VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Ha NGUYEN

A FEEDER ROBOT ARM FOR AIDING THE ELDER PEOPLE WITH MEALS

Information Technology – Embedded Systems Engineering

2018

ACKNOWLEDGEMENTs

Firstly, I would like to give my special thanks to my supervisor at Waseda University, Professor Zhang Cheng. He was a very dedicated and responsible teacher and inspirer who gave me lots of valuable advice as well as experience and motivation for my work. He was also a great support for my life in Japan during my stay, which may have been really hard if lacking support. Without his help and work this thesis would not be possible, and I would never have had the chance to work in such a phenomenon environment.

I would like to thank Mr. Santiago Chavez, my thesis supervisor at Vaasa University of Applied Science for his dedicated encouragement and technical support. He has always been helping me in the completion of this thesis work and available any time I need help for the work.

Special thanks to Kobo Laboratory of Waseda University for providing me with a great range of resources and support for my research here. The project would never have flourished as much without the school's great support both technically and financially. I appreciate greatly how they have treated me even though I am not one of the students.

My appreciation goes to my colleague, friends and family who have supported me both technically and emotionally through the whole working process.

VAASAN AMMATTIKORKEAKOULU

UNIVERSITY OF APPLIED SCIENCES

Degree Program in Information Technology

ABSTRACT

Author	Ha Nguyen
Title	Feeder Robot for aiding elder people in meals
Year	2018
Language	English
Pages	59
Name of Supervisor	Chavez Santiago

Japan’s aging rate is considered extremely high, outweighing all other nations in the world. On the other hand, the robotics and automation industry have received great support and encouragement for growth in every field possible including eldercare. However, we find several common disadvantages in every system, including an exceedingly high cost and lack of flexibility. In order to give a solution to these drawbacks, we aim at designing a cost-efficient robot arm for helping in mealtime with an efficient control system, flexibility in usage and embedded vision systems with big data analysis on the cloud.

Keywords robot, kinematics, c++, MATLAB, SolidWorks, 3D printing

CONTENTS

LIST OF FIGURES AND TABLES

ABBREVIATIONS

{*ABC*}	A frame with the name "ABC"
{B}	Base Frame
{G}	Goal Frame
{S}	Station Frame/Space Frame
{T}	Tool Frame
{W}	Wrist Frame
2D	Two Dimensions
3D	Three Dimensions
3R, R-R-R	3 Revolute Joints
API	Application Programming Interface
Atan2	two-argument arctangent
$\mathbf{c_{123}}$	$\cos(\Theta 1 + \Theta 2 + \Theta 3)$
$\mathbf{s_{123}}$	$\sin(\Theta 1 + \Theta 2 + \Theta 3)$
$\mathbf{c_i, c\theta_i, \cos\theta_i}$	cosine of θ_i
C-space	Configuration Space
DC	Direct Current
DH	Devanit-Hartenberg
DOF	Degrees of Freedom
DXL	Dynamixel
H/HB	Highest Byte
L/LB	Lowest Byte

IDE	Integrated Development Environment
OS	Operating Systems
R	Revolute Joint
RPM	Rounds per Minute
SCARA	Selective Compliance Articulated Robot Arms
$\mathbf{s_i}$, $\mathbf{s\theta_i}$, $\mathbf{sin\theta_i}$	sine of θ_i
SDK	Software Development Kit
TTL	Transistor-Transistor Logic

1. INTRODUCTION

1.1. Waseda University

Waseda University is a Japanese private research university in Shinjuku, Tokyo. With contributions and achievements spanning across Japanese education history since 1882, the institute has consistently ranks among the most academically selective and prestigious universities in Japanese as well as worldwide ranking, also being the best private university in Japan.

Waseda University currently ranks at 208th in the QS World University Rankings, 97th in Engineering and Technology section. With thirty-six departments, as of 2018 Waseda currently has 46,301 students in 13 undergraduate and 21 postgraduate schools. "It was founded on three principles: academic independence, practical innovation and enlightened citizenship. Waseda's mission is to build leaders, and it does, producing seven prime ministers and countless other leaders in government, business, journalism, science, literature and arts. Waseda is Japan's most global campus, with over 5000 international students from 100 countries, and partnerships with over 600 prominent institutions in 84 countries."

1.1.1. Kobo and the Graduate Program for Embodiment Informatics

Kobo laboratory's concept is centered around the idea of a 'workshop' study space. Research students belonging to different departments, different laboratories and professors come together in one common space considered as their shared 'workshop', which is independent from their academic laboratory and professor. This act as a very effective academic stimulant, motivating students to devote themselves outside the scope of their original studies, meet and work with other students and promote interdisciplinary studies.

Embodiment Informatics is the main program behind the laboratory. It is described as being an academic field which integrates together the outer frame of embodiment with the information inside to provide valuable application benefits in multiple field. The purpose of this type of collaboration is to create composite value from the benefits of various technologies. It is favorable for students in this program to acquire a broad range of engineering knowledge, namely basic mechanics subjects for informatics graduates, and basic informatics knowledge for mechanics graduates.

1.2. Application Description

1.2.1. Background Information and Goals

Japan’s aging rate is considered extremely high, outweighing all other nations in the world. Furthermore, it is now increasing rapidly and is predicted to continue this trend for many years to come. The situation has been putting a huge stress not only on the labor-population financially but also on the healthcare industry. By now, people aged 65 and older in Japan make up a quarter of its total population, even reaching one third by 2050 according to prediction. Taking care of senior citizens has been in a dire status due to their children who are part of the working-age population as well as to nursing homes with serious lacking in medical staffs, especially nurses. This is detrimental to the living quality and the need for independence of the elderly specifically as well as the whole population in general.

On the other hand, the robotics and automation industry have received great support and encouragement for growth in every field possible including eldercare. The existing of robot arm for assistive eating is not brand new on the market. However, we find several common disadvantages in every system, including an exceedingly high cost and lack of flexibility. Firstly, the high price for owning a device is making it hard for the popularization of eating robot assistant in household. Secondly, all of them are designed according to people who are right handed, which creates difficulty and discomfort to the left-handed.

In order to give a solution to these drawbacks, we aim at designing a cost-efficient robot arm for helping in mealtime with an efficient control system, flexibility in usage and embedded vision systems with big data analysis on the cloud. For dealing with liquid food, the spoon shape is made with similar design with a soup spoon to be able to hold liquid; furthermore, the spoon travel trajectory is designed to that the spoon position is made stable at all time, avoiding spilling. On the arm is an attached camera with vision system for taking information on the remaining food and upload them to the cloud for analysis and taking care of the user’s nutrition. Finally, as most of the systems now is made according to the usage of right-handed people, we made the systems with two modes for both left-handed and right-handed people to create a feeling of comfort for every user.

Finally, all of the mentioned are just fully automated devices without any data usage and therefore cannot get feedback on the eating of the users for the medical staffs to adjust accordingly.

1.2.2. General Description

This thesis cover the projects from the initial design, assembly and programming. Even though the final goal of the whole project was to produce a marketable feeding arm, the given tasks were only to consider several prototypes and API development for future project worker to further improve on.

In more details, the thesis work covers from creating several 3D prototype in SolidWorks while adjusting to what is needed to be made. Afterward, the tasks were to find suitable component to go with the arm, print it out and assembly each versions. Finally, it was necessary to make a fully functional API with the most useful and necessary functions.

