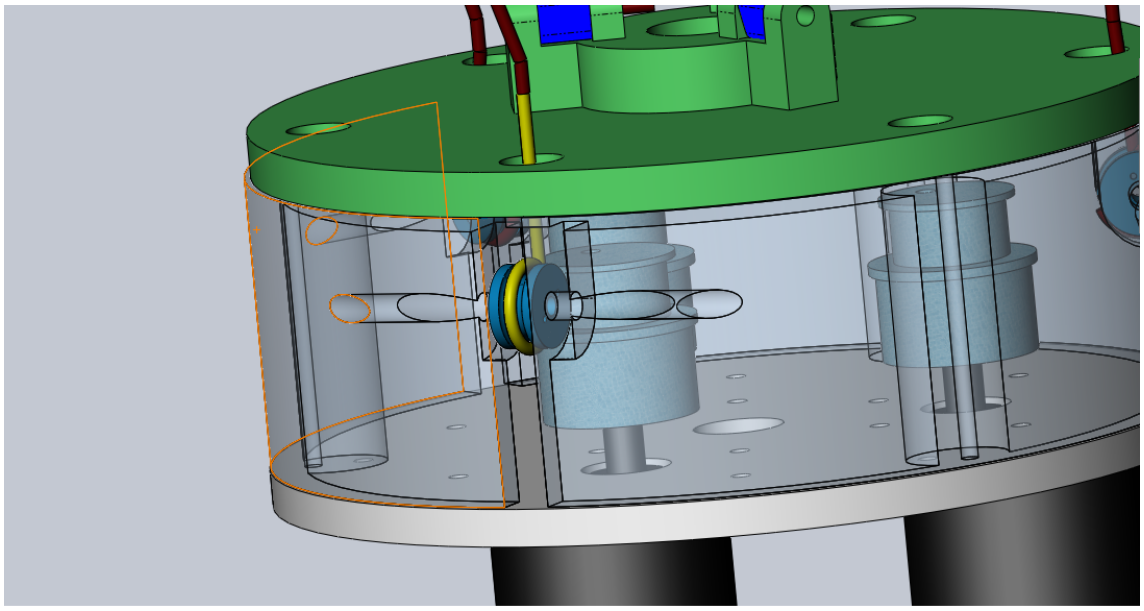


Figure 2-2: Tendon actuation

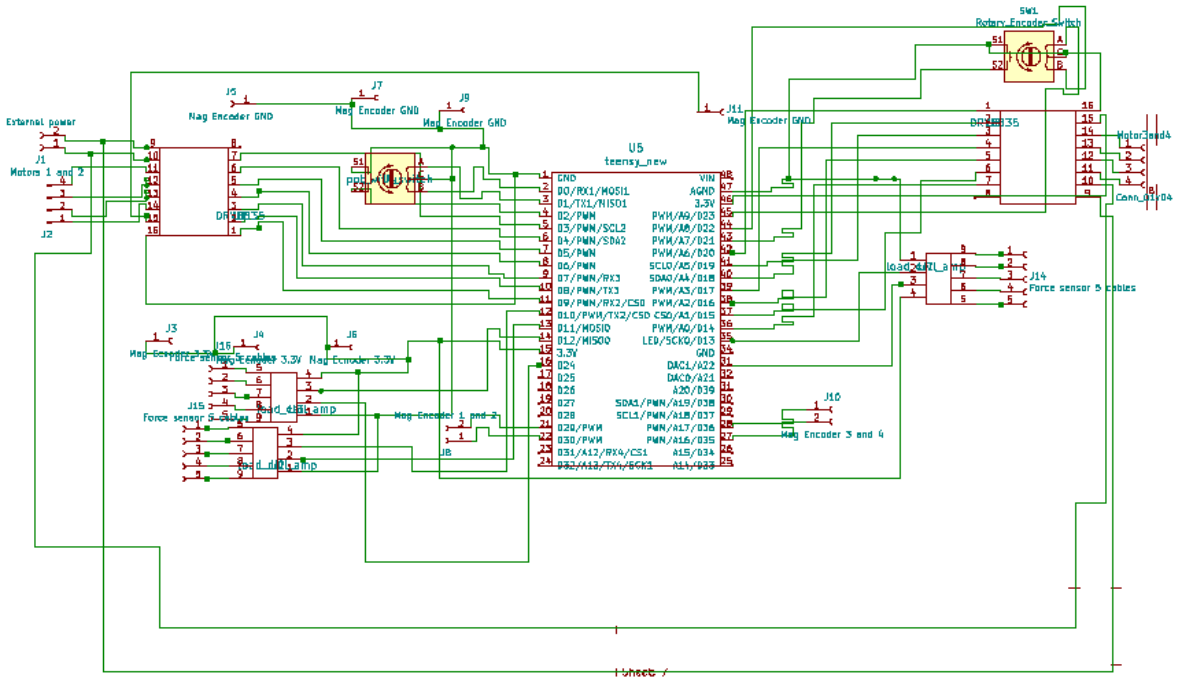


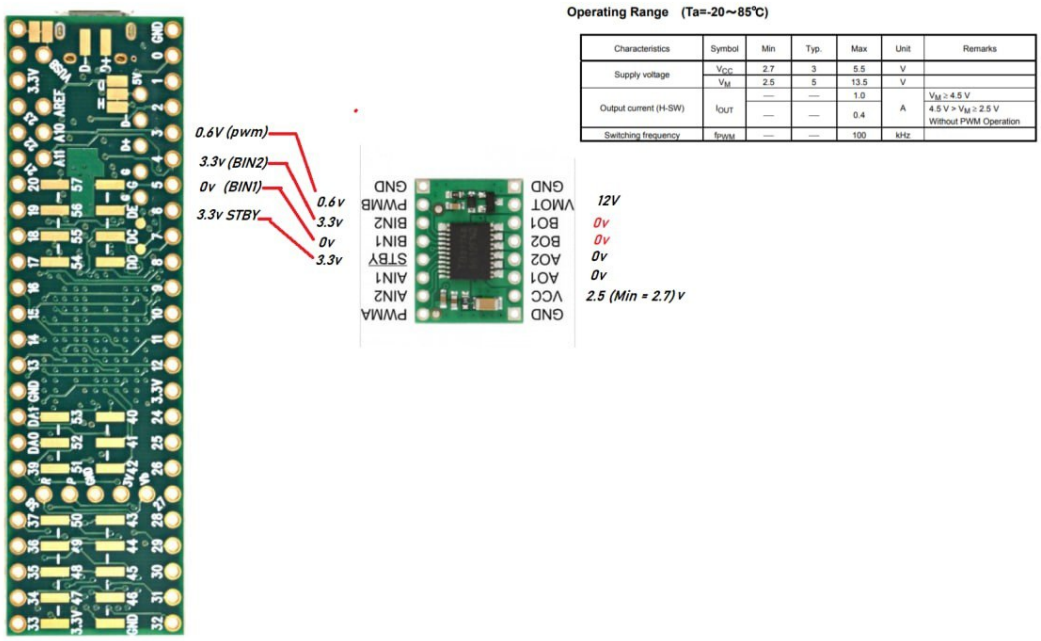
Figure 2-3: Circuit schematics design in KiCAD

2.2 Electrical design

Together with the mechanism of the shoulder joint also the electronics was developed.

Picture 2-3 shows the schematics of the circuit of the mother-board that reads the sensory information, control the actuation system and communicate with the PC where the GUI is running.

Figure 2-5 reports the KiCAD PCB of the mother-board where all the necessary electronic components and the additional electronics boards can be mounted. The PCB was built to reduce the amount of required wires to connect to the sensors and actuators. Almost all of the footprints and schematics were designed by the author of this document. I created the PCB using a CNC machine and after, to avoid the presence of short circuits, I tested all the connection on the PCB with a multi-meter. The KiCAD model is flipped for all components which allows checking the Teensy 3.5 pins easily by looking at its numbered side. This is one of the unusual sides of the design. However, this does not affect the functionality. Also, the reset button of Teensy 3.5 can be used in implementing some control function.



Characteristics	Symbol	Min	Typ.	Max	Unit	Remarks
Supply voltage	V_{CC}	2.7	3	5.5	V	
	V_M	2.5	5	13.5	V	
Output current (H-SW)	I_{OUT}	—	—	1.0	A	$V_M \geq 4.5$ V
		—	—	0.4		4.5 V > $V_M \geq$ 2.5 V Without PWM Operation
Switching frequency	f_{PWM}	—	—	100	kHz	

Figure 2-4: Teensy and motor driver pins, data-sheet is from driver chip's website

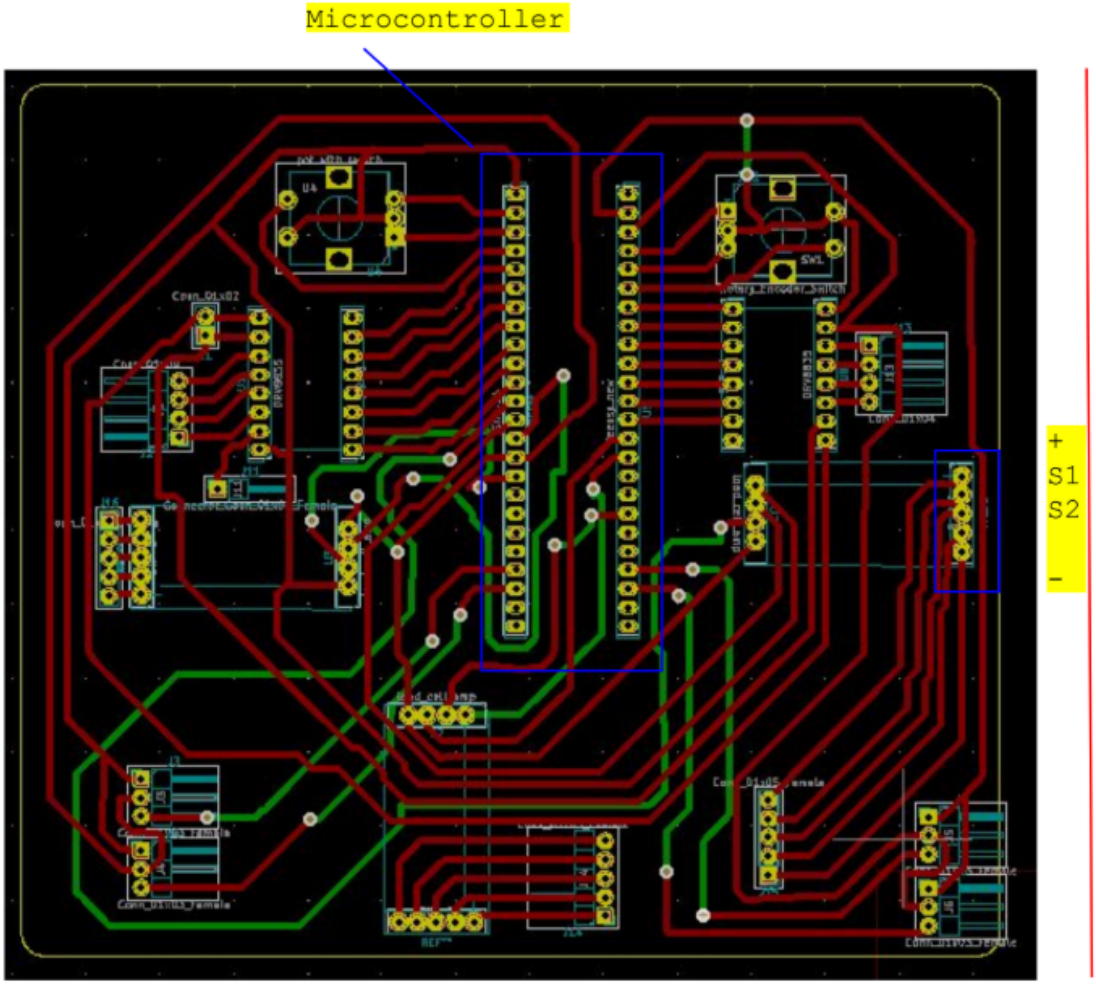


Figure 2-5: PCB design in KiCAD

2.3 Manufacturing and assembly

Figure 2-6 shows the built prototype of our tendon-driven joint which is intended to replicate the functionality of the human's shoulder joint. At the moment it is limited to 2 DOFs as long as we are using only 3 tendons. However, it can be extended to include 3 DOFs if we implement the YAW rotation using an additional motor whose shaft can pass through the center of the joint. Each of the three actuators winds the tendons and thread through holes of the platforms. And the next end of the tendons is fixed to the moving platform. The angle between each tendon is 120 degrees. There are included two rotary encoders for a real-time (with Teensy constraints) testing of the shoulder joint motion manually. These 2 rotary encoders send signals to the DRV8835 motor board to control the speed of the motors in forward and backward directions. One rotary encoder was initially to control motors 1 and 2. Using the switch integrated in the rotary encoder it is possible to stop the motor, reverse the direction and switch from motor 1 to motor 2. The second rotary encoder does the same thing but for motors numbers 3 and 4 (not used) in the prototype. There are 2 motor driver boards, 1 motor driver is for motor 1 and 2 and the second motor driver is for motor 3 and 4.

The prototype includes also 3 force sensors in order to measure the tension on each tendon. These force sensors F1, F2 and F3 are calibrated with a calibration factor of -70.0, -700.0, -1000.0 correspondingly and this is somehow equivalent to measuring the mass of real objects with some weight limit. These force sensors are working in parallel with the rotary encoders and motor drivers. There is a 1-2 seconds delay when the force sensor sends its value to the serial monitor, this is probably because there are 7 sensors and 2 motor drivers and 2 rotary encoders working at the same time and each of them writes data on the serial monitor. But this can be improved optimizing the code.

Also, 4 magnetic encoders sense the rotation of the actuators in clockwise and anti-clockwise directions and send them measurement to the Teensy as a PWM signal. These encoders count relatively precisely when not mapping their output, this means that these magnetic encoders work as non-calibrated more precisely. Their signal ranges from 0 to 1000 and it works well, showing the counts of the revolutions when the motor rotates. But, we mapped 0 to 1000 signal to be 0 to 360, so that we could associate with a real circular motion. However, at first we saw some jump in the measurement. However, this problem has been solved by removing the delay during reading the encoder. This, on the

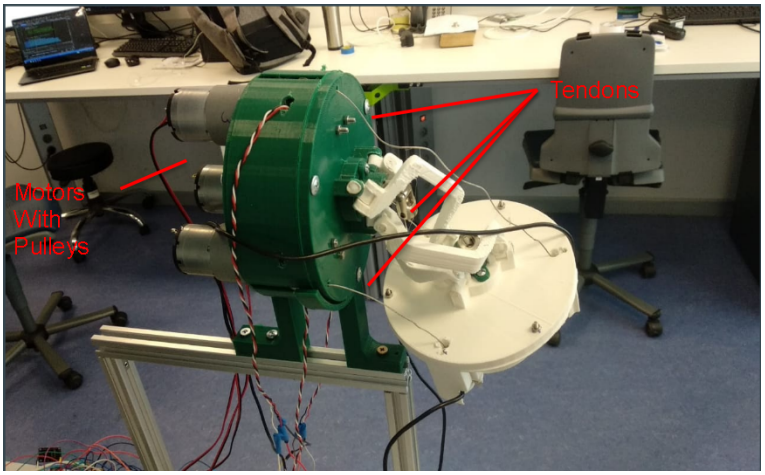


Figure 2-6: Proof of concept for a 3-DOF shoulder joint (only two actuated) realized in PLA.

other hand, works very smoothly when we do not map the magnetic encoder signal and leave it to be 0 to 1000. Also, these magnetic encoders work simultaneously with 3 force sensors and 2 motor drivers, and 2 rotary encoders. This all happens because of Teensy's all pin interrupts capabilities.

When 3 tendons work in parallel, in order to have motion, the force that is applied to the actuated tendon should be opposite compared to the other two tendons that have to loosen/tighten. (At this point the tendon simulation is not scripted, but controlled by Newton's laws). In the physical simulation, by using the Python program, each button was programmed to make the 3 motors work at the same time. To make the best movement, it was decided to make all tendons tightened at first, and coordinate them according to the motion we wanted to realize. While the joint is moving it is necessary to avoid the situations whee tendons exceed their force limits or loosen too much.

In Figure 2-8 the motherboard is represented. It includes the micro-controller, the motor boars and all the wires necessary to connect the force sensors and the position encoders. Up to now the interface works as expected and can control the shoulder joint. The PCB plate contacts have been tested with a multi-meter to detect short-circuits or discontinuities. The motor boards and the Teensy 3.6 were mounted using male & female connectors in a way to easily substitute them if required. In particular, the female connectors where soldered on the mother-board while the male connectors on the Teensy 3.6 and the motor boards. After seeing that it is working as expected, each component was soldered one after another and the correct integration with the Teensy 3.6 verified.