1.2.3. Functions

By the time the thesis work period has ended, the program is running in Windows using Visual Studio. The API functions of the arm include:

- Moving and monitoring
- Calculation of inverse kinematics
- Initializing and control the motor
- Adjustment of speed

1.2.4 Similar projects

At the time of development, there were a number of projects readily available on the market with similar target and constructions. Below I will discuss briefly the inspirations and improvement that are aimed to be improved in the final project:

- **Obi**

Obi (as shown in Figure 1) is a flexible and safe robotic arm that is made to aid with eating for the disable. It has 5 DOFs, which is one thing that the team at the university decide to adopt later on because of its flexibility. Obi also comes with multiple method of activation with different types of input buttons. But like other products that we look at, it totally lacks of data usage and has a very high price, even with the rental option.

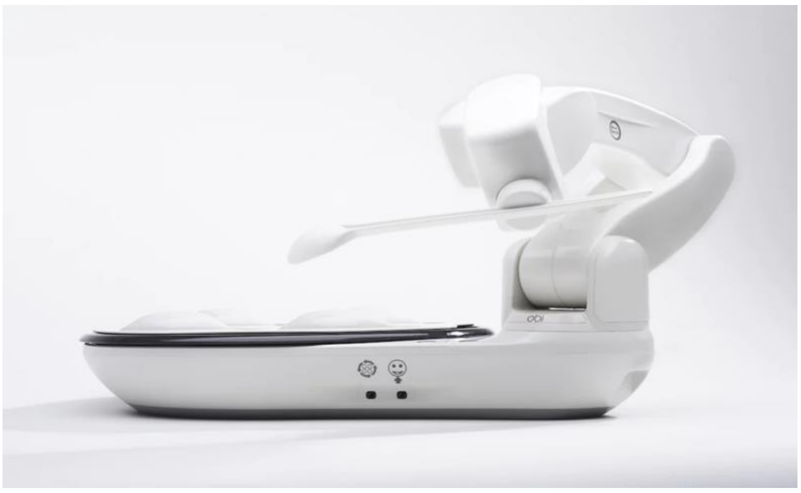


Figure 1 The robot arm Obi

- **Bestic**

Bestic (Figure 2) is a very simple and intuitive feeding device aimed at assisting disabled people who are not capable of feeding themselves. With 4 DOFs, this arm was the primary inspiration for the project at first and during the thesis working period, but it was later decided to go for a robot arm with more flexibility and naturally in its movement. Other than the movement, Bestic always lacks in term of functions and adjustability.

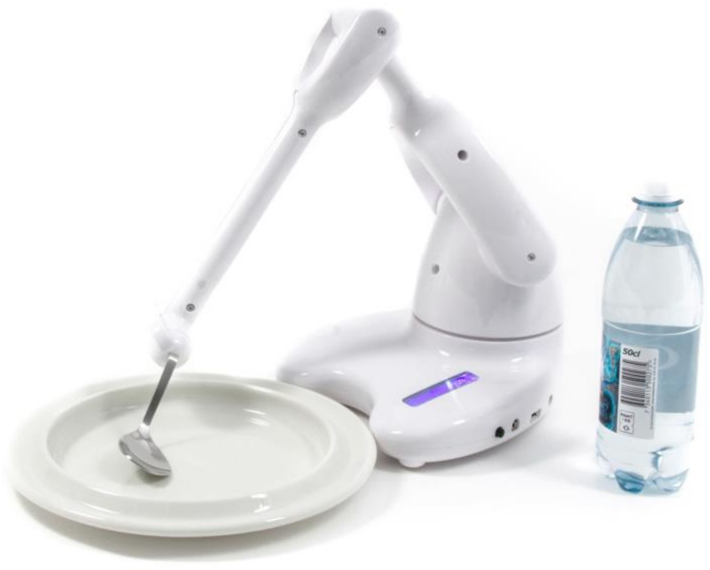


Figure 2 The robot arm Bestic

- **MySpoon**

MySpoon (Figure 3) is a product developed in Japan and is considered to be the best possible design. The only thing considered to be a disadvantage would be the stick that is used to control the arm, which could be hard to use for a number of target customers. Later on, in the project, it was decided to go for a Selective Compliance Articulated Robot Arm (SCARA) design similar to this product.

Overall, the design is very flexible and efficient. Along with what the team sees as advantages of this project, we want to improve the design and also add data-functionality.

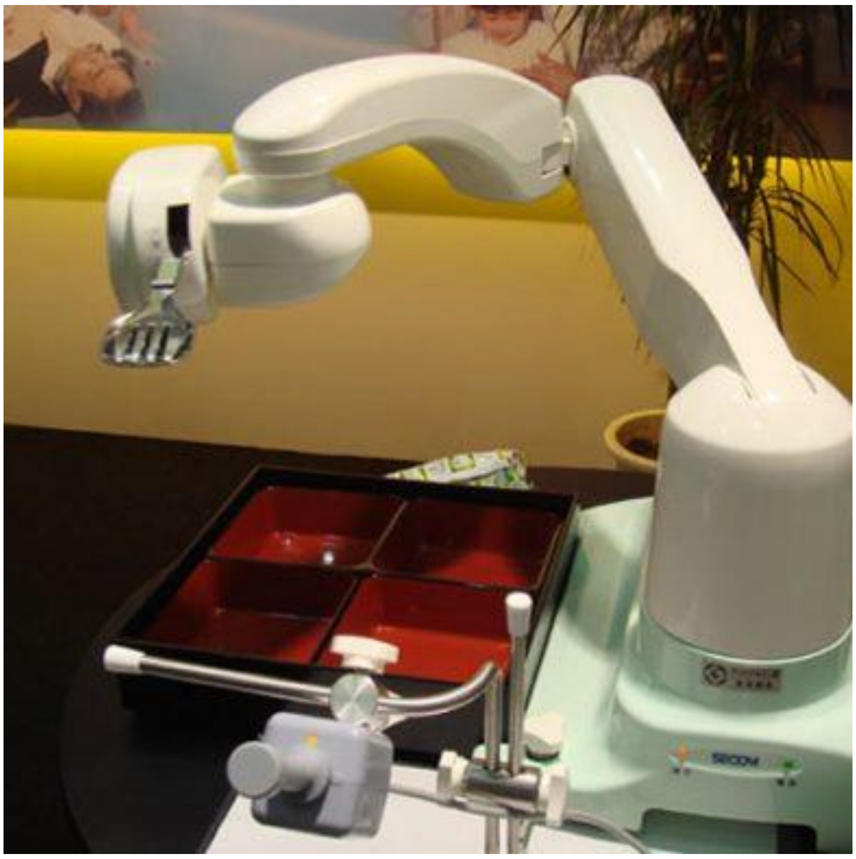


Figure 3 The robot arm MySpoon

1.3 Structure of this thesis

The rest of this thesis begins from Chapter 2, which discusses the technical knowledge base needed for the development of this project. From Chapter 3 to Chapter 7, the process of creating the prototypes is discussed, from design and assembly to calculation, operating and programming for control (all sorted relatively in term of time domain). Chapter 3 illustrates the design process of multiple production prototypes. Chapter 4 discusses about the calculation behind the arm and also the kinematics model. Chapter 5 is about the software implementation, namely the functions that the API includes, how they work and what is their purpose coming from the calculations of the previous chapter. Chapter 6 discusses the logging that comes with the software as well as possible error that may occur. The last chapter, Chapter 7, concludes the thesis.

2. TECHNICAL BACKGROUND

2.1. Robotics

2.1.1. Mathematical Prerequisites

For calculation of the arm’s kinematics, there must be adequate mathematical abilities in both geometric and algebraic field. In general, mathematical prerequisites (but not limited to, and will not be discussed in this paper) may include:

- Pythagorean, cosine rules, geometric identities
- 2D and 3D spaces
- Planes, vectors including transformation of planes and vectors
- Rotation matrixes, identity matrix, skew-symmetrical matrix
- Linear algebra matrix operations, including dot product, cross product, transpose, multiplication, addition, and subtraction

It is also required other than elementary algebra that one is familiar with **atan2**, which is a function that will be used mainly in this work in instead of arctangent. Atan2(x, y) is a function which takes in two argument y and x and gives the result as an angle in the Euclidean plane that goes between the positive x-axis and the line pointing from the origin point to point (x, y).

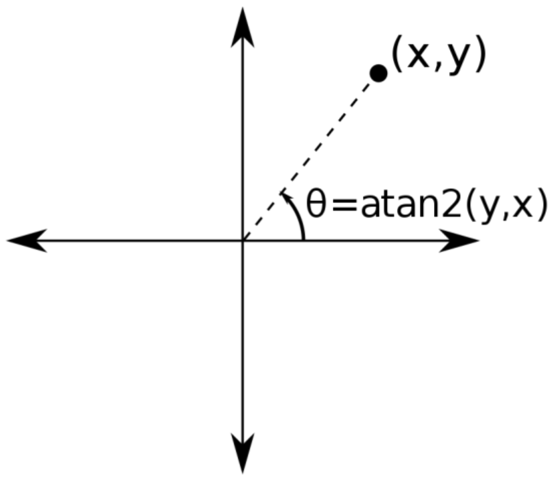


Figure 4 atan2 function

It is different from the atan as the angles given are signed. The counter clockwise angles will give positive values while clockwise will gives negative value. This is very similar to human's convention of giving the angle in the plane and eliminating two main problems when working with atan: tangent returns no value (which results in data error for calculation) with the angle 90° and -90 and gives the same value for the angles that are 180° apart from each other. This is however, not the case with atan2.

2.1.2. Commonly used conventions and concepts

It is useful before considering the control operation to be familiar with basic robotics conventions. In this thesis research the system is a robot manipulator constructed using rigid links. Throughout this paper, though most of the concepts can be applied to nearly any standard robot systems, I would like to look at cases of robot arms or manipulator specifically and may use these terms interchangeably.

A manipulator is a mechanical system made of a set of **links** connected by many types of **joints**, with **actuators** like stepper motors or DC motors which create forces or torques causing the robot arm's links to move. The systems usually end with an **end-effector** which may be a tool, a gripper or a hand for grasping, manipulating other objects in the working environment.

The most important configurations are the ones regarding the positions of the arm, given by specifications of the positions of all points of the robot. That would require us an arbitrary number of information needed. However, because the robot's links are rigid and nearly 100% of the cases of a known shape, only a few numbers are needed to represent its state in the coordinate system. The **degree of freedom (DOF)** of the robot is the minimum number of independent parameters that defines its configuration. There is also an n-dimensional space called the **configuration space (C-space)** which is a space that contains all possible configuration of a manipulator. The configuration of the manipulator at a point in time is displayed as a point in its C-space.

This system, like most other robot manipulator, is a **spatial rigid body**, meaning a rigid body moving freely in a three-dimensional space, unlike a **planar rigid body** which moves in a planar workspace. A spatial rigid body has six degrees of freedom by default minus the number of independent constraints. With this agreed, a robot arm utilizing the most of its workspace which is limited by the length of its link should consists of at least six degrees of freedom (which most advanced manipulators have). We have a general convention for calculating degrees of freedom as bellow:

degrees of freedom = (sum of freedoms of the bodies) – (number of independent constraints)

The number of independent constraints relies heavily on the number of joints in the systems as well as the types of joints. The study of joints as well as the conventions for calculating the DOF using the **Gruebler's equation** however are outside the scope of this thesis due to its complications. The studied systems in consideration only consists of **Revolute Joint (R)**, also called hinge joint, which has one degree of freedom (Figure 5). The studied arm with four R's therefore has four degrees of freedom, all of which are revolute. From here onward all movement and control discussed will be concerning the revolute joints only.

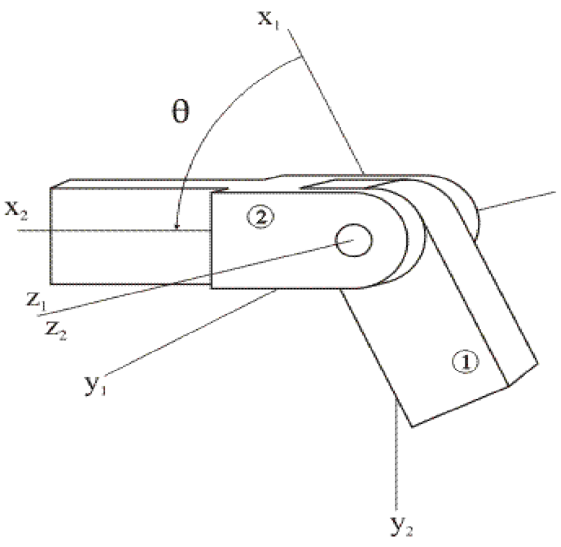


Figure 5 A revolute joint

For a robot with *n* DOF's, we use **explicit parametrization** which consists of representing a *n*-dimensional space it in *n* coordinates, the minimum necessary will be used.

2.1.3. Spatial Descriptions and Transformation

For any 3D physical space, we would need six numbers in three pairs at minimum to describe exactly the position and orientation of a rigid body. For calculation however, we describe position by attaching several reference frames to the body relatively to a common fixed frame. The configuration of this frame is then presented relatively as a 4x4 matrix.

The operations needed to be applied to these frames are (1) translate and/or rotate a vector of a frame, and (2) change the specification of one vector or frame from one coordinate systems to coordinate systems in another frame. All operations are made by linear algebra operation on matrices.

The allocation of axes in this work strictly follows the **Right-Hand** convention which specifies the positive movement.

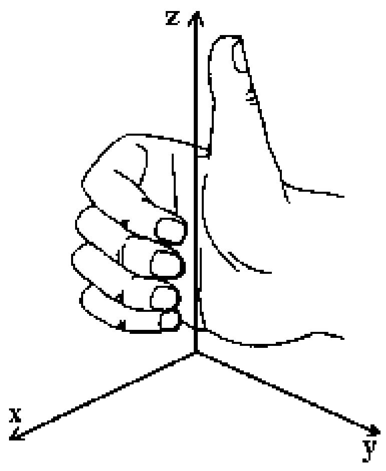


Figure 6 Right Hand Rule

A **spatial description** consists of two elements: a description of a position and a description of an orientation. Here we use the convention where we describe with a 3x1 **position vector** and a 3x3 **rotation matrix**.

a) Description of a position

As mentioned above, in a specific coordinate system, we are able to locate any point of the universe using a 3x1 position vector. It is common to work in multiple coordinate systems for a series of frames, we almost always add the the name of the frame in consideration to the vectors. Our convention is: vectors' names are written in capital letters, with a *leading* superscript indicating the coordinate system to which they are referenced. For example, we have the vector ${}^{A}P$; this means that this vector P in frame A, or explicitly speaking the components of ${}^{A}P$ are distances along the axes of frame {A}. Then, the elements of the vector can be given as below, with the name of the axes in following subscript:

$$ {}^{A}P = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \quad (0) $$

b) Description of an orientation

There are three major uses for a rotation matrix R: to represent an orientation, to change the reference frame in which a vector or frame is represented or to rotate a vector or a frame. <2> The first case R is used as a representation while the latter two uses R as an operator. Notation-wise we use R_a to refer to the orientation of frame {a} in relative to {s}. For specifying the orientation of {a} in relative to {b}, we use R_{ba}.