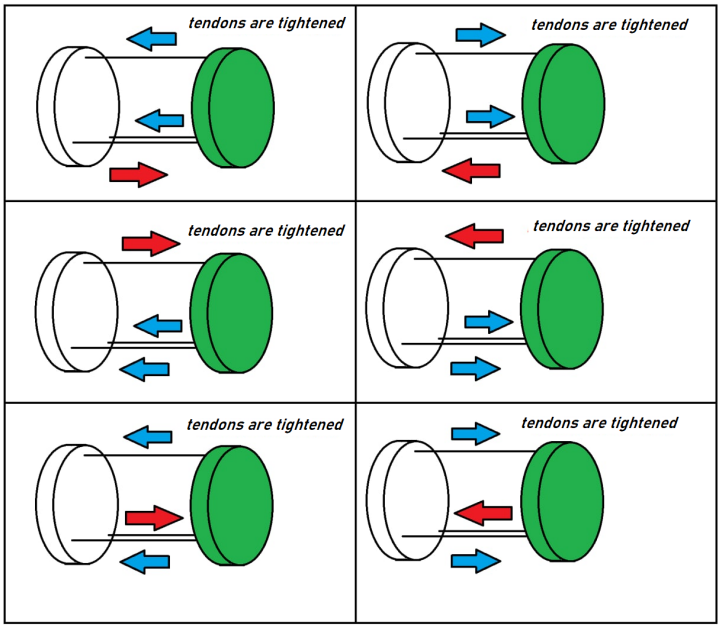


Figure 2-7: Concept explanation [4]

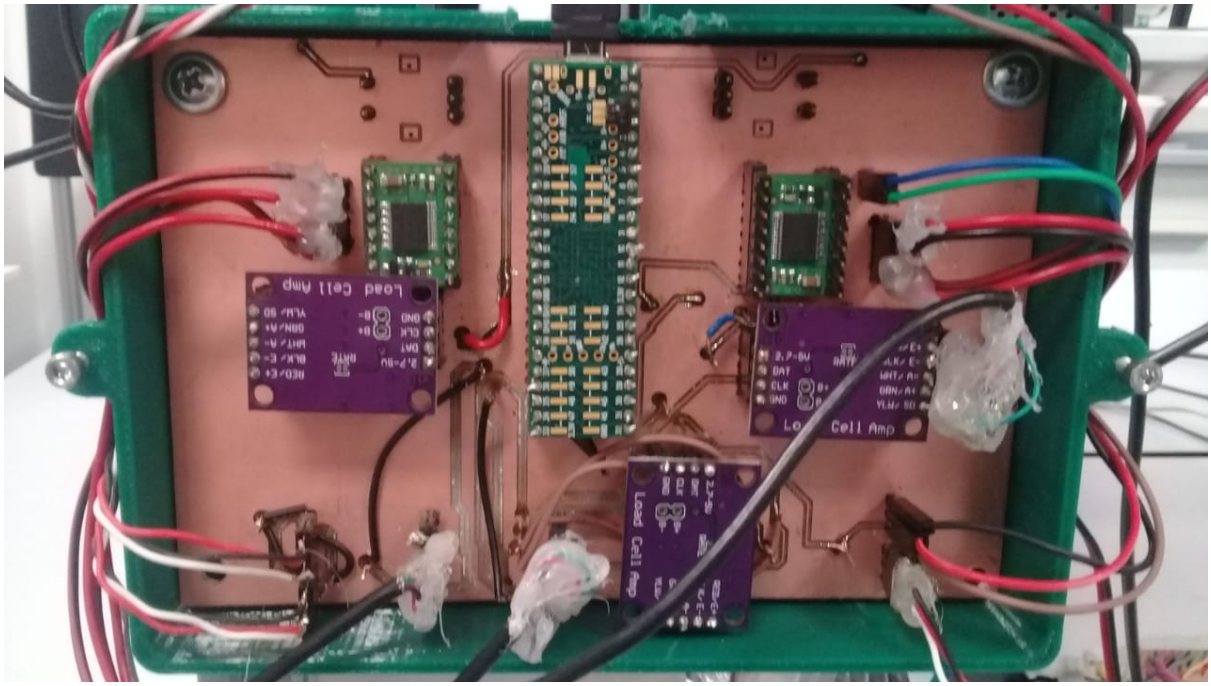


Figure 2-8: Motherboard which incorporates a Teensy 3.5, force sensors amplifiers, motor controllers, and rotary encoders.

2.4 Software part and GUI to control the joint

The motors can be stopped at any point as long as we are using threads.

When running the Python interface code, see figure 2-6, an updated interface can be seen. When the gauge is rotated and the actuator is chosen; for example the actuator 1, then if we click the button the PWM duty cycle value is send to the microcontroller. The tendon which is attached to actuator 1 gets loosened or tightened depending on which direction the gauge is rotated. This logic is the same for actuator 2 and actuator 3. Also, the indicator lamps show the color green if the corresponding actuator is actuated. The interface can also read the force sensor values instead of reading the position encoder values. The position encoder values can be read by default. The actuators can be turned off in case of an emergency using one button, so all the actuators shut down immediately. Also, the table below shows the values of position encoders or force sensors. The dynamic graph represents the position encoder or force sensor values while they are changing dynamically depending on time. The table values also correspond to this time. The 2nd or 3rd actuators can be activated at the same time, but it is recommended to be careful with the mechanical limit of the shoulder so that the tendons' or PLA components' lives do not get shortened.

The dial in the interface is used to represent the negative and positive values from -100 to 100. The button "send the value" is used to send the values in the dial. "Turn off the actuators" is used to turn off all the actuators. "Actuator 1" actuates actuator 1 and deactivates it back. "Actuator 2" actuates the actuator 2 and deactivates back and so on. And the red indicator is turned on when one of the actuators is activated. The graph for the encoder represents the corresponding position values received from each of the magnetic encoders that are installed over the head of the actuators such that the encoder can increase or decrease its values depending on the direction of the motor rotation.

Figure 2-12 depicts how commands are encoded to be sent to the micro-controller from the Python graphical interface. The serial port is used as a connection between the Python interface and the Teensy 3.6 micro-controller. Integers are converted to bytes and send to the micro-controller to activate functions, they allow reading and writing data from and to the micro-controller.

For example to run motor 3 forward with 0.5V:

Figure 2-9: Concept explanation: interface

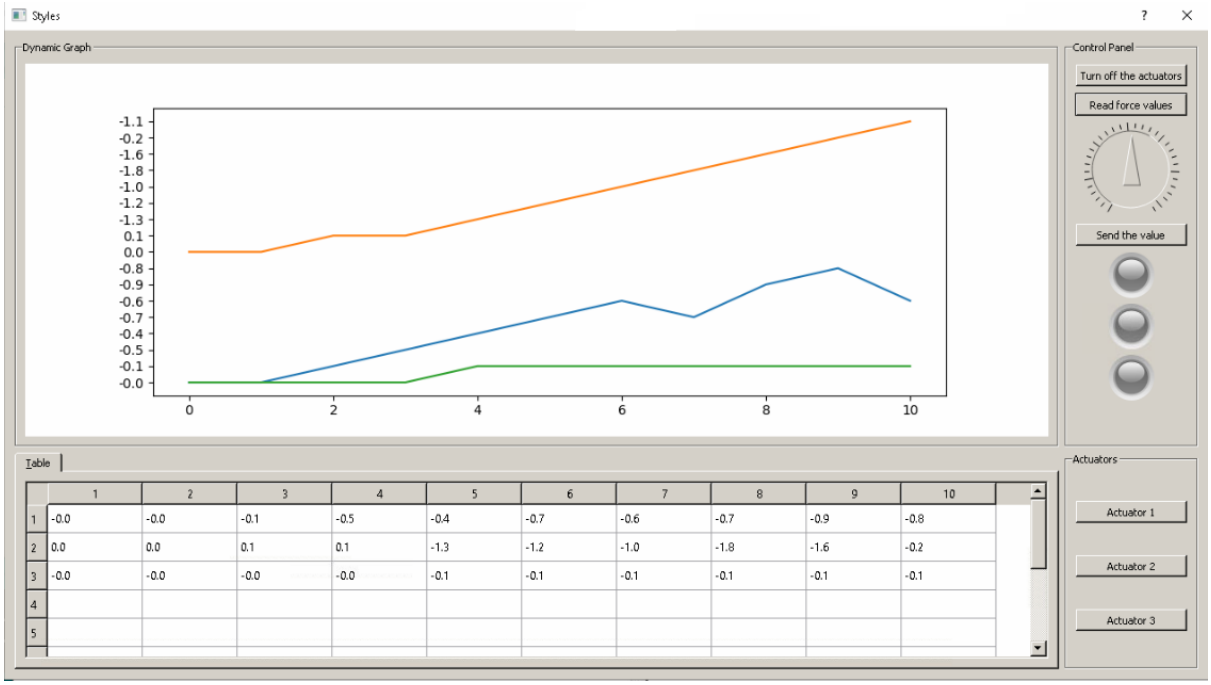


Figure 2-10: Example of an interface to manage the spheroid joint (the graph shows the force sensor measurements)

0:0|1:0|2:0|3:50|4:0|5:0|6:0|7:0|8:0|9:253|

3-motor stop:

0:0|1:0|2:0|3:-1|4:0|5:0|6:0|7:0|8:0|9:253|

stop all:

0:-1|1:0|2:0|3:0|4:0|5:0|6:0|7:0|8:0|9:0|

We had to rewrite most of our code since the program was not working as expected, specifically it could not process one byte at a time to send both force sensor and position encoder data from the Python interface to the micro-controller and back. And it needed some delay to start reading sensor data, also there was a delay to send the last position data to Arduino from the file. In this case, the measurements were not accurate, motors started to work before reading data. To make everything working at the same time, we had to send commands in one line, and process it first, only then start everything at loop without delay. Also, the program was optimized to have less code. It was not suitable to use string library in Arduino IDE, so we used C implementation using char array.

The main of Program:

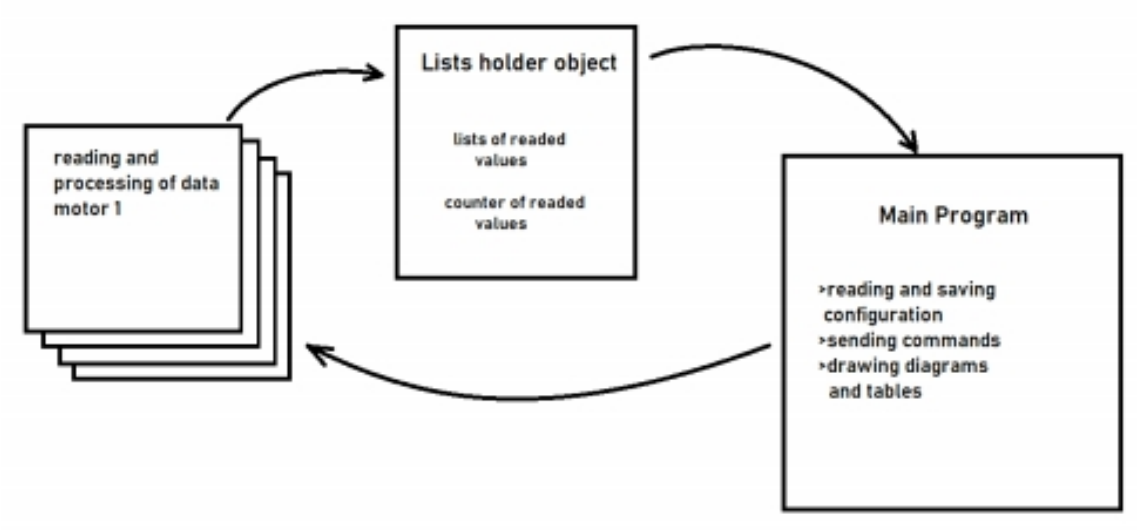


Figure 2-11: The main of program

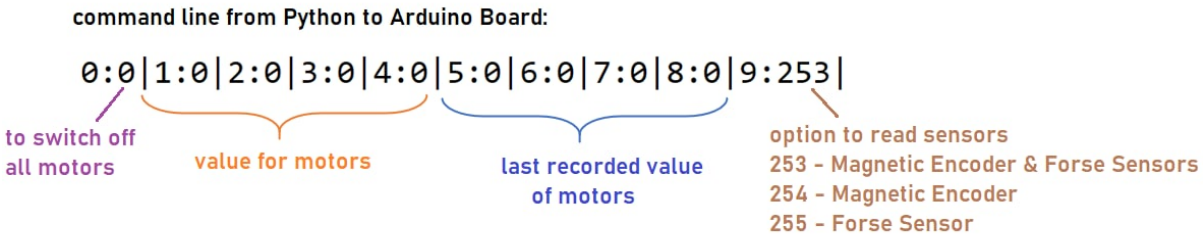


Figure 2-12: Command codes from Python to Arduino

When working on program of sending commands to Microcontroller and reading real-time data from it, two main obstacles were encountered. Commands, at first, were sent byte by byte, the internal loop in Microcontroller read each byte and processed it with some programmed delay. And then send sensor data also with some delay. Even if delay was very small, bytes that were send as a command information and bytes that were read and processed not only suffered from lag but also was prone to interfere with each other. Better decision was to organize one command for controlling motors and fetching specific data. Below is an example, value for motors are power from 0 to 120 (255 is maximum, but only 1 motor can run with such power. Since each motor driver controller 2 motors, we could potentially operate 4 motors in parallel sending signal values less than 127).

Example to run the second motor: 0:0|1:0|2:70|3:0|4:0|5:0|6:514|7:0|8:0|9:255| the second motor PWM (Pulse Width Modulation) value is 70, last saved position is 514 degrees (which was set to maximum, it means motor previously rotated to its maximum to the right), 255 is command to read both Magnetic Encoder and Force Sensor at the same

```
def createThreads(self):
    self.threads = []
    self.workers = []
    self.threads.append(0)
    self.workers.append(0)
    for i in range(1,5):
        self.threads.append(QThread())
        self.workers.append(fetchData())
        self.workers[i].moveToThread(self.threads[i])
        self.threads[i].started.connect(self.workers[i].run)
        self.workers[i].finished.connect(lambda: self.stop_thread(i))
        #self.workers[i].finished.connect(self.workers[i].deleteLater)
        #self.threads[i].finished.connect(self.threads[i].deleteLater)
        #self.workers[i].progress.connect(self.drawEncFrs)
        self.workers[i].finished.connect(self.drawEncFrs)
```

Figure 2-13: Using threads as interrupt method

```
def stop_thread(self, opt):
    self.workers[opt].stop()
    self.threads[opt].quit()
    self.threads[opt].wait()
```

Figure 2-14: Stop thread

time. Such a solution allowed the micro-controller to parse the input and then start running the motor and reading the sensor data almost at the same time, e.g., without delay. Here is an example to stop all motors: 0:-1|1:0|2:0|3:0|4:0|5:0|6:0|7:0|8:0|9:0| To stop only the second motor: 0:0|1:0|2:-1|3:0|4:0|5:0|6:0|7:0|8:0|9:0| The second problem was that when reading from Serial port real-time data, the Main Program was busy, and GUI was not responding. So that decision was made to create 4 threads to each motor. Maybe it was to some extent more than necessary cause there is only one Com Port for reading and sending data, but it made each process less dependent on each other. Here is an example of creating 4 threads when the Main Program is initialized, also notice that worker and thread objects of PyQt5 are not deleted to be able to start process again if we send a new command.

Also, before sending command to stop a specific motor or to stop all running motors, we must be sure that the Com Port is not busy, e.g., the respective thread of fetching data from the sensors is properly stopped.

And while pressing the button:

The main program is so organized, that pressing GUI buttons and accessing GUI Dial object, participate in creating command line self.cmd.

```
if (self.threads[act_motor].isRunning()):
    self.workers[act_motor].stop()
    self.threads[act_motor].quit()
    self.threads[act_motor].wait()
```

Figure 2-15: While pressing button

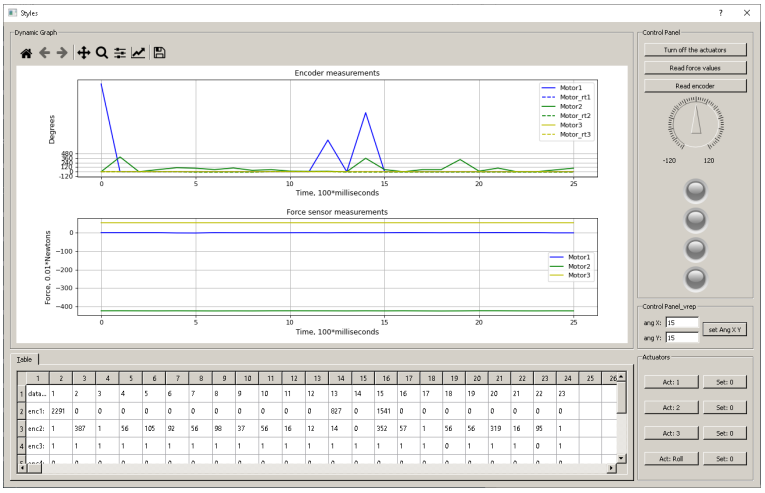


Figure 2-16: We have added set AngX and AngY to GUI to control

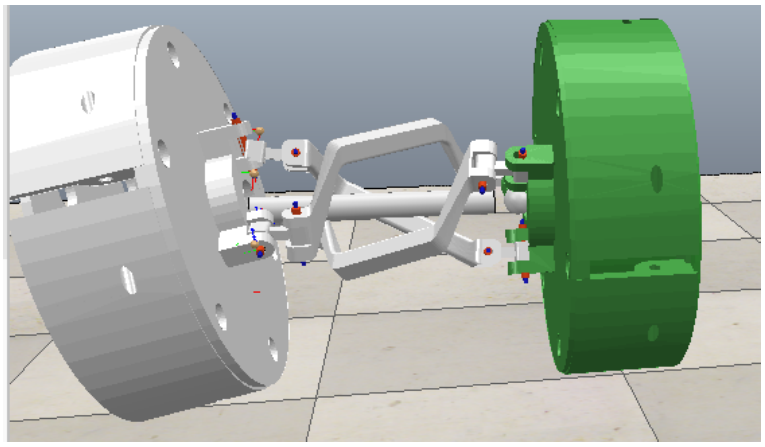


Figure 2-17: V-REP model updated, set angX=15 and angY=15

In addition other fields were added to the GUI, thus AngX and AngY, to control also the V-REP model.

For example, in this case, we set angX=15 and angY=15 we can see the corresponding rotation:

These are the first and second page of the GUI and it is working as expected:

2.5 Force and position control with updated design

The following is the tendon length and position or angle relationship 2-20.

Right now the motor rotates from -514 to 514 degrees. It takes approximately 3 rotations. There was one challenge, when we were reading encoder and force data, we could not send a command to stop the motor. It was solved by reading input every 7 iterations from the micro-controller and making 4 separate processes (threads) in Python

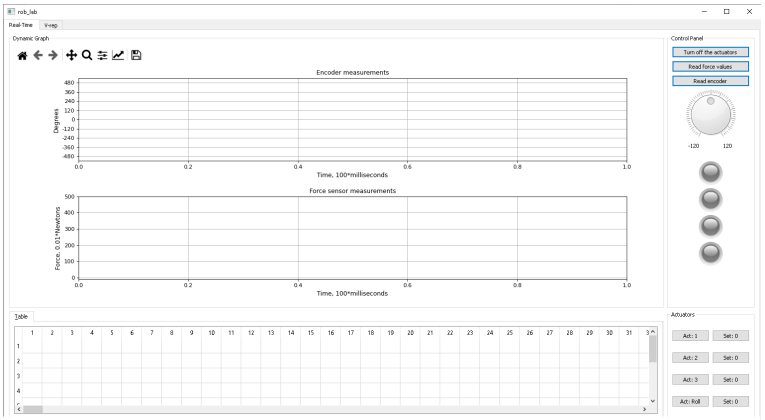


Figure 2-18: GUI, tab1, shoulder joint control

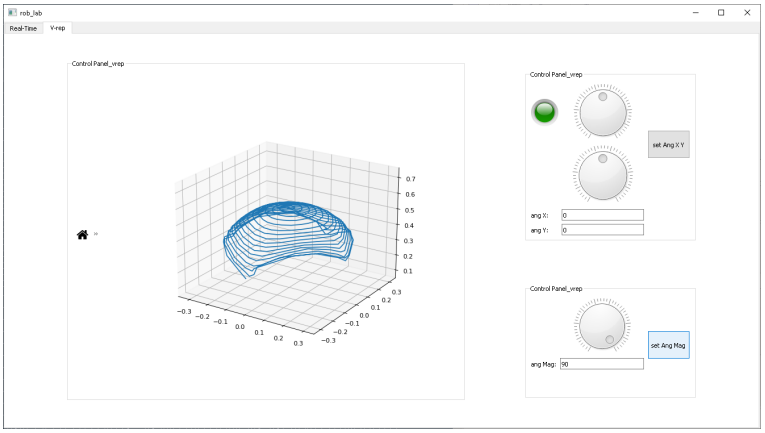


Figure 2-19: GUI, tab2, CoppeliaSim control

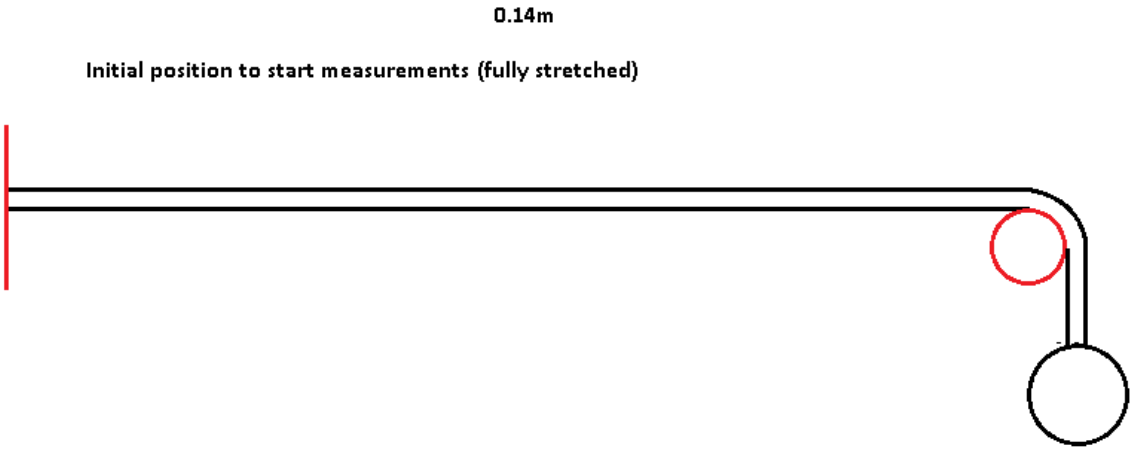


Figure 2-20: Position or angle relationship

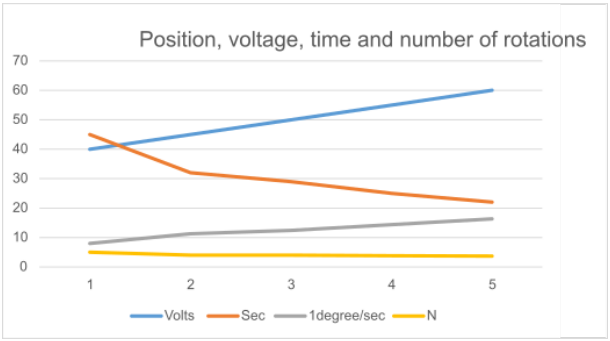


Figure 2-21: Position, voltage, time and number of rotations

for each motor to fetch data. Also the button "set 0" was added. When the motor is installed in default position, this button refreshes configuration file to pos:0|0|0|0|.

The figure below 2-22 shows the position and force control scheme.

As you could see from figures 4-10 and 4-11 it is starting from the saved value in the control text file. The encoder decrements according to the direction and then increments from that point. It can also be calibrated now.

Table 2.2: Position voltage-time and number of rotations

volts (approx)	sec	1 degree/sec	N
0.40	45	8	5
0.45	32	11,25	4
0.50	29	12,4137931	4,027778
0.55	25	14,4	3,819444
0.60	22	16,36363636	3,666667

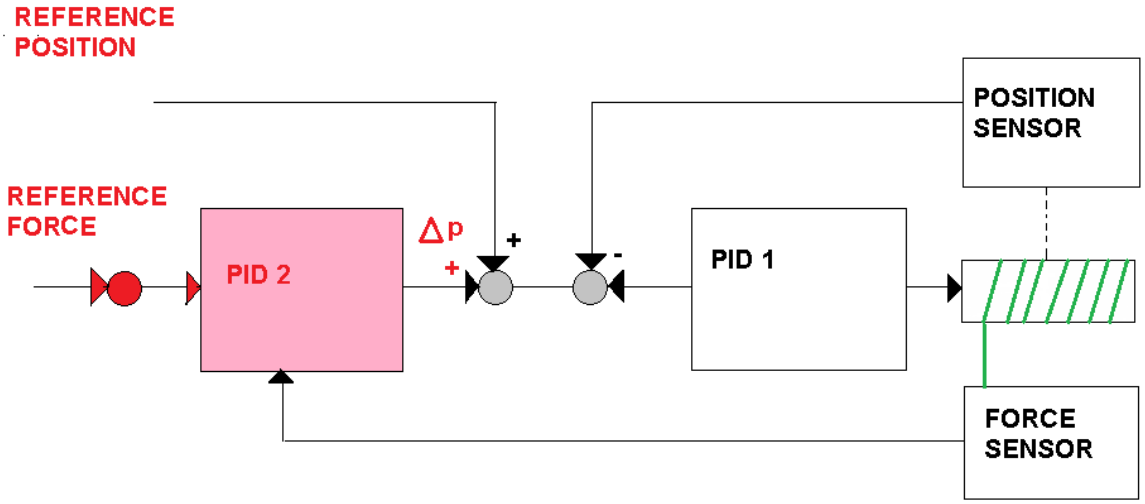


Figure 2-22: Force and position control

change.log | Qt5_Example.py | remoteApi.dll | run_motor3_backup.ino | ctrl.cfg

```
1  pos:|0|0|0|553|
2  enc:|0|0|0|0|0|0|0|1
3  angle passed(degrees):|0|0|0|553|
4  pure distance passed(m):|0.0000|0.0000|0.0000|0.0888|
5
```

Figure 2-23: Position is saved on a text file

Chapter 3

Articulatio spheroidea model and simulation

3.1 Detailed design of three degrees of freedom manipulator

The kinematic architecture of the shoulder joint is presented in figure 3-1. It consists in three legs that connect the fixed platform to the moving platform. Each leg has a RRRR kinematics, thus a sequence of four revolute joints.

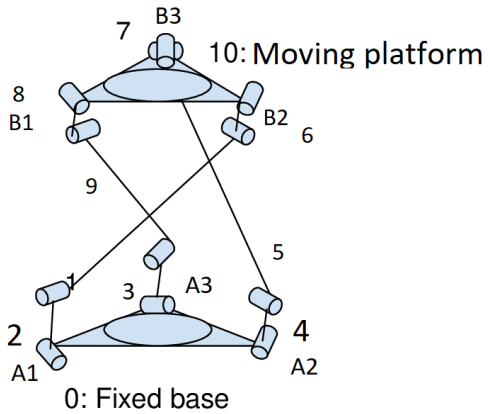


Figure 3-1: Architecture of parallel kinematic mechanism with rotational joints.

If we indicate L as the numbers of independent loops in the closed kinematic chain, n=11 the number of links including the fixed and moving base, and j=12 the number of binary joints. Using Euler equation:

$$L = j - n + 1 \tag{3.1}$$

$$L = 12 - 11 + 1 = 2 \tag{3.2}$$

And for any parallel manipulator the number of limb:

$$m = L + 1 = 2 + 1 = 3 \tag{3.3}$$

In this special manipulator the DOFs are three and therefore they differ from the number of limbs. Therefore Glübler criterion does not apply.

3.2 Kinematics for the primary limb

For this parallel manipulator, w_c=47.44mm, l_c=109.25mm, h_0=128.32mm. If we consider

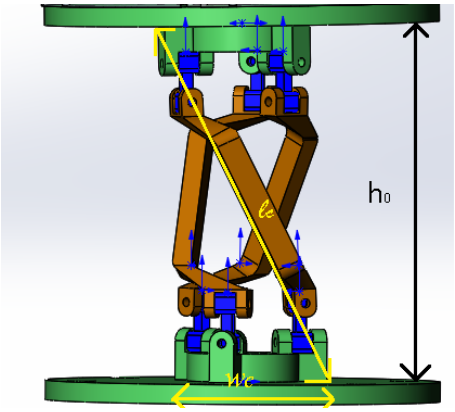


Figure 3-2: h_0 is the distance between the platforms, l_c is the length of the limb,w_c is the distance between the limbs

each leg as an open kinematic chain we can use the screw method to formalize the forward model. Starting from a reference frame we can define the four screws as the table 3.2.

s_i	s_{oi}
(1, 0, 0)	$(a_0, 0, 0)$
(0, 1, 0)	$(a_0, 0, a_1)$
(0, 1, 0)	$(a_0, 0, a_1 + a_2)$
(1, 0, 0)	$(a_0, 0, 2a_1 + a_2)$

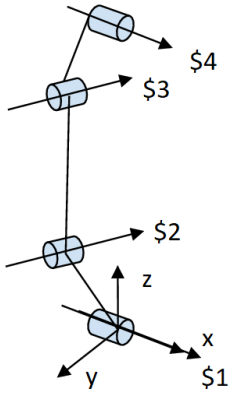


Figure 3-3: Screw method

Note, here we use a notation: $s\theta = s$ and $c\theta = c$. The transformation matrices associated with each of the four screws are, A_1, A_2,A_3, and A_4:

$$A_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_1 & -s_1 & 0 \\ 0 & s_1 & c_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

$$A_2 = \begin{bmatrix} c_2 & 0 & s_2 & a_0(s_2-1) - a_1 s_2 \\ 0 & 1 & 0 & 0 \\ -s_2 & 0 & c_1 & -a_0 s_3 + (a_1+a_2)(s_2-1) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

$$A_3 = \begin{bmatrix} c_3 & 0 & s_3 & a_0(s_3-1) - (a_1+a_2)s_3 \\ 0 & 1 & 0 & 0 \\ -s_3 & 0 & c_3 & a_0 s_3 - (a_1+a_2)(s_3-1) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

$$A_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_4 & -s_4 & -(2a_1+a_2)s_4 \\ 0 & s_4 & c_4 & (2a_1+a_2)(1-s_4) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

$A_{14} = A_1 A_2 A_3 A_4$ gives us the transformation matrix associated with the compound

screw..

$$A_{14} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & p_x \\ a_{21} & a_{22} & a_{23} & p_y \\ a_{31} & a_{32} & a_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

$$p_x = s_2(a_0 s_3 - (a_1 + a_2)(s_3 - 1)) - s_2(a_1 s_2 - a_0(s_3 - 1)) + a_0(s_3 - 1) - s_{2-3}(s_4 - 1)(2a_1 + a_2) \tag{3.9}$$

$$p_y = s_1 s_2(a_1 s_2 - a_0(s_3 - 1)) - s_1(a_1 + a_2 - a_0 s_3) - s_1 s_4(2a_1 + a_2) - (s_4 - 1)(s_1 s_2 s_3 -$$

$$s_2 s_3 s_1)(2a_1 + a_2) - s_2 s_1(a_0 s_3 - (a_1 + a_2)(s_3 - 1)) \tag{3.10}$$

$$p_z = s_1(a_1 + a_2 - a_0 s_3) - s_1 s_2(a_1 s_2 - a_0(s_3 - 1)) + (s_4 - 1)(s_1 s_2 s_3 - s_1 s_2 s_3)(2a_1 + a_2)$$

$$-s_4 s_1(2a_1 + a_2) + s_1 s_2(a_0 s_3 - (a_1 + a_2)(s_3 - 1)) \tag{3.11}$$

3.3 Single Limb Inverse Kinematics

The inverse kinematics for one limb of the shoulder joint is the same as for the other limbs. Applying the DH method, thus multiplying left and right side of the matrix equation by the inverse of the elementary transformations matrices we can solve for the four joints angles.

$$p = A_1 A_2 A_3 A_4 p_0 \tag{3.12}$$

$$A_2^{-1} A_1^{-1} p = A_3 A_4 p_0 \tag{3.13}$$

$$p_0 = \begin{bmatrix} a_0 \\ 0 \\ 2a_1 + a_2 \\ 1 \end{bmatrix} \tag{3.14}$$

$$p = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \tag{3.15}$$

$$u_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{3.16}$$

$$u = R_1 R_2 R_3 R_4 u_0 \tag{3.14}$$

$$u = \begin{bmatrix} c_{2-3} \\ -s_{23} s_1 \\ s_{23} s_1 \end{bmatrix} \tag{3.15}$$

$$u_y^2 + u_z^2 = s_{23}^2 \tag{3.16}$$

$$\theta_{23} = \pm\sqrt{u_y^2 + u_z^2} \tag{3.17}$$

$$\frac{u_y}{u_z} = -tan\theta_1 \tag{3.18}$$

$$\theta_1 = -atan2(\frac{u_y}{u_z}) \tag{3.19}$$

By equating

$$A_2^{-1}A_1^{-1}p \tag{3.20}$$

and

$$A_3A_4 \tag{3.21}$$

using MATLAB results:

$$\theta_{4,1} = -\frac{\pi}{4} - \arcsin(\frac{\sqrt{\frac{1}{2}}(p_y s_1 + p_z s_1)}{2(2a_1 + a_2)}) \tag{3.22}$$

using MATLAB results and

$$\theta_{4,2} = \frac{3\pi}{4} + \arcsin(\frac{\sqrt{\frac{1}{2}}(p_y s_1 + p_z s_1)}{2(2a_1 + a_2)}) \tag{3.22}$$

$$u_x = c_{2-3} \tag{3.23}$$

$$s_{2-3} = \pm\sqrt{1 - u_x^2} \tag{3.24}$$

$$\theta_{2-3} = atan2(\pm\sqrt{1 - u_x^2}, u_x) \tag{3.25}$$

$$\theta_{2-3} + \theta_{2+3} = atan2(\pm\sqrt{1-u_x^2}, u_x) \pm \sqrt{u_y^2 + u_z^2} \tag{3.26}$$

$$2\theta_2 = atan2(\pm\sqrt{1-u_x^2}, u_x) \pm \sqrt{u_y^2 + u_z^2} \tag{3.27}$$

$$\theta_{2,1} = \frac{atan2(\pm\sqrt{1-u_x^2}, u_x) + \sqrt{u_y^2 + u_z^2}}{2} \tag{3.28}$$

$$\theta_{2,2} = \frac{atan2(\pm\sqrt{1-u_x^2}, u_x) - \sqrt{u_y^2 + u_z^2}}{2} \tag{3.29}$$

$$\theta_{2-3} - \theta_{2+3} = atan2(\pm\sqrt{1-u_x^2}, u_x) \pm \sqrt{u_y^2 + u_z^2} \tag{3.30}$$

$$-2\theta_3 = atan2(\pm\sqrt{1-u_x^2}, u_x) \pm \sqrt{u_y^2 + u_z^2} \tag{3.31}$$

$$\theta_3 = \frac{atan2(\pm\sqrt{1-u_x^2}, u_x) \pm \sqrt{u_y^2 + u_z^2}}{-2} \tag{3.32}$$

$$\theta_{3,1} = \frac{atan2(\pm\sqrt{1-u_x^2}, u_x) + \sqrt{u_y^2 + u_z^2}}{-2} \tag{3.33}$$

$$\theta_{3,2} = \frac{atan2(\pm\sqrt{1-u_x^2}, u_x) - \sqrt{u_y^2 + u_z^2}}{-2} \tag{3.34}$$

3.4 MATLAB simulation of PM

We have simulated one limb of the parallel manipulator in MatLab as it can be seen in the figures 3-4 and 3-5 . When one leg is moved the other two legs will follow. By changing the angle of the first two joints in the first leg it is possible to change the position and orientation of the end-effector, thus the parallel structure has only two DOFs.