The steps of identifying the orientation of a body is as follows: we attach a coordinate system to the body, which for convenience in default position has a special relationship to the reference system (namely perpendicular, parallel, 45° or 135° apart); next, we give the description this system relative to the reference.

c) Frames

A frame is a coordinate system and can serve as a reference system, within which to express the position and orientation of a body. Therefore, frame-related operation (Figure 7) is crucial as we consider the transformation or changing of description of the attribute of a body from one frame to another.

A frame consists of four vectors giving position and orientation information, equivalent to a combination of a position vector and a rotation matrix, both of which are described above.

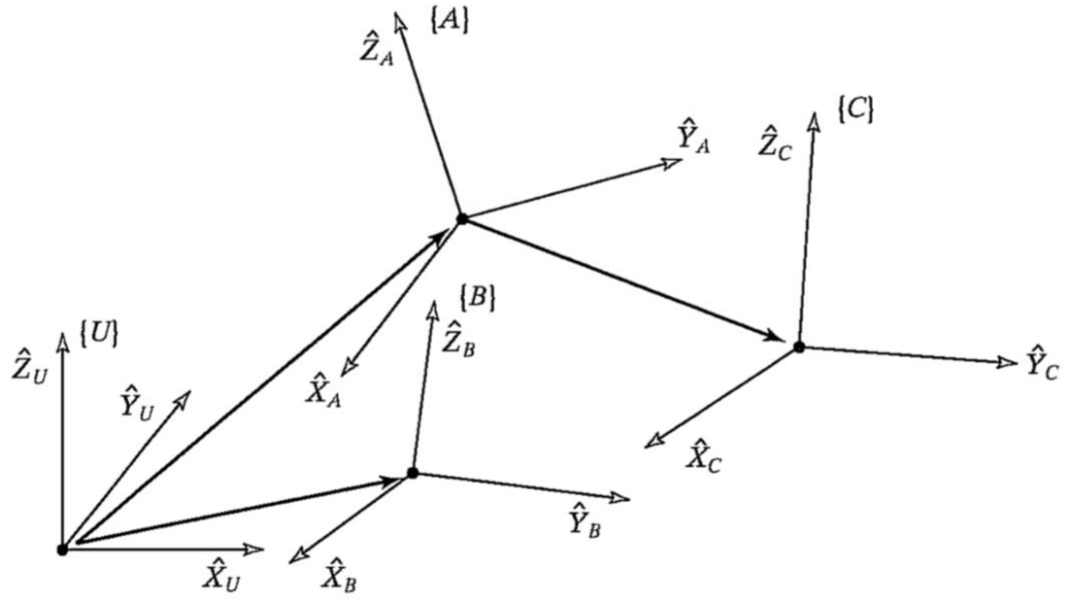


Figure 7 Examples of different frames and distances

2.1.4. Link-Connection Description

Not considering the special case in which one joint may comprise of more than one degree of freedom, for a chain consisting of n joints, the links and also the reference frames will be numbered sequentially from 0 to n (along with special frame name like base, wrist): the ground link is labeled 0, and the end-effector frame is attached to link n.

Joint axes are considered lines in space. With *i* as the number of the joint axis, then the joint axis is a line or vector direction, about which link *i* rotates relatively to link *i-1*. For any two axes in 3D space, the distance between them is measured along mutually perpendicular line. As in Figure 8 bellow, the link length is a_{i-1}. Other than distance, we also have a link twist, which define the angular relative location of the two axes. Imagine a plane whose normal is the perpendicular line just constructed, we can project the axis *i-1* and I onto the plane, and the link twist is the angle between them. Using the right-hand rule, this angle is measured between the two projections around a_{i-1}. The twist of link *i-1* is defined as α_{i-1}. The idea of using just two parameters which are the length and twist, the relationship between any two joints can be defined in space.

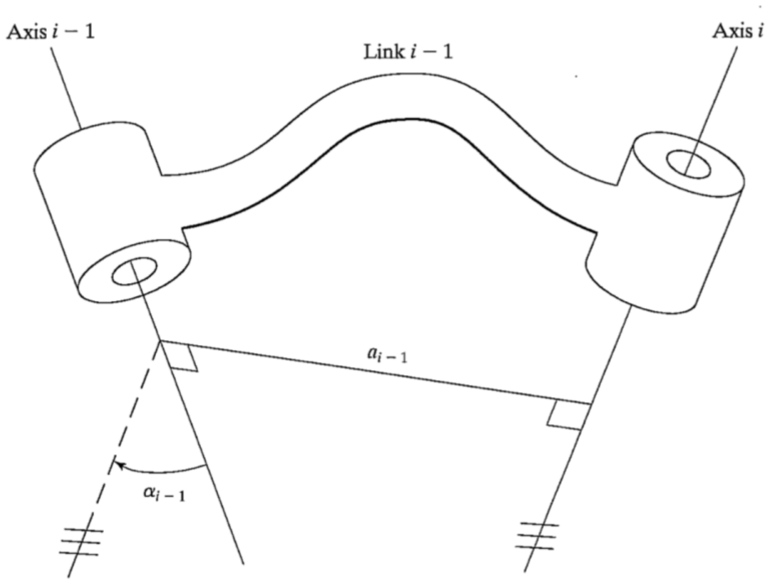


Figure 8 Link connection description

For now, ignoring all the other factors such as load, gearing etc.…, for kinematics investigation, we only worry about two quantities that define the links' relationship to each other: **link offset** and **joint angle**.

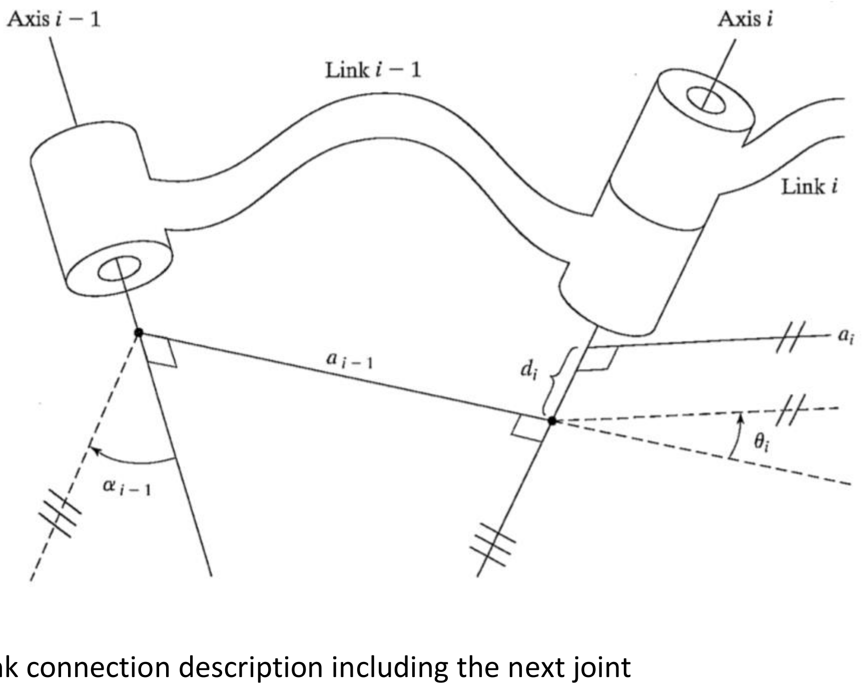


Figure 9 Link connection description including the next joint

Intermediate neighboring links have a common joint axis between them. The link offset is the distance along this common axis while joint angle is the amount of rotation about this common axis. For joint axis i the offset is d_i and joint angle θ_i.

By defining the relationship between the links, the robot can be described kinematically by giving the values of fours quantities for each link, two of which describe the link itself while the other two describe the links' connection to a neighboring link. Since all of our joint are revolute, θ_i is called **joint variable**, when the other three numbers are fixed link **parameters**. The convention of definition for these quantities that is being used here is called the Denavit-Hartenberg notation, which will be discussed below.

2.1.5. Denavit-Hartenberg

Denavit-Hartenberg (Figure 10) is the convention of choice for describing the open chains for forward kinematics. The underlying idea is to attach reference frames to each link in the open chain and from the knowledge about relative displacement between these frames, namely from joint axis to joint axes, derive the forward kinematics model. assuming that there is a fixed reference frame.

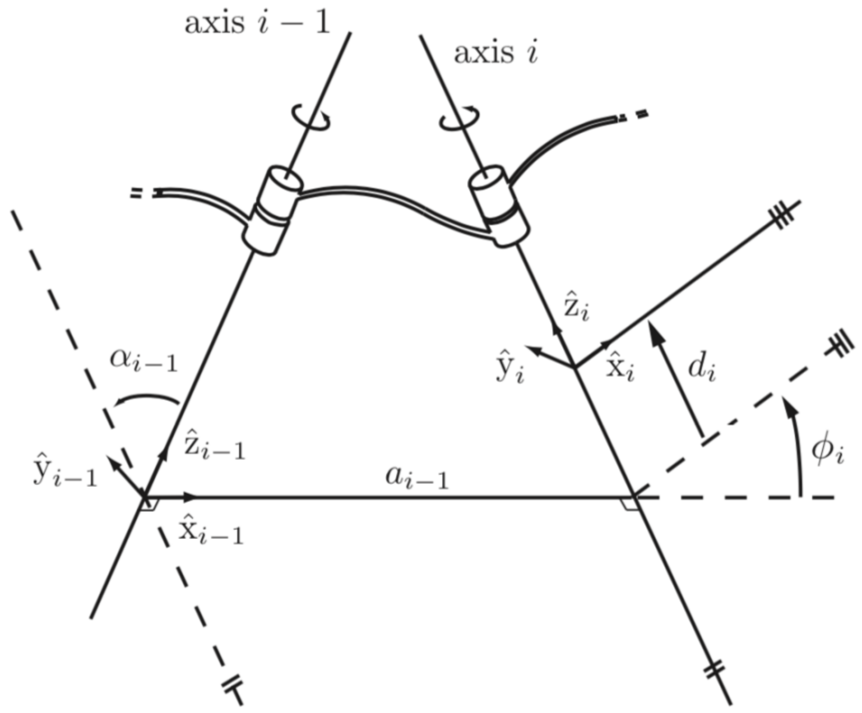


Figure 10 Devanit-Hartenberge convention

Rather than assigning parameters arbitrarily, the Denavit-Hartenberg convention sets the rules. Going from the first to last link, we give each of them a set of four parameters, which has been discussed before:

- Link length: length of the mutually perpendicular line, denoted by scalar ai-1
- Link twist: α_{i-1}, the angle from z_{i-1} to z_i, measured about x_{i-1}
- Link offset: di, offset from the intersection of the link-i frame (positive direction is defined to be along the z_i axis)
- Joint angle: ϕ_i, the angle from x_{i-1} to x_i, measured about the z_i axis

There are also two special cases where the mutually perpendicular line is undefined or not unique, those being when the adjacent revolute joint axes intersect or are parallel. Both of the two cases are met in the project. When these axes intersect, the link length is 0, x_{i-1} is perpendicular to the plane spanned by z_{i-1} and z_i. For when the axes are parallel,

we try to choose a mutually perpendicular line which is the most convenient for calculation while leading to as many negative values as possible.

2.1.6. Forward Kinematics

Forward kinematics consist of calculating the end-effector from all joint coordinates θ. The aim first of all is to make a kinematic model which relates the {T} frame to frame {B}. The first step includes making a table for link parameters including the link length, twist, offset, angle. For n number of links, we have n-1 row in the table, and n-1 transformation matrixes. The general form of a transformation matrix is as follow:

$$ {}_{i}^{i-t}T = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i \cdot \alpha_{i-1} & c\theta_i \cdot c\alpha_{i-1} & -s\alpha_{i-1} & -s\alpha_{i-1} \cdot d_i \\ s\theta_i \cdot s\alpha_{i-1} & c\theta_i \cdot s\alpha_{i-1} & c\alpha_{i-1} & c\alpha_{i-1} \cdot d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} (1) $$

After the link frames have been defined with link parameters found, we continue to make the kinematic equations by multiplying individual link-transformation matrices together in the following concatenation:

$${}^{0}_{N}T = \ {}^{0}_{1}T \ \ {}^{1}_{2}T \ \ {}^{2}_{3}T \dots \ {}^{N-1}_{N}T (2)$$

In this experiment, the forward kinematics is used as a test method for identifying problems with the model rather than a product-oriented use. Though it can be used to calculate inverse kinematics, another method was chosen to do the calculation.

2.1.7. Inverse Kinematics

Contrary to forward kinematics, which involve getting the coordination of the end-effector by doing calculation on the given value in units of degree from motors, inverse kinematics is considerably a more difficult but useful problem: from the desired position and orientation of the end-effector (in some cases, also the position of other joints in case of obstacle), compute the joint angles to achieve this result.

There are two type of solutions, **closed-form** and **numerical**, of which closed-form was chosen. For most of the cases, a closed-form solution is much more desirable as the numerical methods take too much computer resources and make it almost impossible for real-time application. Among the **closed-form** solution, there are two methods: **algebraic** and **geometric**. The distinction between these two are not crystal clear, but for our case, an algebraic solution may involve using the existing kinematic model from forward kinematics formulation, while geometric involves utilizing the geometric relationship between the links.

A popular issue that usually occurs during calculation is multi-solution. Like the image following, with one (x,y) value, we may have two different solution for the angle leading up to such position. If such a case occurs (for example in Figure 11), we chose the solution that has the smallest degree of the smallest indexed joint (in this case, theta 1).

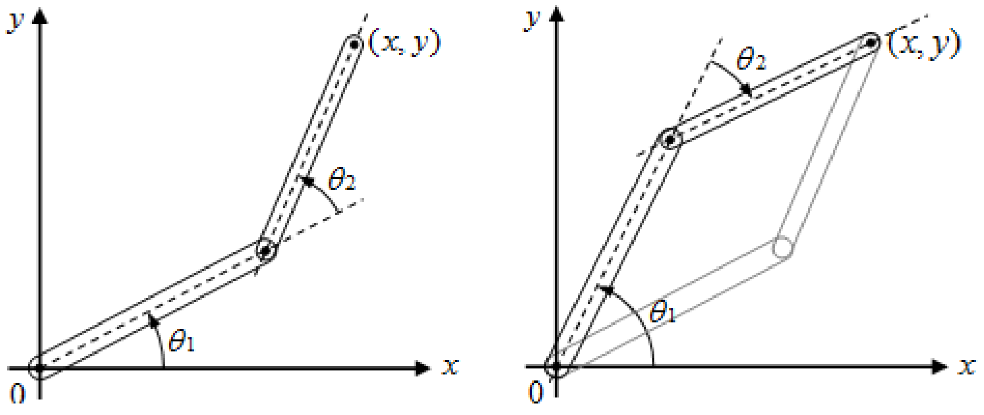


Figure 11 Multiple solutions

2.2. Dynamixel SDK

Because all of the motors used in this project are from ROBOTIS's actuator series Dynamixel, we were using the company's tools also. Dynamixel SDK is a software development kit that provides Dynamixel control functions using packet communication. The SDK supports a variety of programming language and can be developed natively on all three major computer operating systems.