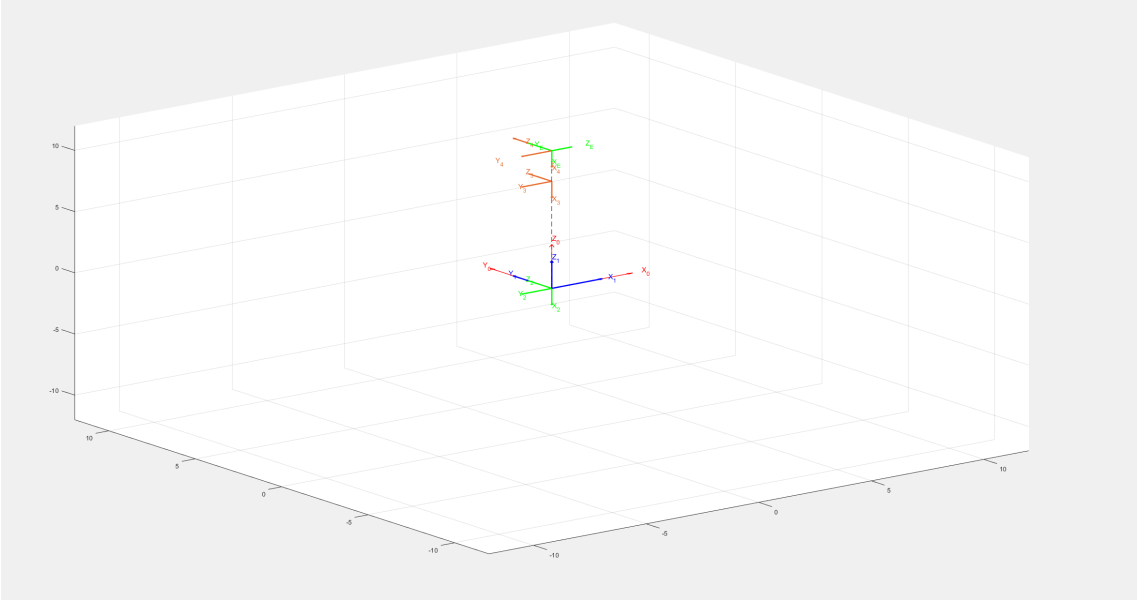


Figure 3-4: MATLAB simulation of PM, the first leg is bent $\frac{\pi}{4}$ (θ angle) and therefore the moving platform is bent $\frac{\pi}{4}$ (ϕ angle)

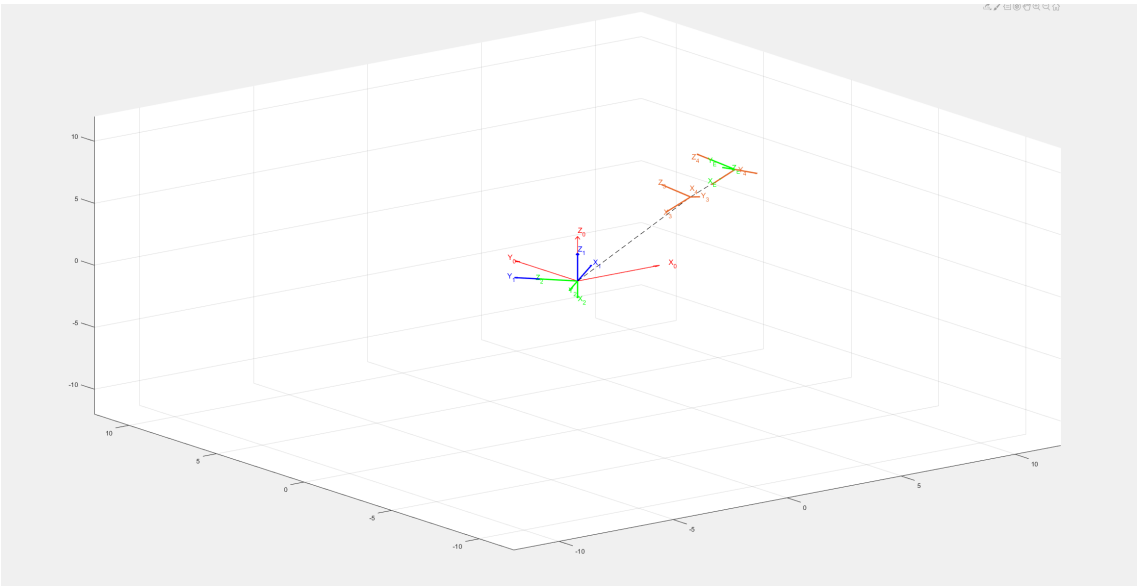


Figure 3-5: MATLAB simulation of PM, the first leg is in the base position

3.5 Inverse Kinematics Planar Case - Geometric Method

Formulation of dependency

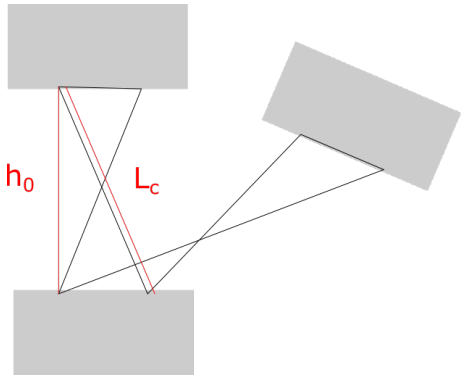


Figure 3-6: Formulation of dependency

$$tg(\psi)y_c = x_c - h_0 * tg(\psi) \tag{3.35}$$

$$tg(\psi) = \frac{x_c}{h_0 + y_c} \tag{3.36}$$

$$r = x_c * sin(\psi) \tag{3.37}$$

$$sin(\frac{\theta}{2}) = \frac{k}{\frac{w}{2}} \tag{3.38}$$

$$k = \frac{w}{2} * sin(\frac{\theta}{2}) \tag{3.39}$$

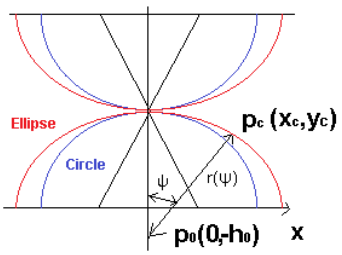


Figure 3-7: Geometry of movement

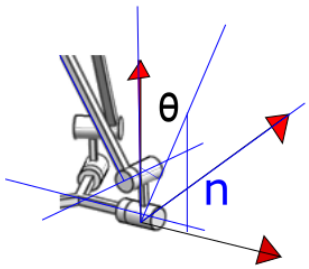


Figure 3-8: Geometrical solution attempt for the one limb adapted from [7]

$$z_0 = x_a = f + s_2 * m + s_3 n - f \tag{3.40}$$

$$x_0 = y_a = s_1 * n + s_4 * k \tag{3.41}$$

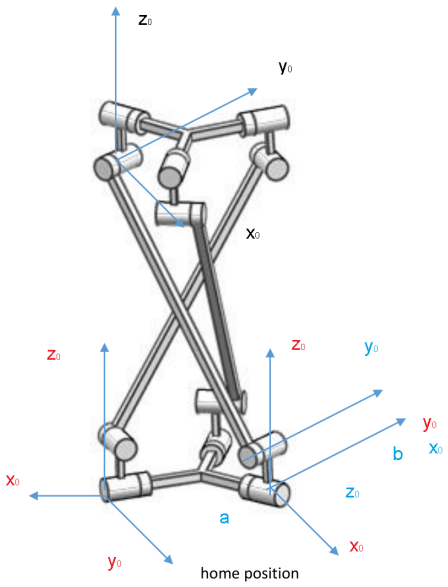


Figure 3-9: Geometrical solution attempt for the one limb adapted from [7]

$$y_0 = z_a = c_4 * n + c_2 * m + s_3 * n + c_4 * k \tag{3.42}$$

3.6 Inverse Kinematics of the Shoulder Joint

Figure 3-10 shows the kinematic model of the shoulder joint that is made up of a moving platform, fixed base, and a central limb that allows a rolling motion. The moving platform is considered as spatially oriented, thus, it will be analyzed as three extensible limbs connected to the moving platform at points B_i and the fixed base at points A_i. Pulleys across the platform can be used to vary the lengths L of the robe to control the motion of the moving platform.

For analysis, two Cartesian coordinate systems O(x_0, y_0, z_0) and E(x_1, y_1, z_1) are attached to the fixed base and moving platform respectively. The origin of the platform is chosen to be at the center of an additional limb. The transformation from the moving frame E to the fixed frame O is described by a 4x4 transformation matrix ${}^{O}T_E$. The initial orientation of the moving frame E coincides with the fixed frame O and the final displacement is obtained by performing a rotation of an angle ϕ about the z-axis, followed by a second rotation of $\frac{\theta}{2}$ about the y-axis, followed by a translation of a quantity h along the z-axis, a rotation of $\frac{\theta}{2}$ about the y-axis and finally a rotation of an angle $-\phi$ about

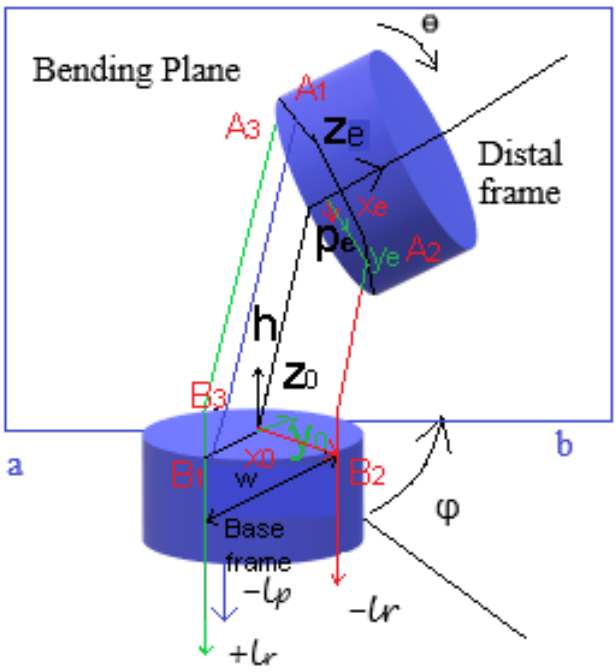


Figure 3-10: Geometric representation of our shoulder joint, adapted from [7]

the z-axis. The resulting transformation matrix is given by equation 3.43, [7].

$$
{}^{0}T_E = R_z(\phi)R_y(\frac{\theta}{2})T_z(h)R_y(\frac{\theta}{2})R_z(-\phi) =
$$

$$
\begin{bmatrix}
1-2C^2(\phi)S^2(\frac{\theta}{2})) & -2S(2\phi)S^2(\frac{\theta}{2}) & C(\phi)S(\theta) & hC(\phi)S(\frac{\theta}{2}) \\
-2S(2\phi)S^2(\frac{\theta}{2}) & 1-2S^2(\phi)S^2(\frac{\theta}{2}) & S(\phi)S(\theta) & hS(\phi)S(\frac{\theta}{2}) \\
-C(\phi)S(\theta) & -S(\phi)S(\theta) & C(\theta) & hC(\theta) \\
0 & 0 & 0 & 1
\end{bmatrix}
\tag{3.43}
$$

If we assume that the position vector a of point B_i is given by

$$
{}^{E}B_i = \begin{bmatrix} b_{i_x}, & b_{i_y}, & b_{i_z}, & 1 \end{bmatrix}^T \tag{3.44}
$$

Then the position of the moved vector with respect to the fixed frame O is

$$
{}^{\mathbf{O}}\mathbf{B_i} = {}^{O}T_E * {}^{E}B_i \tag{3.45}
$$

Knowing the extremities of the tendons as the point vectors ${}^{\mathbf{O}}\mathbf{B_i}$ and ${}^{\mathbf{O}}\mathbf{A_i}$ it is possible to calculate the length of each tendon i as $||{}^{\mathbf{O}}\mathbf{B_i} - {}^{\mathbf{O}}\mathbf{A_i}||$.

3.7 3D design components of PM

The 3D design of the parallel structure is shown in figure 3-11. The numbers in the figure represent different parts of that manipulator: 1 - fixed base, 2 - base and limb connection, 3 - rotational joint, 4 - link, 5 - rotational joint, 6 - union, 7 - platform and limb connection, 8 - moving platform.

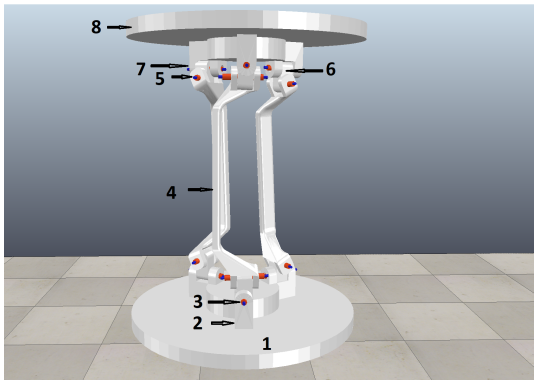


Figure 3-11: Low inertia high-stiffness parallel manipulator

Three tendons are displaced around the parallel structure at 120 degrees. They apply forces to the moving platform and therefore the torques T1, T2, and T3 as reported in the V-REP model. T3 is known and and T1 are calculated and T2 are calculated. T3 = 10 -> Target torque 3.

Figure 3-12 shows the shoulder joint simulated model (in V-REP, older version)

The figure 3-13 shows how the shoulder joint is bent in a V-REP simulation environment. a) model is controlled with x and y angles, the second b) with tendons. **Torque and Speed formula**

$$Torque = \frac{Power}{speed} \tag{3.47}$$

or

$$\tau = \frac{P}{w} \tag{3.48}$$

Where P is the power of the actuator,

τ is the torque,

w is the angular speed or velocity.

Figure 3-14 represents show a first realization of the shoulder joint.

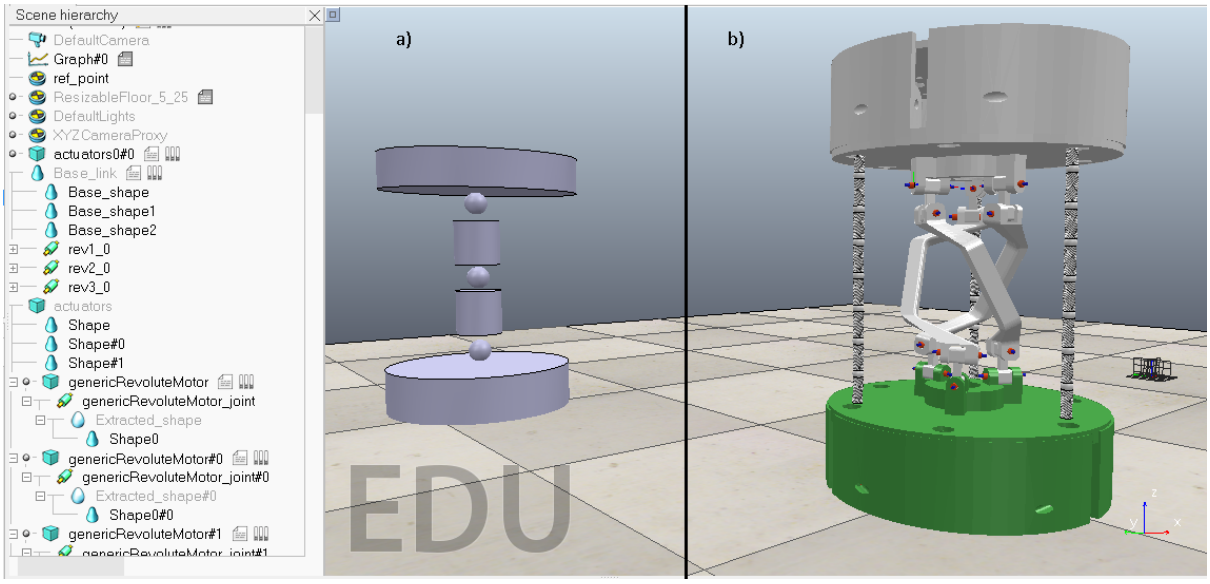


Figure 3-12: V-REP simulation, a) the model which is controlled with x and y angles b) shoulder joint

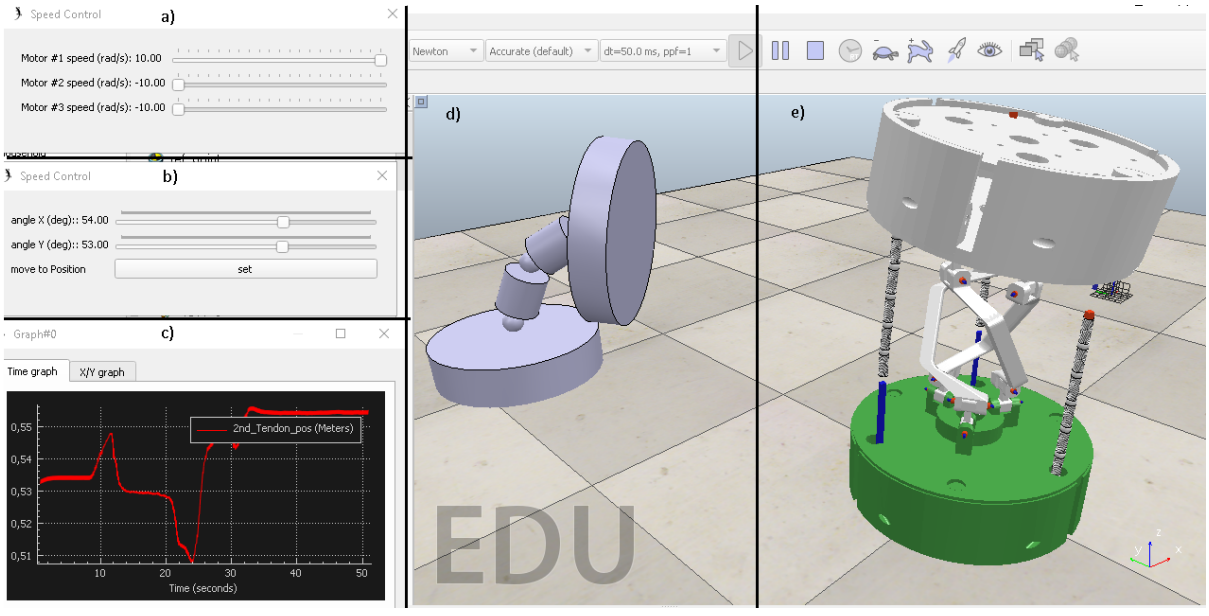


Figure 3-13: V-REP simulation, a) speed control of the tendon actuated shoulder joint simulation b) speed control of the model which is controlled with x and y angles c) graph which is obtained from shoulder joint simulation d) the model which is controlled with x and y angles is bent to a target location e) simulation of the tendon actuated shoulder joint where it is bent to a target location

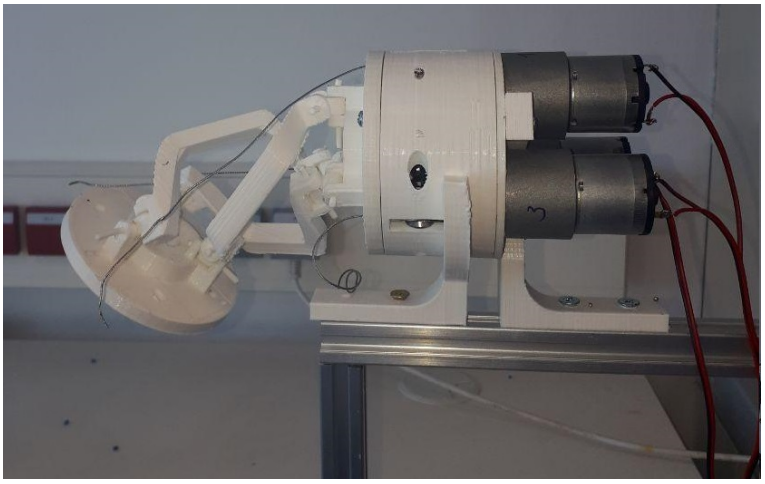


Figure 3-14: Shoulder joint platform older design version

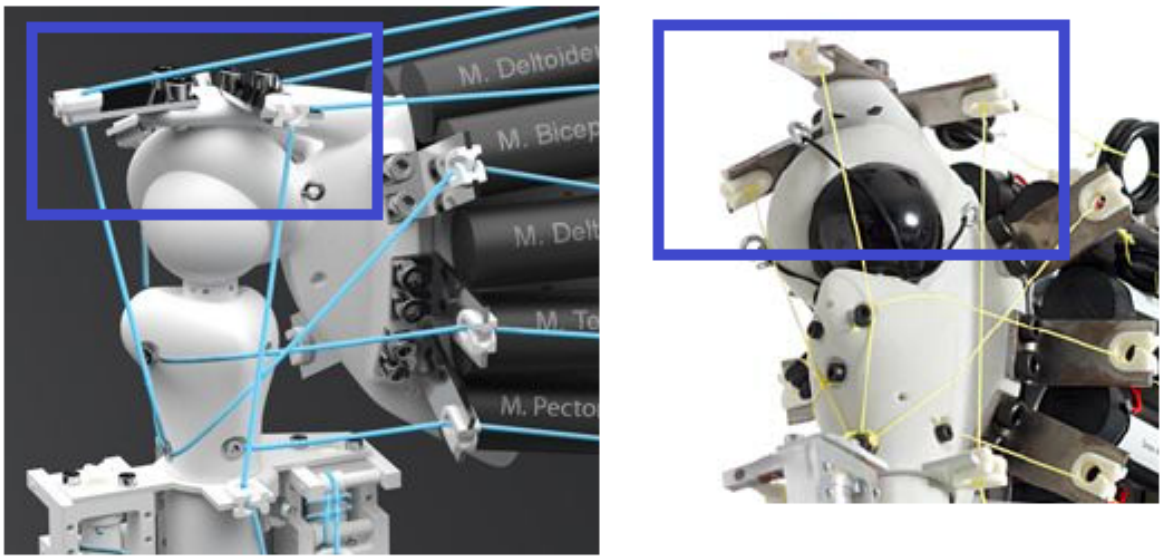


Figure 3-15: Example of tendon actuated humanoid shoulder joint. Tendons are in vertical orientation [4]

When in vertical orientation our shoulder model's tendons look like the mechanism in figure 3-15. As advantage, in the design we implemented, the links introduce additional constrains and the shoulder keeps assembled also in absence of tension in the tendons.