2.3. Relevant Technologies and Hardware

C++

C++ was the programming language of choice. The development kit made for the actuators is written natively in C, C++ and Python, and C++ was chosen due to its efficiency and availability of needed functions and support.

Solid Works 2017

All 3D printed parts in the project were made using Solid Works 2017 (Figure 12), a solid modelling computer-aided design and computer-aided engineering software.

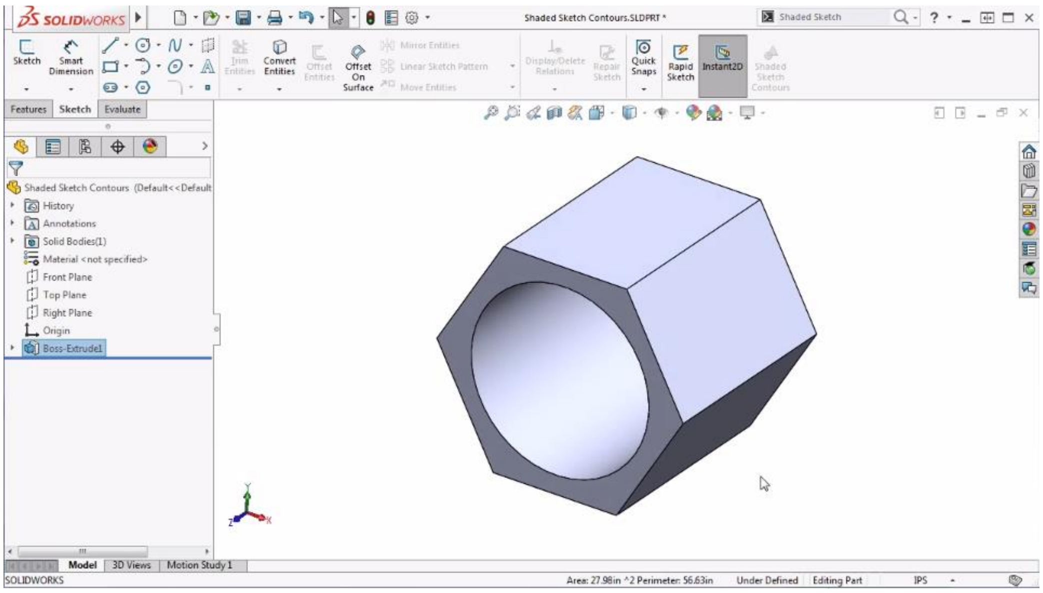


Figure 12 SolidWorks 2017

Visual Studio Community 2017

Visual Studio Community (Figure 13) is a free, fully-featured, and extensible IDE for creating modern developer apps for multiple operating systems. The software aims at students, academics, individual developers, open-source projects and small non-enterprise team. The whole project was developed in Visual Studio Community 2017 for designing the algorithms and for running on Windows.

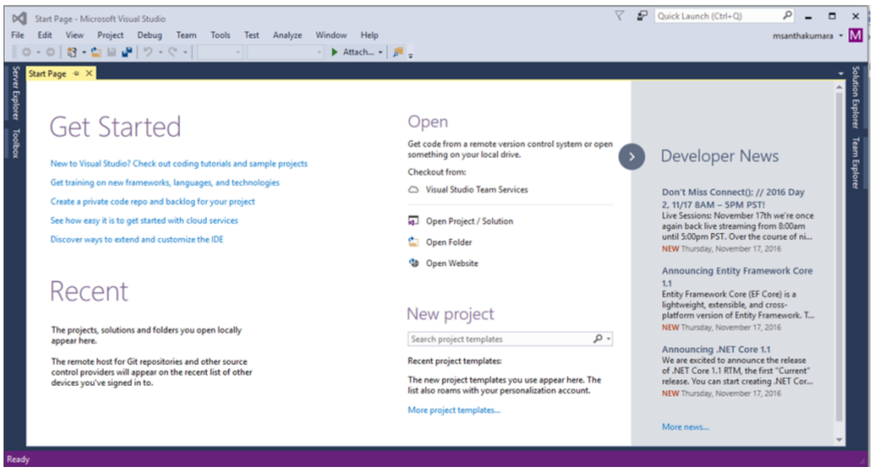
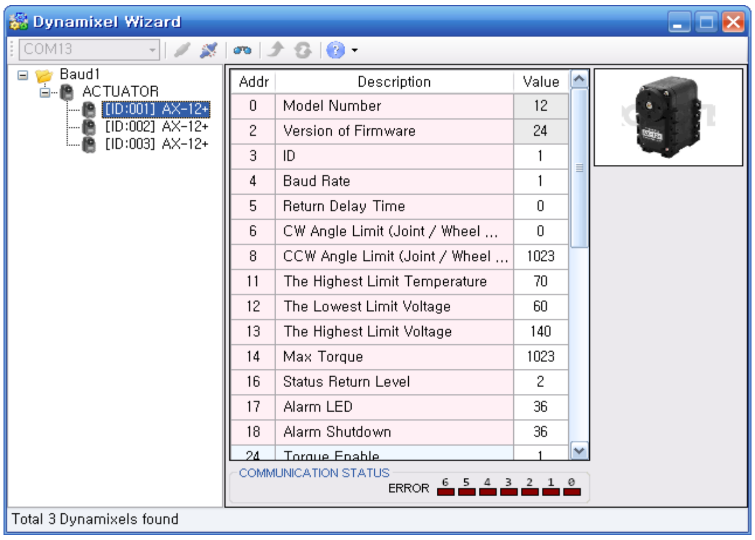


Figure 13 Visual Studio Community 2017

RoboPlus

RoboPlus is a Windows suite of software that allows for easily interacting with all ROBOTIS hardware, which are the actuators in this project. The set consists of five programs in total, but we only touch on RoboPlus Manager (Figure 15) and Dynamixel Wizard (Figure 14) in this project.



Addr	Description	Value
0	Model Number	12
2	Version of Firmware	24
3	ID	1
4	Baud Rate	1
5	Return Delay Time	0
6	CW Angle Limit (Joint / Wheel ...	0
8	CCW Angle Limit (Joint / Wheel ...	1023
11	The Highest Limit Temperature	70
12	The Lowest Limit Voltage	60
13	The Highest Limit Voltage	140
14	Max Torque	1023
16	Status Return Level	2
17	Alarm LED	36
18	Alarm Shutdown	36
24	Torque Enable	1

Figure 14 Dynamixel Wizard

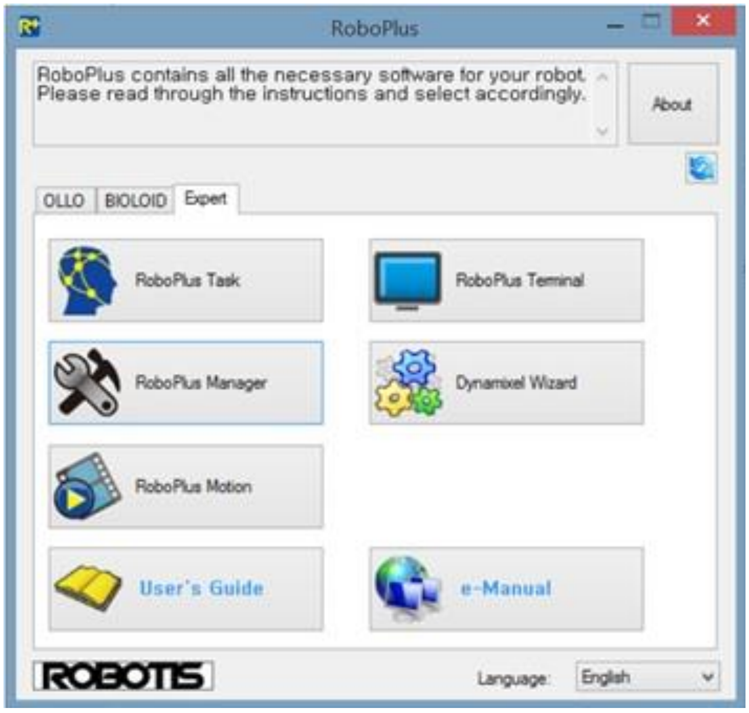


Figure 15 RoboPlus 1.0

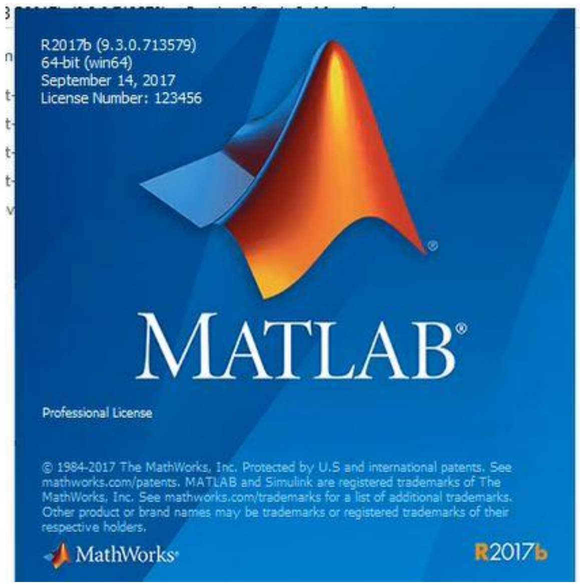
KEYENCE AGILISTA-3100

All 3D-printed parts in the project were made using the KEYGENCE AGILISTA-3100 3D printer (Figure 16). The 3100 is an industrial 3D printer, with Inkjet system and photopolymerization making it extremely fast and precise. Furthermore, the support material is water soluble, making it easier for the making of each part.

Figure 16 KEYGENCE AGILISTA-3100 3D printer

MATLAB 2017b

MATLAB (Figure 17) is a software environment for numerical computation with its own proprietary programming language made by MathWorks. Within the scope of this project, MATLAB was used mostly for calculating the kinematics model for checking with parameters.


R2017b (9.3.0.713579)
64-bit (win64)
September 14, 2017
License Number: 123456

Professional License

© 1984-2017 The MathWorks, Inc. Protected by U.S and international patents. See mathworks.com/patents. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Figure 17 MATLAB 2017b

Dynamixel Hardware

For motors, a selection of three servo motor models were used, all of which are from the DYNAMIXEL series from ROBOTIS, a Korean robotics company. They are smart actuator which are developed to be daisy chained joints on a robot or mechanical structure. Each servo module actuator is a full package which includes: a fully integrated DC Motor, reduction gearhead, controller, driver and network functions. They are programmable and networkable, having all the data and status sent and received through data packet stream. A wide selection of motors is made with different price points, functionalities, maximum stall torque (as well as unofficial loading torque) and maximum speed. For this project, we chose the three models (in order from weakest to strongest): AX-12A, MX-28AT and MX-64AT. All of them are connected in a daisy chain through TTL communication.

Figure 18 Dynamixel MX-64T

Figure 19 Dynamixel MX-28AT

Figure 20 Dynamixel AX-12A

For programming and writing program to the motors directly from PC, USB2Dynamixel device must be used and connected through the 3P connectors. After switching to TTL mode and connect to the first motor, we can communicate with the motor freely either from programming or RoboPlus.

An external power source is also connected to the USB2Dynamixel to power the actuators with a small accessory called SMPS2Dynamixel, which is connected to USB2Dynamixel through 3P connector.

Figure 21 USB2Dynamixel

Figure 22 SMPS2Dynamixel

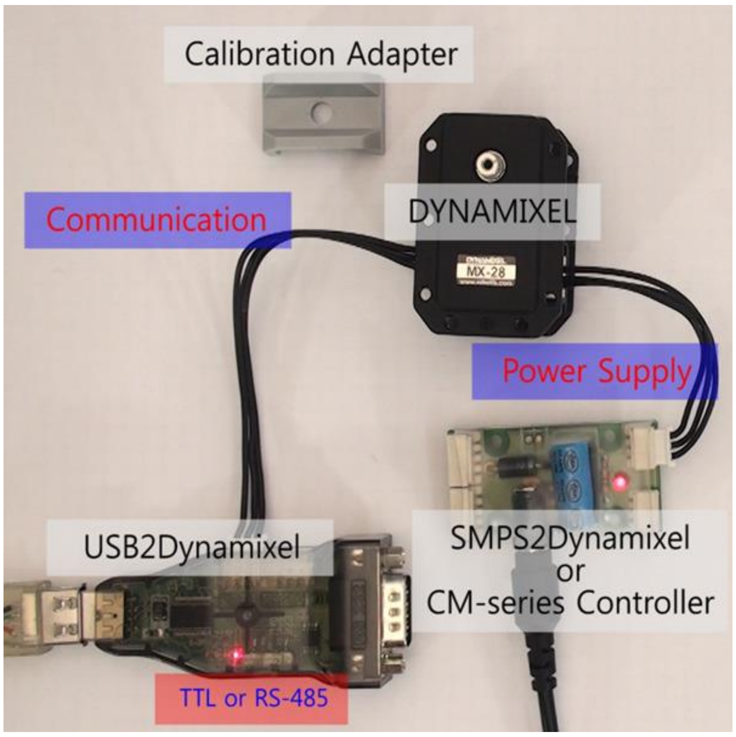


Figure 23 Device connection

3. 3D DESIGN AND ASSEMBLY

3.1. First prototype

The first prototype was made with four DXL motors: one MX-64T, one MX-28 and two AX-12A. All design was self-designed in Solid Works 2017/2018 and printed using Keyence AGILISTA 3D printer with silicone as the printing material. The design of each parts is as bellow in Figure 24 to 26:

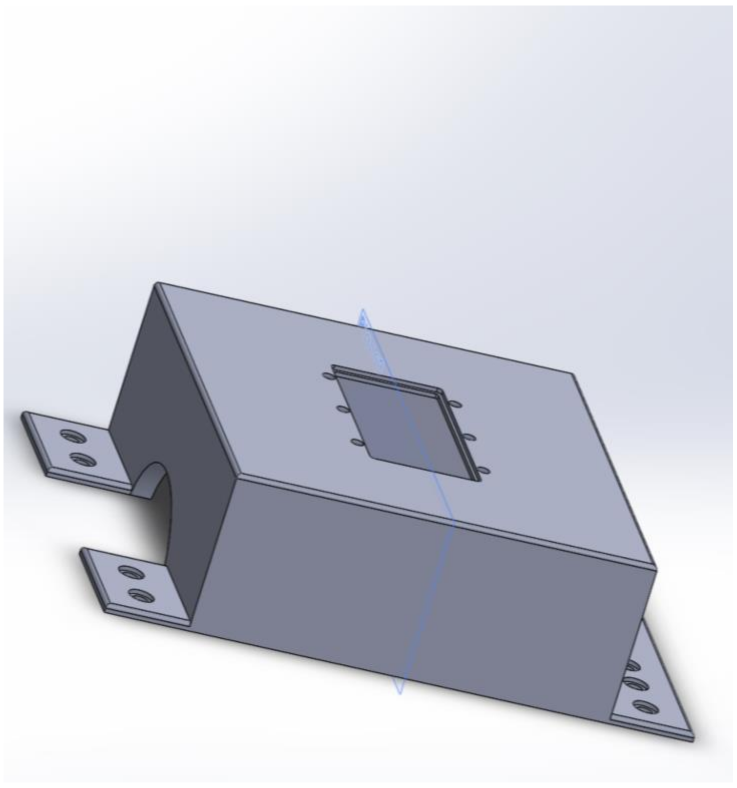


Figure 24 Prototype 1 -Base

The base was made using rectangular shapes, with a hole at the top for connecting the base motor and hole at the side to have it screwed to a surface to easy control (Figure 24).