The figure 3-16 shows an approach to imitating the spherical joint. Due to the difficulty of replicating the tendon in V-REP, we have built a V-REP model which is a simulation of the real hardware. However, the workspace shows that the imitation model differs from the real hardware.

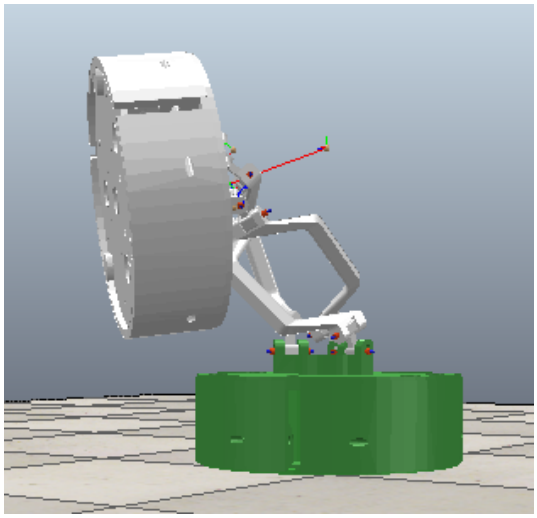


Figure 3-16: A V-REP model which imitates the spherical joint bent to 90 degrees

3.8 EEG Control System Setup

One of the goals of this thesis was to evaluate the possibility to control the shoulder using EEG signals. This may be useful when performing rehabilitation therapies or to control a prosthesis device. The preparation of a model of the shoulder is crucial to conduct preliminary experiments without involving the real robot prototype. In the beginning, will be possible to work in a simulation environment where EEG signal are integrated and used to control the shoulder model. After when the control system is tuned will be possible to do real experiment involving the shoulder prototype. The shoulder prototype and the model we developed have 2DOFs, however to emulate the DOFs of a human shoulder another rotational joint can be added in series to the moving platform.

For the EEG integration part, is possible to use a motor imagery dataset to pars the shoulder movements and their duration time (in the source code they are stated as x, y, z directions) and then use this information to control actuators 1, 2, and 3 correspondingly to move the shoulder joint. So, firstly we can test the system without EEG with the motor imagery dataset and when we can make it work we will use EEG to directly do the control. We will also focus on implementing prediction. In this code, for example, we will only have a com port for the shoulder teensy 3.5, because the IP address is here for the UR5.

The software part using EEG data acquired from the brain from an experiment in [8] was adapted so that it controls the spherical joint that we have discussed in this thesis instead of the UR-5 manipulator. So, the signals that are useful to rotate the spherical

joint in our shoulder were used for the software to send it to the Teensy micro-controller to see the movement in the spherical joint. However, testing the system with EEG cap and the software is left as future work.

3.9 Approach and methods to resolve the matter

We are working on this shoulder and have some outcome which is a robotic shoulder that can be controlled using the interface. We will also be working with V-REP to simulate the shoulder joint to be able to test all the movements before we finish the whole development (see some related paper from our university: [26]), but it now also works in the real life.

3.10 Encountered Research and Technical Challenges

The main challenge is that it is difficult to make a 3-DOF shoulder joint. The human shoulder joint is not easy to replicate, because as an organism we have everything that supports our joints: lubrication, sphere-like mechanisms, and everything that evolved during human evolution time. The second problem is due to the tendon-driven mechanism. Tendons are moved by pulleys attached to motors equipped with gearbox. If the motor by rotating unwind the tendon till the end and continue is motion it will start to wind the tendon in the opposite direction and the control system will stop to work. Therefore a mechanism need to be implemented to always keep the absolute position of the tendon.

The third problem is represented by possible collisions between the parts of the shoulder. For example, the limbs should not touch the platform during the motion. Thus, it is important to have an accurate geometric model of the manipulator. The fourth problem is a real-time signal transfer problem. And measuring these systems with sensors is also difficult if you aim to make a low-cost design. For example, in our case, at first the signals from the force sensors to the Teensy 3.6 was delayed of 1-2 seconds.

The fifth problem is an integration problem. As a whole system all sensors, actuators, and signals from the controller should work smoothly. And the interface should be working synchronously as well. This is one of the challenging parts of the shoulder joint design.

I found the biggest bug that many can encounter in Arduino, it is common to send data

with Serial.print() and Serial.println() The serial print turns integer and character into a byte array that is why it is commonly used. But sometimes sending several Serial.print commands or Serial.print inside the loop, makes Serial port filled with different byte arrays. End when reading in client applications like Python, these byte arrays can mess up. It leads to not adequate reading or corrupt data. The solution is that everything has to be put inside char or byte array, concatenated strcpy, and strcat, and send in one command. with termination symbol 'backslash 0'. It is not recommended to use std::string library in Arduino, since it results in memory overwriting using characters and bytes was time-consuming. For one formatted character:

Listing 3.1: Arduino spherical joint control part

```
val_f = scale[j].get_units();
      char line[10];
      sprintf(line, "%.1f", val_f);
      strcat(line_arr,line);
      strcat(line_arr,"|");
```

but results were successful.

Chapter 4

Testing and results

Our main goal was to achieve full control of the shoulder joint using a graphical interface, thus allow monitoring and controlling the joint position and tendons forces. In addition we developed a V-REP model to simulate the shoulder joint so that to test all the movements before we finish the whole development.

All the necessary resources to develop the model and the real prototype were available in our laboratory or at NU. The total cost of the shoulder joint materials is approximately under 500$ (See the table in the Appendix A). In Figure 4-1 is represented a first version of the final prototype that mount rigid tendons made out of metal. The three motors that control the lengths and forces of the tendons are visible in the back of the fixed platform (in green). The green case include the pulleys were tendons are winded.

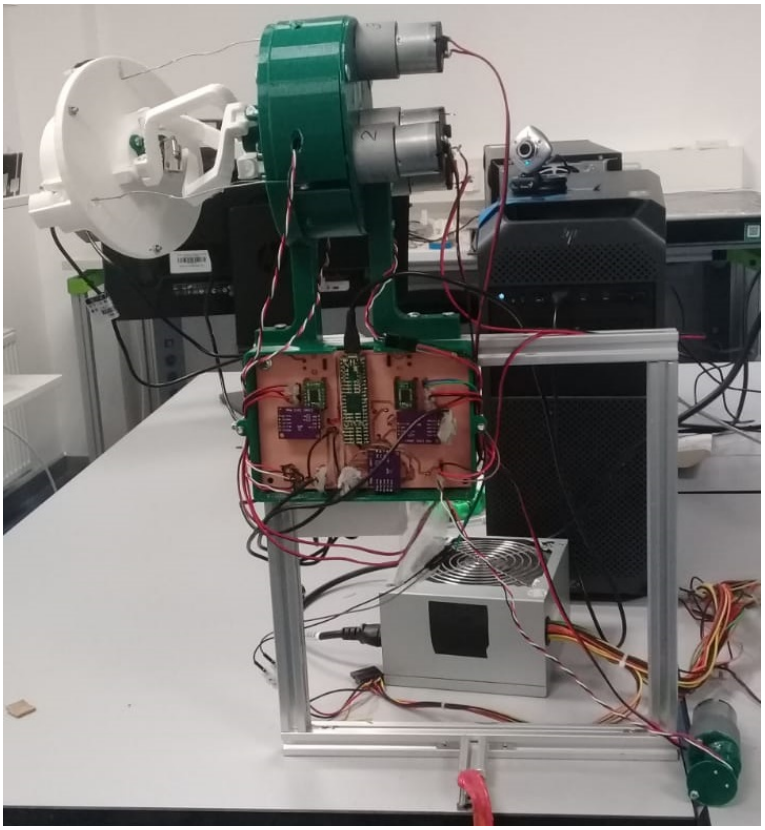


Figure 4-1: Proof of concept for a 3-DOF shoulder joint (only two actuated) realized in PLA. The shoulder joint is bent to the target position.

In Figure 4-2 the final prototype is represented where the rigid metal tendons were substituted with flexible one. This allow to decrease the rigidity of the joint and reduce the risk of damaging the prototype in case of failures of the control system. In addition metal supports were installed on the moving platform to reduce the friction. Force sensors, load cells, were installed in the moving platform to measure the forces applied by the tendons.

4.1 Force sensor experiment results

In this experiment we moved the real prototype in all the possible directions while monitoring the value of the tensions in the three tendons. This required to coordinate the tendons in order to avoid over-constrained configurations.

The Figures 4-3 and 4-4 show the act-left of about 90 degrees. While figure 4-4 shows the three force sensors measurements.

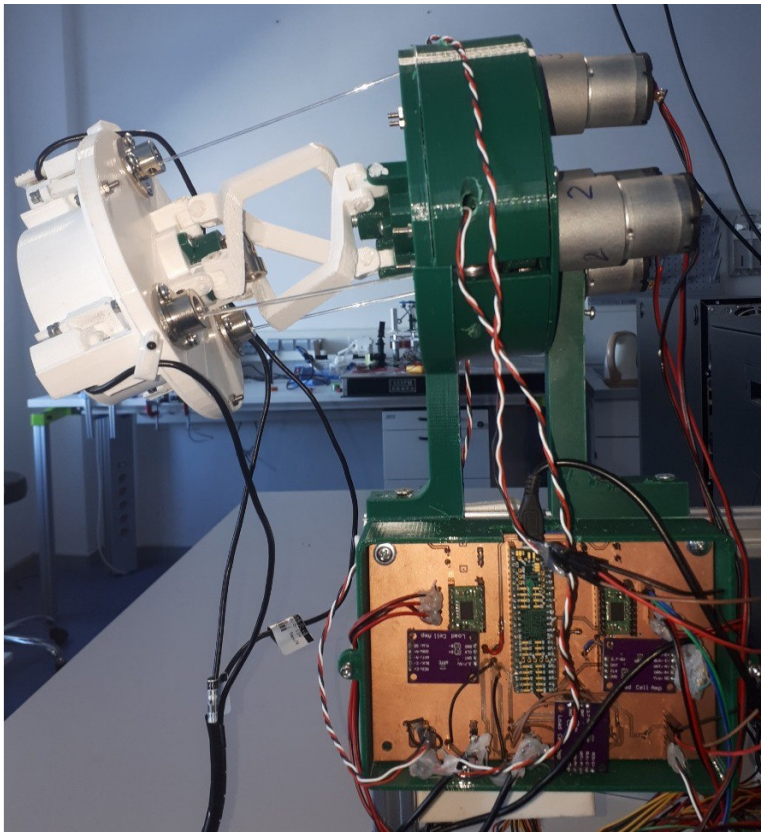


Figure 4-2: Proof of concept for a 3-DOF shoulder joint with an elastic tendon (only two actuated) realized in PLA. The shoulder joint is bent to the target position.

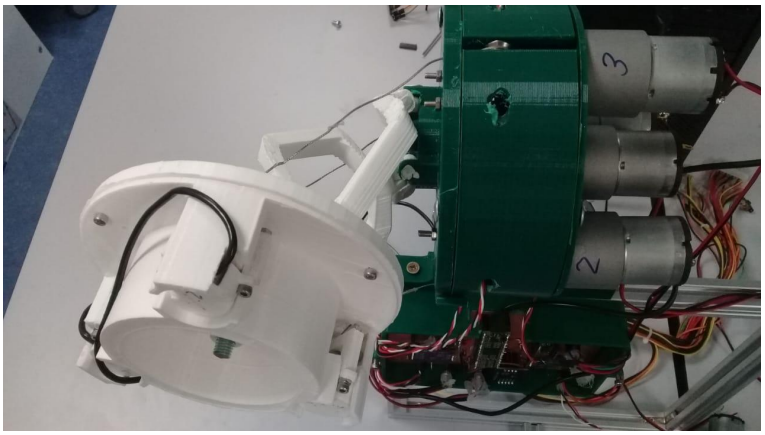


Figure 4-3: Act left about 90 degrees

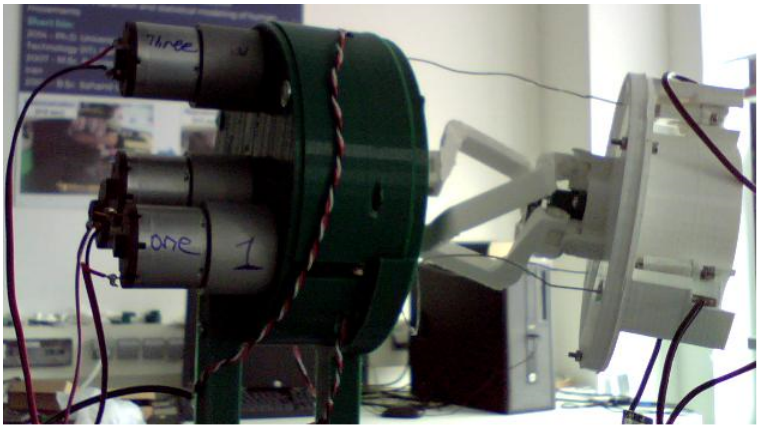


Figure 4-5: Act-right about 30 degrees and Act-top -45 degrees

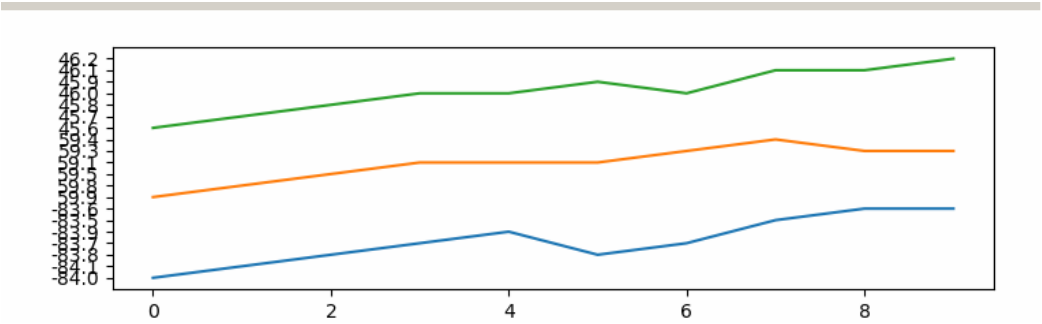


Figure 4-6: Act-right about 30degrees and Act-top -45 degrees dynamic graph

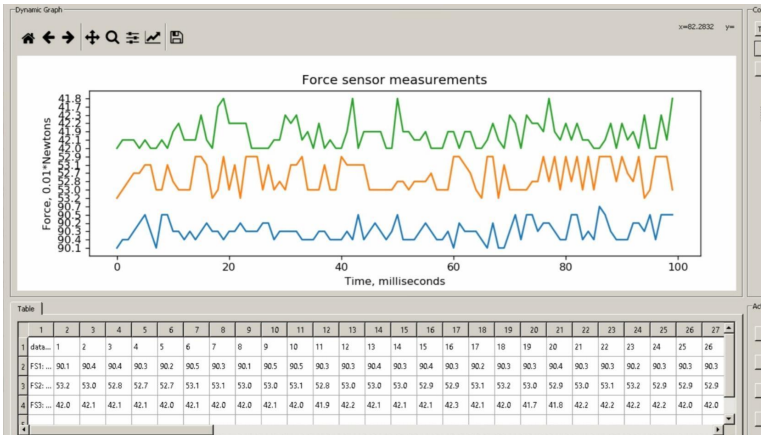


Figure 4-4: Act left about 90 degrees, 3 force sensor measurements

The figures 4-5 and 4-6 show the act-right of about 30 degrees and act-top of -45 degrees (the joint is rotated to the bottom, the blue line represents the tendon that experiences the high value of tension the orange line the tendon with the second highest tension (motor which rotates the joint to the right), force sensor value should be multiplied by 0.01 Newton).

Act-left of 75 degrees is represented in the figures 4-7 and 4-8 (the green line repre-

sents the tendon that experiences the high value of tension, force sensor value should be multiplied by 0.01Newton).

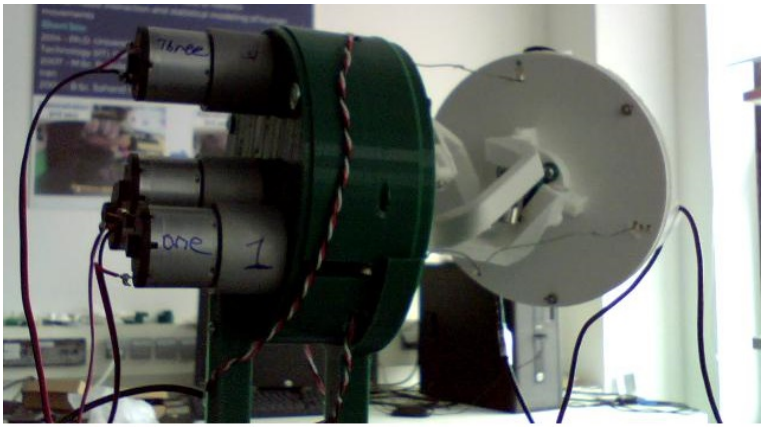


Figure 4-7: Act-left 75 degrees

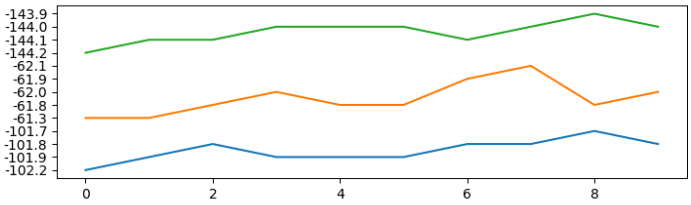


Figure 4-8: Act-left 75 degrees dynamic graph

4.2 Position encoder experiment results

In order to measure the absolute position of the tendons a counter mechanism was implemented to allow measuring rotation of the motor from 0 to 360 $\times$ 3 degrees. This required to store the last position of the motor before shutting down the system. To observe the behavior of the pulley while winding and unwinding the tendon we build a small motor setup, see Figure 4-9

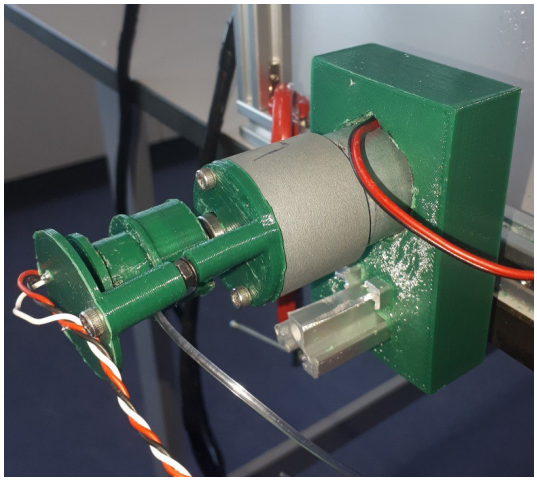


Figure 4-9: Position encoder experimental setup

In figure 4-10, continuous line, we can see the position of the encoder increasing from the last recorded position to 360 degrees then reset to zero and increase again. As dashed line we can see the absolute pulley position ranging from -480 to +480 degrees.

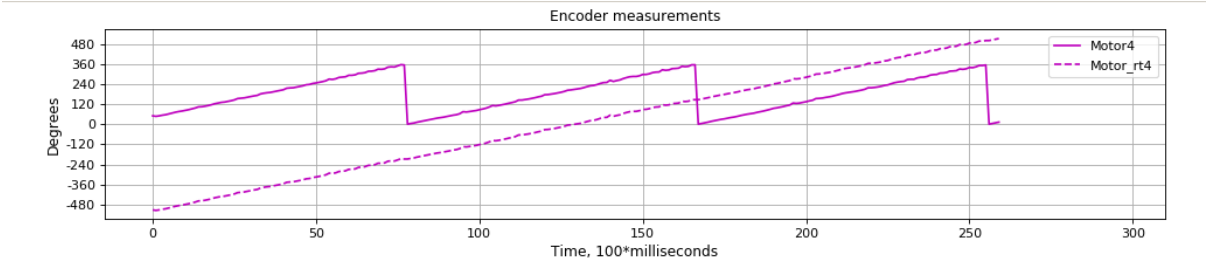


Figure 4-10: Position encoder on forward direction

In figure 4-11, continuous line, we can see the position of the encoder decreasing from 360 degrees to zero. As dashed line we can see the absolute pulley position ranging from 480 to -480 degrees. The second plot of this figure reports the tensions of the three tendons.

Figures 4-12 and 4-13 show two experiments where the reading from the encoder failed and therefore the position signal present discontinuities.

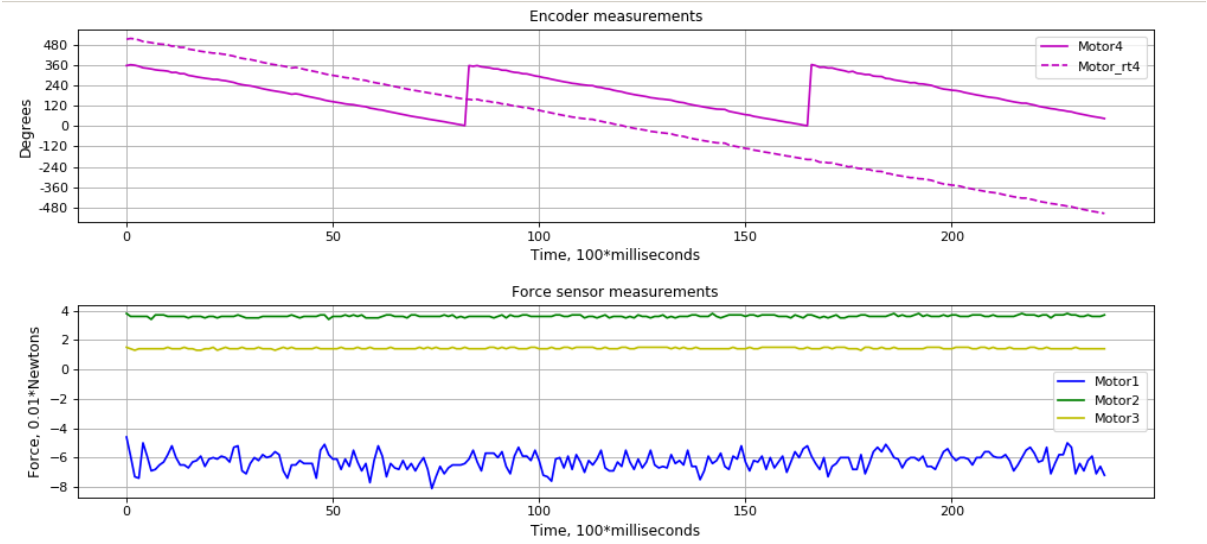


Figure 4-11: Position encoder on reverse direction

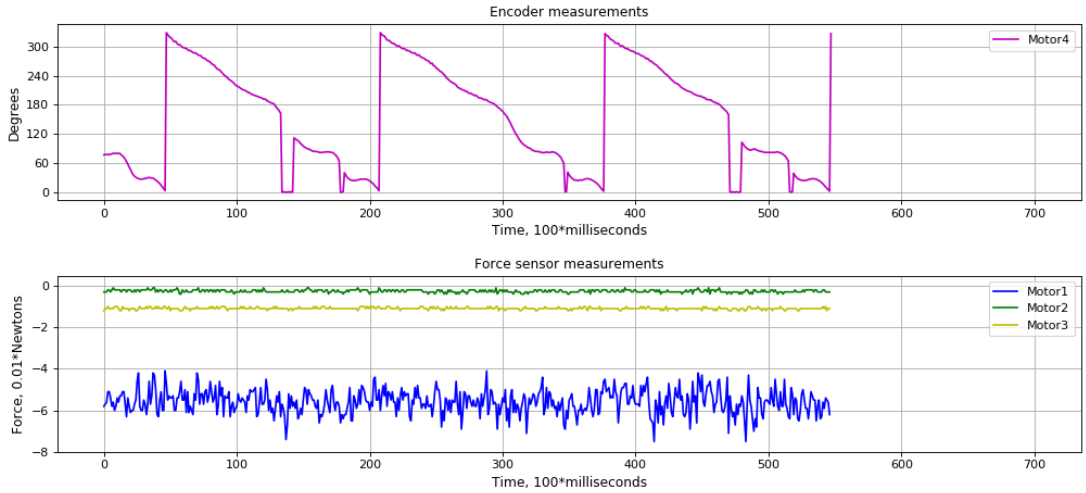


Figure 4-12: Position encoder forward 3 complete rotations

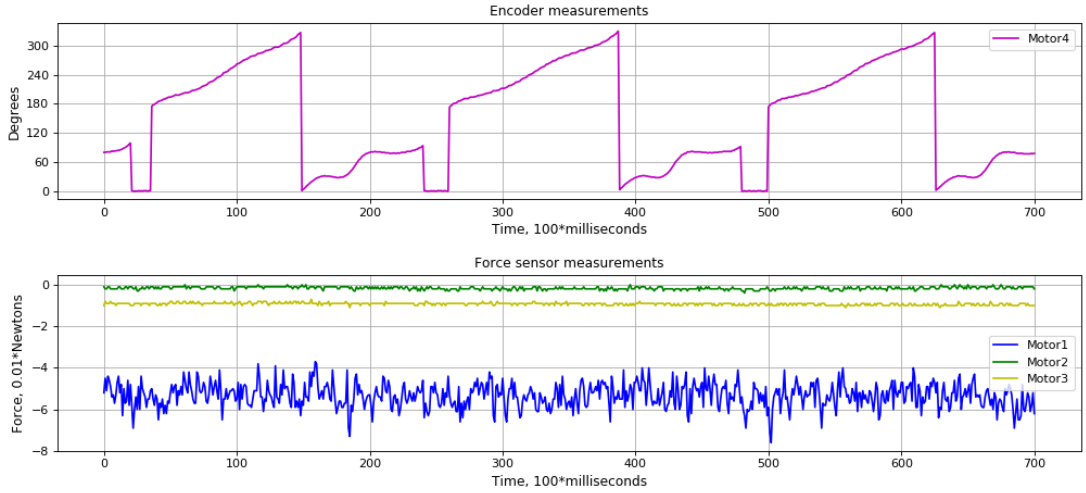


Figure 4-13: Position encoder reverse 3 complete

4.3 Workspace Measurements

In this experiment the model of the robot was moved in order to cover different points of the workspace. In particular the two angles defined the orientation of the moving platform were range in the interval 0-90 degrees.

Figure 4-14 shows the second tab of the interface, the workspace can be drawn according to the angle of the rotation given to the moving platform.

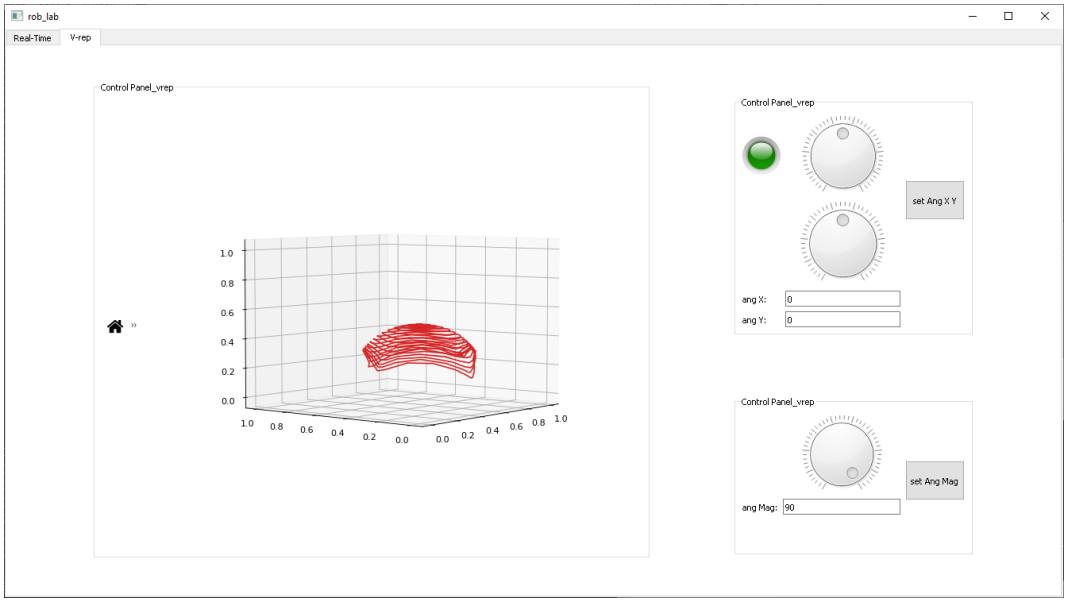


Figure 4-14: Workspace of a imitating V-REP model represented in 3d on GUI

As it is possible to notice the workspace resemble the surface of a sphere which radius is equal to the diameter of one of the two spheres that represent the rolling motion.

In Figure 4-15 the workspace of the shoulder joint in the 3D space, in figure 4-16 it is reported the projection of the workspace in the x-y plane.

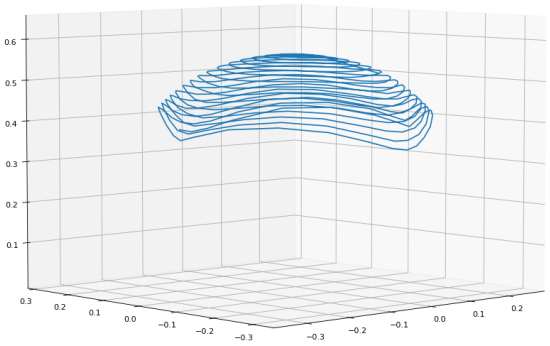


Figure 4-15: Workspace of the shoulder joint obtained from the V-REP model.

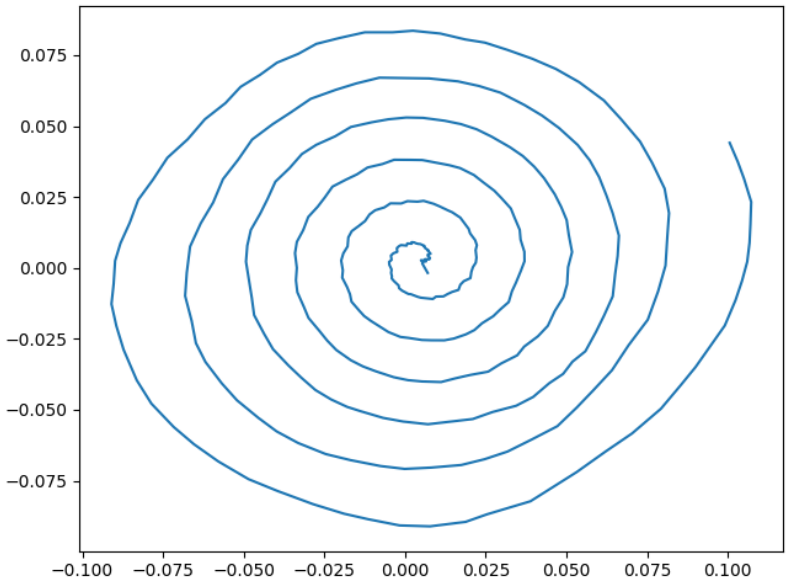


Figure 4-16: Projection of the workspace on the x-y plane. The workspace was obtained from the V-REP model.

Chapter 5

Conclusion and future work

In this work a shoulder joint was developed for application in humanoid robotics. The mechanism has a parallel kinematic structure with three limbs that allow a total of 2DOFs. The shoulder joint is actuated with three tendons that are controlled by revolute motors. Magnetic encoders were installed to estimate the lengths of the tendons and therefore the orientation of the moving platform. In addition force sensors were installed to measure the tension of the tendons and therefore the forces applied to the moving platform.

A kinematic model was developed to allow controlling the robot. A realistic kinematic-dynamic model was developed in V-REP software. A graphical interface was designed in order to monitor all the sensory data and conduct a series of experiments on the model and the real prototype.

The workspace of the shoulder joint was measured by controlling the lengths of the tendons within their maximum range.

Overall we have achieved full control of tendon actuated shoulder joint. We have learned that in the V-REP simulation there are limitations due to the difficulty to represent correctly the tendon behavior. The kinematic model we developed is based on the simplifying assumption that one limb moves and the other two limbs will follow the first one. The geometric method finds the length d_i of the manipulator which gives us the position and orientation of the moving platform. This method requires further investigation and research in the future.

Appendix A

GUI software

A.1 GUI software in python

Listing A.1: Python interface

```
from PyQt5.QtCore import QDateTime, Qt, QTimer
from PyQt5.QtWidgets import (QApplication, QCheckBox, QComboBox,
QDateTimeEdit,
        QDial, QDialog, QGridLayout, QGroupBox, QHBoxLayout, QLabel,
        QLineEdit,
        QProgressBar, QPushButton, QRadioButton, QScrollBar,
        QSizePolicy,
        QSlider, QSpinBox, QStyleFactory, QTableWidget, QTabWidget,
        QTextEdit,
        QTableWidgetItem, QVBoxLayout, QWidget, QHeaderView)
from PyQt5.QtGui import *
import time
import serial
from QLed import QLed

from PyQt5 import uic, QtCore, QtWidgets
from PyQt5.QtCore import QObject, QThread, pyqtSignal, pyqtSlot
from matplotlib.backends.backend_qt4agg import FigureCanvasQTAgg as
FigureCanvas
from matplotlib.backends.backend_qt4agg import NavigationToolbar2QT
    as
NavigationToolbar
```

```
22 from matplotlib.figure import Figure
23
24 import math
25
26
27 import matplotlib
28 matplotlib.use('Qt5Agg')
29 from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg,
30 NavigationToolbar2QT as NavigationToolbar
31
32 class list_holder():
33
34     def __init__(self, enc_zero_cnt, frs_zero_cnt, init_deg, enc_val,
35     rt_val, frs_val, rt_arr, n_cnt):
36         self.enc_zero_cnt = enc_zero_cnt
37         self.frs_zero_cnt = frs_zero_cnt
38         self.init_deg = init_deg
39         self.enc_val = enc_val
40         self.rt_val = rt_val
41         self.frs_val = frs_val
42         self.rt_arr = rt_arr
43         self.n_cnt = n_cnt
44
45
46 class fetchData(QObject):
47
48     finished = pyqtSignal(list_holder)
49     progress = pyqtSignal(int)
50
51     def __init__(self):
52         QThread.__init__(self)
53         self.active = True
54
55     def __del__(self):
56         self.wait()
57
58     def _fetch(self):
59         n_cnt = 0
60
```

```
61          enc_list = []
62          init_deg = [0, 0, 0, 0, 0]
63
64          enc_val = [[], [], [], [], []]
65          rt_val = [[], [], [], [], []]
66          rt_arr = [[], [], [], [], []]
67
68          frs_list = []
69          frs_val = [[], [], [], []]
70
71          enc_zero_cnt = [0, 0, 0, 0, 0]
72          frs_zero_cnt = [0, 0, 0, 0, 0]
73
74          ser=serial.Serial('COM3',9600, timeout=7)
75
76          print("Start reading...")
77
78          while(n_cnt<=700 and self.active):
79              frs = ser.readline()
80              print(frs)
81              frs_str = str(frs)
82              if ("fs" in frs_str):
83                  frs_list = frs_str.split("|", 5)
84
85                  for i in range(1, 4):
86                      if float(frs_list[i])==0.0:
87                          frs_zero_cnt[i] += 1
88                      frs_val[i].append(float(frs_list[i]))
89
90              enc = ser.readline()
91              print(enc)
92              enc_str = str(enc)
93              if ("enc" in enc_str):
94                  enc_list = enc_str.split("|", 10)
95                  if (n_cnt == 0):
96                      init_deg[1:5] = enc_list[1:5]
97
98                  for i in range(1, 5):
99                      if int(enc_list[i])==0:
```

```
100                         enc_zero_cnt[i] += 1
101                     enc_val[i].append(int(enc_list[i]))
102 
103                     if ":" in enc_list[i+4]:
104                         val = enc_list[i+4]
105                         rt_arr[i] = val.split(":",2)
106                         rt_val[i].append(int(rt_arr[i][1]))
107 
108             if enc_list[9]=="off":
109                 self.enc_last = enc_list
110                 print("Limit reached. Please rotate in different
111                 direction")
112                 break;
113 
114             n_cnt = n_cnt + 1
115 
116         ser.close()
117         print("End")
118 
119         self.ret_lists = list_holder(enc_zero_cnt, frs_zero_cnt,
120         init_deg, enc_val, rt_val, frs_val, rt_arr, n_cnt)
121 
122         return self.ret_lists
123 
124     def run(self):
125         self.active = True;
126         obj_lists = self._fetch()
127         self.finished.emit(obj_lists)
128 
129     def stop(self):
130         self.active = False;
131         self.wait()
132 
133 
134 
135 
136 class MplCanvas(FigureCanvasQTAgg):
137     def __init__(self, parent=None, width=5, height=4, dpi=120):
138         fig = Figure(figsize=(width, height), dpi=dpi)
```

```
139         self.axes = fig.add_subplot(111)
140         super(MplCanvas, self).__init__(fig)
141 
142 class WidgetGallery(QDialog):
143     def __init__(self, parent=None):
144         super(WidgetGallery, self).__init__(parent)
145         self.setFixedHeight(800)
146         self.setFixedWidth(1300)
147         styleComboBox = QComboBox()
148 
149         styleLabel = QLabel("Style:")
150 
151         self.useStylePaletteCheckBox = QCheckBox("&Use style's
152         standard palette")
153         self.useStylePaletteCheckBox.setChecked(True)
154         disableWidgetsCheckBox = QCheckBox("&Disable widgets")
155         self.createTopLeftGroupBox()
156         self.createTopRightGroupBox()
157         self.createBottomLeftTabWidget()
158         self.bottomLeftTabWidget.setFixedHeight(220)
159         self.createBottomRightGroupBox()
160         self.createProgressBar()
161 
162         topLayout = QHBoxLayout()
163 
164         mainLayout = QGridLayout()
165         mainLayout.addWidget(self.topLeftGroupBox, 1, 0)
166         mainLayout.addWidget(self.topRightGroupBox, 1, 1)
167 
168         mainLayout.addWidget(self.bottomLeftTabWidget, 2, 0)
169         mainLayout.addWidget(self.bottomRightGroupBox, 2, 1)
170         self.setLayout(mainLayout)
171         self.setWindowTitle("Styles")
172         self.changeStyle('Windows')
173         self.createThreads()
174 
175 
176     def changeStyle(self, styleName):
177         QApplication.setStyle(QStyleFactory.create(styleName))
```

```
178         self.changePalette()
179 
180     def changePalette(self):
181         if (self.useStylePaletteCheckBox.isChecked()):
182             QApplication.setPalette(QApplication.style()
183             .standardPalette())
184         else:
185             QApplication.setPalette(self.originalPalette)
186     def advanceProgressBar(self):
187         curVal = self.progressBar.value()
188         maxVal = self.progressBar.maximum()
189         self.progressBar.setValue(curVal + (maxVal - curVal) // 100)
190 
191     def createTopLeftGroupBox(self):
192         self.topLeftGroupBox = QGroupBox("Dynamic Graph")
193 
194         self.figure = Figure(figsize=(8,6), dpi=80)
195         gs = self.figure.add_gridspec(2, 1)
196         self.ax1 = self.figure.add_subplot(gs[0, 0])
197         self.ax2 = self.figure.add_subplot(gs[1, 0])
198         self.ax1.grid(True)
199         self.ax2.grid(True)
200         self.canvas = FigureCanvas(self.figure)
201         self.toolbar = NavigationToolbar(self.canvas, self)
202         self.ax1.set_ylabel("Degrees", fontsize = 11)
203         self.ax1.set_xlabel("Time, 100*milliseconds", fontsize = 11)
204         self.ax1.set_title("Encoder measurements", fontsize = 11)
205         self.ax1.set_ybound(-540, 540)
206         self.ax1.set_yticks([-480, -360, -240, -120, 0, 120, 240,
207         360, 480])
208         self.ax1.set_yticklabels(('-480', '-360', '-240', '-120',
209         '0', '120', '240', '360', '480'))
210 
211         #self.ax2.set_ylim(auto=True)
212         self.ax2.set_ybound(-10, 500)
213         self.ax2.set_ylabel('Force, 0.01*Newtons', fontsize = 11)
214         self.ax2.set_xlabel("Time, 100*milliseconds", fontsize = 11)
215         self.ax2.set_title("Force sensor measurements", fontsize =
              11)
```

```
216
217
218         self.ax1.set_position([0.1, 0.6, 0.85, 0.35])
219         self.ax2.set_position([0.1, 0.1, 0.85, 0.35])
220
221         layout = QVBoxLayout()
222         layout.addWidget(self.toolbar)
223         layout.addWidget(self.canvas)
224
225         self.show()
226         self.topLeftGroupBox.setLayout(layout)
227
228     def createBottomLeftTabWidget(self):
229         self.bottomLeftTabWidget = QTabWidget()
230         self.tab1 = QWidget()
231         self.tab1.resize(1600,400)
232         self.tableWidget = QTableWidget(9, 1000)
233         header = self.tableWidget.horizontalHeader()
234         header.setSectionResizeMode(QHeaderView.Stretch)
235         self.tableWidget.resize(120,80)
236         self.tab1hbox = QHBoxLayout()
237         self.tab1hbox.addWidget(self.tableWidget)
238         self.tab1.setLayout(self.tab1hbox)
239         self.bottomLeftTabWidget.addTab(self.tab1, "&Table")
240
241     def dialMoved(self):
242         val = self.dial3.value()
243         self.label.setText(" Dialer Value : " + str(val))
244
245     def createBottomRightGroupBox(self):
246         self.bottomRightGroupBox = QGroupBox("Actuators")
247         self.Button_Right = QPushButton("Act: 1")
248         self.Button_Right.setCheckable(True)
249         self.Button_Right.clicked.connect(self.Button_Right_Handler)
250         self.Button_Left = QPushButton("Act: 2")
251         self.Button_Left.setCheckable(True)
252         self.Button_Left.clicked.connect(self.Button_Left_Handler)
253         self.Button_Top = QPushButton("Act: 3")
254         self.Button_Top.setCheckable(True)
```

```
255         self.Button_Top.clicked.connect(self.Button_Top_Handler)
256         self.Button_Roll = QPushButton("Act: Roll")
257         self.Button_Roll.setCheckable(True)
258         self.Button_Roll.clicked.connect(self.Button_Roll_Handler)
259 
260 
261         self.Zero_Right = QPushButton("Set: 0")
262         self.Zero_Right.clicked.connect(lambda: self.setZero(1))
263         self.Zero_Left = QPushButton("Set: 0")
264         self.Zero_Left.clicked.connect(lambda: self.setZero(2))
265         self.Zero_Top = QPushButton("Set: 0")
266         self.Zero_Top.clicked.connect(lambda: self.setZero(3))
267         self.Zero_Roll = QPushButton("Set: 0")
268         self.Zero_Roll.clicked.connect(lambda: self.setZero(4))
269 
270 
271         #layout2 = QVBoxLayout()
272         layout2 = QGridLayout()
273         layout2.addWidget(self.Button_Right, 1, 0)
274         layout2.addWidget(self.Button_Left, 2, 0)
275         layout2.addWidget(self.Button_Top, 3, 0)
276         layout2.addWidget(self.Button_Roll, 4, 0)
277 
278         layout2.addWidget(self.Zero_Right, 1, 1)
279         layout2.addWidget(self.Zero_Left, 2, 1)
280         layout2.addWidget(self.Zero_Top, 3, 1)
281         layout2.addWidget(self.Zero_Roll, 4, 1)
282 
283         self.bottomRightGroupBox.setLayout(layout2)
284         self.motors = [0, 0, 0, 0, 0]
285         self.pos_last = [0, 0, 0, 0, 0]
286         self.enc_last = [0, 0, 0, 0, 0]
287     def createTopRightGroupBox(self):
288 
289         self.topRightGroupBox = QGroupBox("Control Panel")
290 
291         stopAllButton = QPushButton("Turn off the actuators")
292         stopAllButton.setDefault(True)
293         stopAllButton.clicked.connect(self.StopAll_Handler)
```

```
294
295         readForceButton=QPushButton("Read force values")
296         readForceButton.setDefault(True)
297         readForceButton.clicked.connect(self.readForce)
298
299
300         readEncButton=QPushButton("Read encoder")
301         readEncButton.setDefault(True)
302         readEncButton.clicked.connect(self.readEncoder)
303
304         self.dial3 = QDial(self.topRightGroupBox)
305         self.dial3.setMinimum(-120)
306         self.dial3.setMaximum(120)
307         self.dial3.setNotchesVisible(True)
308         self.dial3.valueChanged.connect(self.dialMoved)
309
310         self.dialInfo = QLabel()
311         self.dialInfo.setText("           " + str(self.dial3.minimum
                ())
312         + "                " + str(self.dial3.maximum()))
313
314
315         self.label = QLabel()
316
317         layout = QVBoxLayout()
318         layout.addWidget(stopAllButton)
319         layout.addWidget(readForceButton)
320         layout.addWidget(readEncButton)
321         layout.addWidget(self.dial3)
322         layout.addWidget(self.dialInfo)
323         layout.addWidget(self.label)
324
325
326         self.led1=QLed(self, onColour=QLed.Red, shape=QLed.Circle)
327         self.led1.value=False
328
329         self.led2=QLed(self, onColour=QLed.Red, shape=QLed.Circle)
330         self.led2.value=False
331
```

```
332         self.led3=QLed(self, onColour=QLed.Red, shape=QLed.Circle)
333         self.led3.value=False
334 
335         self.led4=QLed(self, onColour=QLed.Red, shape=QLed.Circle)
336         self.led4.value=False
337 
338         self.light = 0
339 
340         layout.addWidget(self.led1)
341         layout.addWidget(self.led2)
342         layout.addWidget(self.led3)
343         layout.addWidget(self.led4)
344         layout.addStretch(1)
345         self.topRightGroupBox.setLayout(layout)
346 
347     def sendSerial(self):
348         print("cmd sent: ", self.cmd)
349         ser=serial.Serial('COM3',9600, timeout=3)
350         ser.write(self.cmd)
351         ser.close()
352         """
353         ser=serial.Serial('COM3',9600, timeout=3)
354         line = ser.readline()
355         print("cmd received: ", line)
356         ser.close()
357         """
358 
359     def writeFile(self):
360         pos_str = 'pos:|0|'
361         enc_str = 'enc:|0|'
362         for i in range(1,5):
363             pos_str = pos_str + str(self.pos_last[i]) + '|'
364             enc_str = enc_str + str(self.enc_last[i]) + '|'
365         pos_str = pos_str + '\n'
366         enc_str = enc_str + '\n'
367         with open("ctrl.cfg","w") as ctrl_cfg:
368             ctrl_cfg.write(pos_str)
369             ctrl_cfg.write(enc_str)
370 
```

```
371     def readFile(self):
372         try:
373             ctrl_cfg = open("ctrl.cfg","r")
374             self.pos_last = ctrl_cfg.readline().split("|", 6)
375             self.pos_last = self.pos_last[1:6]
376             self.pos_last = [int(i) for i in self.pos_last]
377             self.enc_last = ctrl_cfg.readline().split("|", 6)
378             self.enc_last = self.enc_last[1:6]
379             self.enc_last = [int(i) for i in self.enc_last]
380             ctrl_cfg.close()
381         except:
382             print("cfg is empty or not correct")
383 
384         if len(self.pos_last)>=4:
385             cmd_bytes = b''
386             for i in range(5,9):
387                 cmd_bytes = cmd_bytes + str(i).encode("utf-8") +
388                 b':' + str(self.pos_last[i-4]).encode("utf-8") + b'|'
389         else:
390             cmd_bytes = bytes("5:0|6:0|7:0|8:0|", 'utf-8')
391             self.pos_last = [0, 0, 0, 0, 0]
392             self.enc_last = [0, 0, 0, 0, 0]
393         self.cmd = self.cmd + cmd_bytes
394 
395     def cmd_line(self):
396         stop_cmd = False
397         self.cmd = b'0' + b':' + str(self.motors[0]).encode("utf-8")
398         + b'|'
399         for i in range(1,5):
400             motor_byte = str(i).encode("utf-8")
401             if (self.motors[i]==1):
402                 val_byte = str(self.dial3.value()).encode("utf-8")
403                 motor = i
404             elif (self.motors[i]==-1 or self.motors[i]==0):
405                 if self.motors[i]==-1:
406                     stop_cmd = True
407                 val_byte = str(self.motors[i]).encode("utf-8")
408             self.cmd = self.cmd + motor_byte + b':' + val_byte + b'|'
409 
```

```
410
411         if (self.motors[0]==-1):
412             cmd_bytes = bytes("
                    0:-1|1:0|2:0|3:0|4:0|5:0|6:0|7:0|8:0|9:0|",
413             'utf-8')
414             self.cmd = cmd_bytes
415             self.motors[0]=0
416             self.sendSerial()
417             self.writeFile()
418
419         elif (stop_cmd == True):
420             cmd_bytes = bytes("5:0|6:0|7:0|8:0|9:0|", 'utf-8')
421             self.cmd = self.cmd + cmd_bytes
422             self.motors[0:5]=[0 for el in self.motors[0:5]]
423             self.sendSerial()
424             self.writeFile()
425         else:
426             self.readFile()
427             self.sendEncFrs(motor)
428
429
430     def sendEncFrs(self, opt):
431         self.cmd = self.cmd + b'9' + b':' + b'253' + b'|'
432         self.sendSerial()
433         self.threads[opt].start()
434
435     def readEncFrs(self, fetched_lists):
436
437         enc_zero_cnt = fetched_lists.enc_zero_cnt
438         frs_zero_cnt = fetched_lists.frs_zero_cnt
439         init_deg = fetched_lists.init_deg
440         enc_val = fetched_lists.enc_val
441         rt_val = fetched_lists.rt_val
442         frs_val = fetched_lists.frs_val
443         rt_arr = fetched_lists.rt_arr
444         n_cnt = fetched_lists.n_cnt
445
446
447         deg_passed = [0,0,0,0,0]
```

```
448         dist_passed = [0,0,0,0,0]
449
450         line1 = [[], [], [], [], []]
451         line2 = [[], [], [], [], []]
452         color = ['', 'b', 'g', 'y', 'm']
453
454         upd1 = 0
455         if len(self.ax1.lines)>0:
456             self.ax1.lines.clear()
457         for i in range(1,5):
458             if (enc_zero_cnt[i]<=n_cnt-1 and len(enc_val[i])>1):
459                 line1[i], = self.ax1.plot(enc_val[i], color[i], label
                        =
460                 'Motor' + str(i))
461                 line2[i], = self.ax1.plot(rt_val[i], color[i] + '--',
462                 label='Motor_rt' + str(i))
463                 upd1 = 1
464
465         if upd1 == 1:
466             self.ax1.legend()
467             #radius is 9.2mm
468             radius = float(9.2/10**3)
469             one_degree_len = float((2*math.pi*radius)/360)
470             #degree passed = (360-init_degree) + (number of rotations
471             * 360 degrees) + current degree: _(*)_
472             for i in range(1,5):
473                 #deg_passed[i] = (360-int(init_deg[i])) +
474                 (int(rt_arr[i][0])*360) + int(self.enc_last[i])
475                 #dist_passed[i] =  one_degree_len * float(deg_passed[
                        i])
476                 self.pos_last[i] = rt_arr[i][1]
477
478
479
480         upd2 = 0
481         if len(self.ax2.lines)>0:
482             self.ax2.lines.clear()
483         for i in range(1,4):
484             if (frs_zero_cnt[i]<=n_cnt-1 and len(frs_val[i])>1):
```

```
485                 line1[i], = self.ax2.plot(frs_val[i], color[i], label
                        =
486                 'Motor' + str(i))
487                 upd2 = 1
488         if upd2 == 1:
489             self.ax2.legend()
490         if (upd1 == 1 and upd2 == 1):
491             for n in range(n_cnt-1):
492                 if n==0:
493                     self.tableWidget.setItem(0,0,
494                     QTableWidgetItem("data no:"))
495                     for i in range(1,5):
496                         self.tableWidget.setItem(i,0,
497                         QTableWidgetItem("enc" + str(i) + ":"))
498                     for i in range(1,4):
499                         self.tableWidget.setItem(i+5,0,
500                         QTableWidgetItem("FS" + str(i) + ": *0.01 N")
                                )
501                 elif n>=1:
502                     self.tableWidget.setItem(0,n,
503                     QTableWidgetItem(str(n)))
504                     for i in range(1,5):
505                         self.tableWidget.setItem(i,n,
506                         QTableWidgetItem(str(enc_val[i][n-1])))
507                     for i in range(1,4):
508                         self.tableWidget.setItem(i+5,n,
509                         QTableWidgetItem(str(frs_val[i][n-1])))
510             self.canvas.draw()
511
512         self.StopAll_Handler()
513
514     def readEncoder(self):
515         print("on work")
516
517     def readForce(self):
518         print("on work")
519
520
521     def setZero(self, opt):
```

```
522         self.pos_last[opt] = 0
523         self.enc_last[opt] = 0
524         self.writeFile()
525 
526     def createThreads(self):
527         self.threads = []
528         self.workers = []
529         self.threads.append(0)
530         self.workers.append(0)
531         for i in range(1,5):
532             self.threads.append(QThread())
533             self.workers.append(fetchData())
534             self.workers[i].moveToThread(self.threads[i])
535             self.threads[i].started.connect(self.workers[i].run)
536             self.workers[i].finished.connect(self.threads[i].quit)
537             self.workers[i].finished.connect(self.workers[i].
                    deleteLater)
538             self.threads[i].finished.connect(self.threads[i].
                    deleteLater)
539             self.workers[i].finished.connect(self.readEncFrs)
540 
541     def Button_Right_Handler(self):
542         if self.Button_Right.isChecked():
543             self.led1.setValue(True)
544             self.led1.repaint()
545             self.motors[1] = 1
546             print("act: Right Start")
547         else:
548             self.led1.value = False
549             self.led1.repaint()
550             self.motors[1] = -1
551             print("act: Right Stop")
552             self.threads[1].stop()
553         self.cmd_line()
554     def Button_Left_Handler(self):
555         if self.Button_Left.isChecked():
556             self.led2.setValue(True)
557             self.led2.repaint()
558             self.motors[2] = 1
```

```
559             print("act: Left Start")
560         else:
561             self.led2.value = False
562             self.led2.repaint()
563             self.motors[2] = -1
564             print("act: Left Stop")
565             self.threads[2].stop()
566         self.cmd_line()
567     def Button_Top_Handler(self):
568         if self.Button_Top.isChecked():
569             self.led3.setValue(True)
570             self.led3.repaint()
571             self.motors[3] = 1
572             print("act: Top Start")
573         else:
574             self.led3.value = False
575             self.led3.repaint()
576             self.motors[3] = -1
577             print("act: Top Stop")
578             self.threads[3].stop()
579         self.cmd_line()
580     def Button_Roll_Handler(self):
581         if self.Button_Roll.isChecked():
582             self.led4.setValue(True)
583             self.led4.repaint()
584             self.motors[4] = 1
585             print("act: Roll Start")
586         else:
587             self.led4.value = False
588             self.led4.repaint()
589             self.motors[4] = -1
590             print("act: Roll Stop")
591             self.threads[4].stop()
592         self.cmd_line()
593     def StopAll_Handler(self):
594         if self.Button_Right.isChecked():
595             self.Button_Right.toggle()
596         if self.Button_Left.isChecked():
597             self.Button_Left.toggle()
```

```
598         if self.Button_Top.isChecked():
599             self.Button_Top.toggle()
600         if self.Button_Roll.isChecked():
601             self.Button_Roll.toggle()
602         self.led1.value = False
603         self.led1.repaint()
604         self.led2.value = False
605         self.led2.repaint()
606         self.led3.value = False
607         self.led3.repaint()
608         self.led4.value = False
609         self.led4.repaint()
610         self.motors[0] = -1
611         print("act: Stop All")
612         self.cmd_line()
613 
614     def createProgressBar(self):
615         self.progressBar = QProgressBar()
616         self.progressBar.setRange(0, 10000)
617         self.progressBar.setValue(0)
618         timer = QTimer(self)
619         timer.timeout.connect(self.advanceProgressBar)
620         timer.start(1000)
621 
622 if __name__ == '__main__':
623     import sys
624     app = QApplication(sys.argv)
625     gallery = WidgetGallery()
626     gallery.show()
627     sys.exit(app.exec_())
```

Listing A.2: Arduino spherical joint control

```
1 
2 void loop() {
3   if (reading == 0) readSerial();
4   delay(500);
5 }
6 
```

```
7  void motor_rotate(const int pinMotor[4], int encVal) {
8    digitalWrite(pinSTBY_1, HIGH);
9    digitalWrite(pinSTBY_2, HIGH);
10   digitalWrite(pinMotor[1], HIGH);
11   digitalWrite(pinMotor[2], LOW);
12   analogWrite(pinMotor[0], encVal);
13 }
14
15 void  reverse_motor(const int pinMotor[4], int encVal) {
16   digitalWrite(pinSTBY_1, HIGH);
17   digitalWrite(pinSTBY_2, HIGH);
18   digitalWrite(pinMotor[1], LOW);
19   digitalWrite(pinMotor[2], HIGH);
20   analogWrite(pinMotor[0], encVal);
21 }
22
23 void stopMotor(const int pinMotor[4]) {
24   digitalWrite(pinMotor[1], LOW);
25   digitalWrite(pinMotor[2], LOW);
26   reading = 0;
27 }
28
29
30 void encoder_read() {
31   reading = 1;
32   int i = 0;
33   while (i <= 50) {
34     delay(100);
35     Serial.print("enc:|");
36     for (int j = 1; j <= 4; j++) {
37       pwm_value[j] = pulseIn(PWM_PINS[j], HIGH, 10000);
38       pwm_value[j] = map(pwm_value[j], 0, 918, 0, 360);
39       Serial.print(pwm_value[j]);
40       Serial.print("|");
41     }
42     Serial.println();
43     i = i + 1;
44   }
45 }
```

```
46
47 void force_init() {
48
49   long zero_factor[4];
50
51   for (int j = 1; j <= 3; j++) {
52     scale[j].begin(DOUT[j], CLK[j]);
53     scale[j].set_scale();
54     scale[j].tare(); //Reset the scale to
55     //Get a baseline reading
56     zero_factor[j] = scale[j].read_average();
57   }
58 }
59
60 void force_read() {
61   reading = 1;
62   int i = 0;
63   //while(true){
64   while (i <= 10) {
65     //force sensor code
66     Serial.print("fs:|");
67     for (int j = 1; j <= 3; j++) {
68       scale[j].set_scale(calibration_factor[j]);
69       Serial.print(scale[j].get_units(), 1);
70       Serial.print("|");
71     }
72     Serial.println();
73     delay(100);
74     i = i + 1;
75   }
76 }
77
78 void enc_force_read() {
79   reading = 1;
80
81   int diff[5] = {0, 0, 0, 0, 0}, prevValue[5] = {0, 0, 0, 0, 0}, rot
       [5] = {0, 0, 0, 0, 0};
82   int rot_lim = 514, i = 0;
83
```

```
84  bool exceeded = false;
85
86  while (i <= 700) {
87    if (i%30==0) readSerial();
88    else{
89      delay(300);
90    }
91
92    //force sensor code
93    Serial.print("fs:|");
94    for (int j = 1; j <= 3; j++) {
95      scale[j].set_scale(calibration_factor[j]);
96      Serial.print(scale[j].get_units(), 1);
97      Serial.print("|");
98    }
99    Serial.println();
100
101   //encoder code
102   Serial.print("enc:|");
103   for (int j = 1; j <= 4; j++) {
104     pwm_value[j] = pulseIn(PWM_PINS[j], HIGH, 10000);
105     pwm_value[j] = map(pwm_value[j], 0, 918, 0, 360);
106     Serial.print(pwm_value[j]);
107     Serial.print("|");
108   }
109   for (int j = 1; j <= 4; j++) {
110     if (i == 0) {
111       diff[j] = 0;
112     } else {
113       diff[j] = pwm_value[j] - prevValue[j];
114     }
115     if (abs(diff[j]) >= 300) {
116       if (diff[j] > 0) {
117         rot[j]--;
118       } else if (diff[j] < 0) {
119         rot[j]++;
120       }
121     } else {
122       if (diff[j] > 0) {
```

```
123             if (abs(diff[j]) < 20) lastValue[j] = lastValue[j] + abs(
                    diff[j]);
124             if (i>=1 && lastValue[j] >= rot_lim) {
125               stopMotor(pinMotors[j]);
126               exceeded = true;
127             }
128           }else if (diff[j] < 0){
129             if (abs(diff[j]) < 20) lastValue[j] = lastValue[j] - abs(
                    diff[j]);
130             if (i>=1 && lastValue[j] <= -rot_lim) {
131               stopMotor(pinMotors[j]);
132               exceeded = true;
133             }
134           }
135         }
136         Serial.print(rot[j]);
137         Serial.print(":");
138         Serial.print(lastValue[j]);
139         Serial.print("|");
140         prevValue[j] = pwm_value[j];
141       }
142
143       if (exceeded) {
144         Serial.print("off");
145         Serial.print("|");
146         Serial.println();
147         //SD_card();
148         reading = 0;
149         break;
150       } else {
151         Serial.print("on");
152         Serial.print("|");
153         Serial.println();
154       }
155       i = i + 1;
156     }
157 }
158
159
```

```
160
161
162
163
164 void readSerial(){
165
166   int MaxSpeed = 120, cnt=0, n = 0;
167   char input[INPUT_SIZE + 1];
168   byte size = Serial.readBytes(input, INPUT_SIZE);
169   input[size] = 0;
170   char* cmd = strtok(input, "|");
171
172   while (cmd != 0) {
173     char* separator = strchr(cmd, ':');
174     if (separator != 0)
175     {
176       *separator = 0;
177       opt[n] = atoi(cmd);
178       ++separator;
179       val[n] = atoi(separator);
180     }
181     cmd = strtok(0, "|");
182     n++;
183   }
184   /*
185   if (val[9]!=NULL){
186     for (int i=0;i<=9;i++){
187       Serial.print(val[i]);
188       Serial.print("|");
189     }
190     Serial.println();
191   }else{
192   */
193   for (int i=0;i<=9;i++){
194     if (val[i]==0 || val[i]==NULL) cnt++;
195   }
196
197   if (cnt<10){
198     for (int i = 1; i <= 9; i++) {
```

```
199       if (opt[0] == 0 && val[0] == -1) {
200         stopMotor(pinMotors[i]);
201       } else if (opt[i] >= 1 && opt[i] <= 4) {
202         if (val[i] == -1) {
203           stopMotor(pinMotors[i]);
204         } else if (val[i] > 0 && abs(val[i]) <= MaxSpeed) {
205           reverse_motor(pinMotors[i], abs(val[i]));
206         } else if (val[i] < 0 && abs(val[i]) <= MaxSpeed) {
207           motor_rotate(pinMotors[i], abs(val[i]));
208         }
209       } else if (opt[i] >= 5 && opt[i] <= 8) {
210         if (reading == 0) {
211           servoId[i-4] = opt[i];
212           lastValue[i-4] = val[i];
213         }
214       } else if (opt[i] == 9 && val[i]!=NULL && reading == 0){
215         switch (val[i]) {
216           case 253:
217             enc_force_read();
218             break;
219           case 254:
220             encoder_read();
221             break;
222           case 255:
223             force_read();
224             break;
225         }
226       }
227     }
228   }
229 }
230
231
232 void SD_card() {
233
234     if (!SD.begin(10)) {
235         Serial.println("SD initialization failed!");
236         while (1);
237     }
```

```
238     Serial.println("initialization done.");
239
240     file = SD.open("test.txt", FILE_WRITE);
241
242     if (file) {
243         file.print("0|");
244         for (int i=1; i<=4; i++){
245           file.print(lastValue[i]);
246           file.print("|");
247         }
248         file.println();
249         file.close();
250     } else {
251         Serial.println("error opening test.txt");
252     }
253 }
254
255 void stop_all() {
256   reading = 0;
257   stopMotor(pinMotor1);
258   stopMotor(pinMotor2);
259   stopMotor(pinMotor3);
260   stopMotor(pinMotor4);
261 }
```

Bibliography

[1] Ball, S. J., Brown, I. E., & Scott, S. H. Medarm: a rehabilitation robot with 5dof at the shoulder complex. *IEEE/ASME international conference on Advanced intelligent mechatronics*, 2007.

[2] Bhattacharyya, S., Konar, A., & Tibarewala, D. N. . Motor imagery, p300 and error-related eeg-based robot arm movement control for rehabilitation purpose. *Medical & biological engineering & computing*, 52(12):1007–1017, 2014.

[3] Buongiorno, D., Sotgiu, E., Leonardis, D., Marcheschi, S., Solazzi, M., & Frisoli, A. Wres: a novel 3 dof wrist exoskeleton with tendon-driven differential transmission for neuro-rehabilitation and teleoperation. *IEEE Robotics and Automation Letter.*, 3(3):2152–2159, 2001.

[4] Graimann, B., Allison, B., & Pfurtscheller, G. . Brain-computer interfaces: A gentle introduction. *In Brain-computer interfaces*, pages 1–27, 2009.

[5] Hussein, M. E. . 3d printed myoelectric prosthetic arm. *(Thesis Bachelor degree Engineering (Mechatronics)*, pages 1–87, 2014.

[6] Jiang, H., Zhang, T., Xiao, C., Li, J., & Guan, Y. Modular design of 7-dof cable-driven humanoid arms. *In International Conference on Intelligent Robotics and Applications*, pages 680–691, 2019, August.

[7] Kim, Y. J., Kim, J. I., & Jang, W. Quaternion joint: Dexterous 3-dof joint representing quaternion motion for high-speed safe interaction. *In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 935–942, 2018, October.

[8] Saduanov, B., Tokmurzina, D., Kunanbayev, K., & Abibullaev, B. . Design and optimization of a real-time asynchronous bci control strategy for robotic manipulator assistance. *In 2020 8th International Winter Conference on Brain-Computer Interface (BCI)*, pages 1–5, (2020, February).

[9] Al-Quraishi, M. S., Elamvazuthi, I., Daud, S. A., Parasuraman, S., & Borboni, A. Eeg-based control for upper and lower limb exoskeletons and prostheses: A systematic review. *Sensors*, 18(10):3342, 2018.

[10] Bhagat, N. A., Venkatakrishnan, A., Abibullaev, B., Artz, E. J., Yozbatiran, N., Blank, A. A., ... & Francisco, G. E. Design and optimization of an eeg-based brain machine interface (bmi) to an upper-limb exoskeleton for stroke survivors. *Frontiers in neuroscience*, 10(122), 2016.

[11] A.D. Biryukov. Development of software for contactless control of an anthropomorphic robot (doctoral dissertation, south ural state university). *dspace.susu.ru*, pages 1780–1785, 2017.

[12] Filimonov, S. A. . Development of a methodology for designing a robotic arm using application software packages. *Visnik of the Cherkasy State Technological University. Series: Technic Sciences*, 1:27–32, 2015.

[13] Folgheraiter, M., Jordan, M., Straube, S., Seeland, A., Kim, S. K., & Kirchner, E. A. Measuring the improvement of the interaction comfort of a wearable exoskeleton. *International Journal of Social Robotics*, 4(3):285–302, 2012.

[14] Frolov, A.A., Biryukova, E.V., Bobrov, P.D., Kurgan, M.E., Pavlova, O.G., Kondur, A.A., ... & Kotov, S. V. The effectiveness of complex neurorehabilitation of patients with post-stroke paresis of the hand using the brain-computer + exoskeleton neurointerface. *Almanac of Clinical Medicine*, 44(3), 2016.

[15] Fu, M. J., Daly, J. J., & Cavusoglu, M. C. Assessment of eeg event-related desynchronization in stroke survivors performing shoulder-elbow movements. *In Proceedings 2006 IEEE International Conference on Robotics and Automation*, pages 3158–3164, 2006, May.

[16] Guger, C., Allison, B. Z., Großwindhager, B., Prückl, R., Hintermüller, C., Kapeller, C., ... & Edlinger. How many people could use an ssvep bci? *Frontiers in neuroscience*, 6:169, 2012.

[17] Lew, E. Y., Chavarriaga, R., Silvoni, S., & Millán, J. D. R. . Single trial prediction of self-paced reaching directions from eeg signals. *Frontiers in neuroscience*, 8(222), 2014.

[18] Luu, T. P., Nakagome, S., He, Y., & Contreras-Vidal, J. L. Real-time eeg-based brain-computer interface to a virtual avatar enhances cortical involvement in human treadmill walking. *Scientific reports*, 7(1):1–12, 2017.

[19] Marakhin, E. Yu., & Belyaeva, A. S. Increasing the speed of the positional control of the movement of the robot arm. *PROBLEMS OF MODERN SCIENCE AND EDUCATION*, 29, 2015.

[20] Nurtay, B., Suranshy, T., & Folgheraiter, M. Development of a semi-rigid tendon actuated limb for robotics applications. *In 2020 5th International Conference on Robotics and Automation Engineering (ICRAE)*, 5:31–35, 2020.

[21] Z. Otarbay, I. Assylgali, A. Yskak, and M. Folgheraiter. Development of a teach pendant for humanoid robotics with cartesian and joint-space control modalities. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–6, 2019.

[22] Popov, L.A. . Anthropomorphic robot by means of 3d printing. *In COLLECTION OF ARTICLES FORTY-SEVENTH INTERNATIONAL SCIENTIFIC CONFERENCE "TECHNOKONGRESS"*, 217:14, 2019, October 07.

[23] S. S. Ryumkin. Adaptive control of a robotic arm. *LBC 32.965 AND 74 EDITORIAL BOARD*, page 30, 2018.

[24] Shcherbak, D. Yu., Ignatiev, V. V., Leshchenko, S. A., & Ignatieva, A. S. Solution of the problem of control of a robot-manipulator with the help of fuzzy logic. *Computer Science, Computer Science and Engineering Education*, 1:41–53, 2018.

[25] Stienen, A. H., Hekman, E. E., Van Der Helm, F. C., & Van Der Kooij, H. . Self-aligning exoskeleton axes through decoupling of joint rotations and translations. *IEEE Transactions on Robotics*, 25(3):628–633, 2009.

[26] Tursynbek, I., & Shintemirov, A. . Infinite torsional motion generation of a spherical parallel manipulator with coaxial input axes. *In 2020 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 1780–1785, 2020, July.

[27] Tursynbek, I., Niyetkaliye, A., & Shintemirov, A. . Computation of unique kinematic solutions of a spherical parallel manipulator with coaxial input shafts. *In 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 1524–1531, 2020, December.

[28] Vorobiev, A. A., Petrukhin, A. V., Zasypkina, O. A., & Krivonozhkina, P. S. Exoskeleton - new opportunities for abilitation and rehabilitation (analytical review). *Questions of reconstructive and plastic surgery*, 2(53), 2015.

[29] Zakirov, R.I., Aliev, M.I., & Morozov, A.I. . Determination of kinematic characteristics of a delta-robot by given parameters of the working area. *Electrical and information complexes and systems*, 14:4, 2018.

[30] E. D. Zavyalov. Development of a robotic platform for ultrasonic testing of welded seams: master's thesis in the field of training: 15.04. *Mechatronics and Robotics*, 15(4), 2018.

[31] Zavyalova, T.V. The problem of optimal stabilization of the speed of the angle of rotation of the robotic arm. *In Mathematical Modeling and Information Technologies*, pages 123–128, 2016.