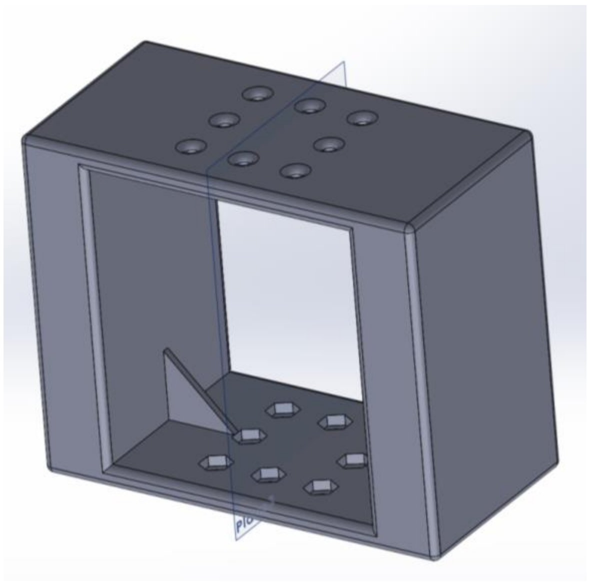


Figure 25 Prototype 1 - Joint 1

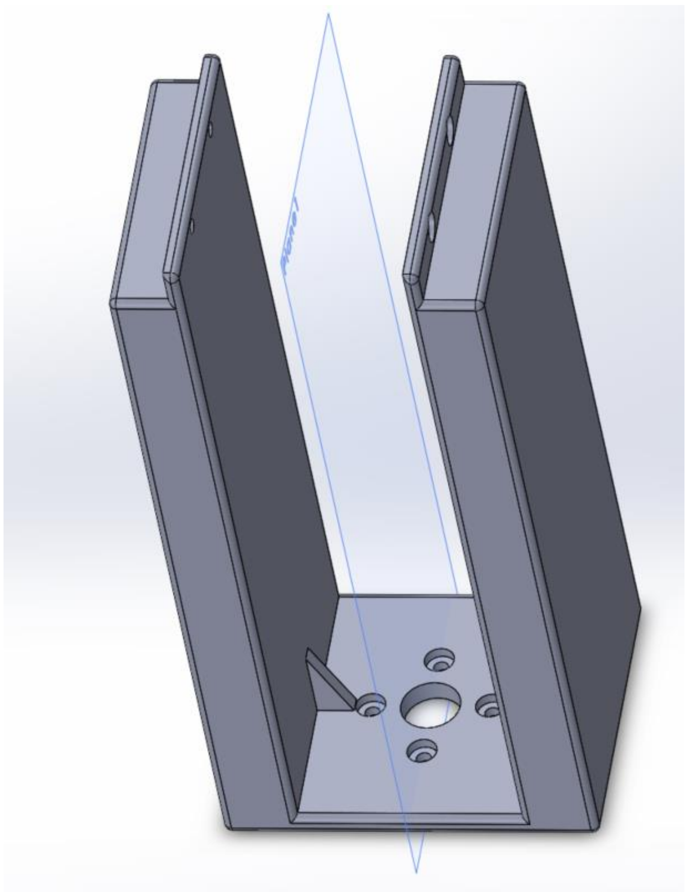


Figure 26 Prototype 1 - Joint 2

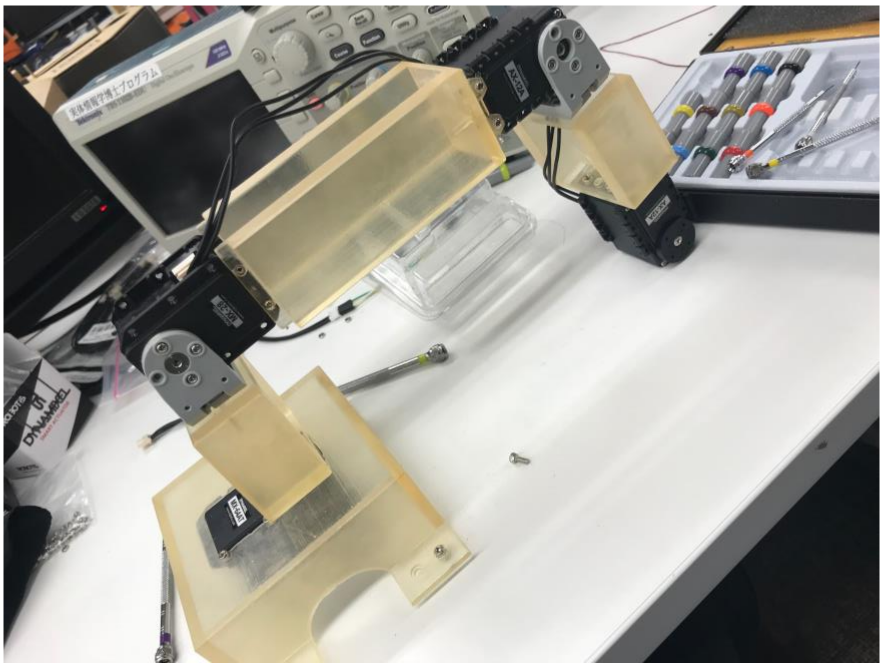


Figure 27 Prototype 1 - Assembly

All the joints have sizing and holes and carving suitable to their position in the arm to be connected to the motor as well as the screw size that being used with the motor provided by the manufacturer. The sizing was also made to go well with components provided by the company for easy future improvement. There were also small ribs to keep the joint sstronger.

As I am getting used to the software and the printing machine, I tried to make it as simple and identified as possible. The result was a success as each part fits perfectly to the motors using the screws that were provided along with the product.

However, the lack of experience with designing and lack of calculation on the material has led to several problems with the design. First of all, in an effort to compensate for the frailty of the material, which was silicone, all the links of this design were made to be thick enough so that breakage would not occur. However, the thickness causes two major problem: the torque needed on the second motor was too big, and the design is too hard to be remade into steel or similar materials later on as it is difficult to find or work with a piece of metal that thick. The length of each link creates a workspace that was less than expectation and it was decided that these problems have to be dealt with later on.

Despite the issues regarding the design, which come into light only later on in the development process, this initial prototype was the one used for developing a major part of the firmware.

3.2. Second prototype

After putting the model to test for a period of time, we have realized a huge problem concerning the thickness of the model, which make it hard for the motor to move slower with a smaller torque and also it is hard to nearly impossible to replace it with metal later on, since metal may be much heavier and it is easier to get metal with a smaller thickness.

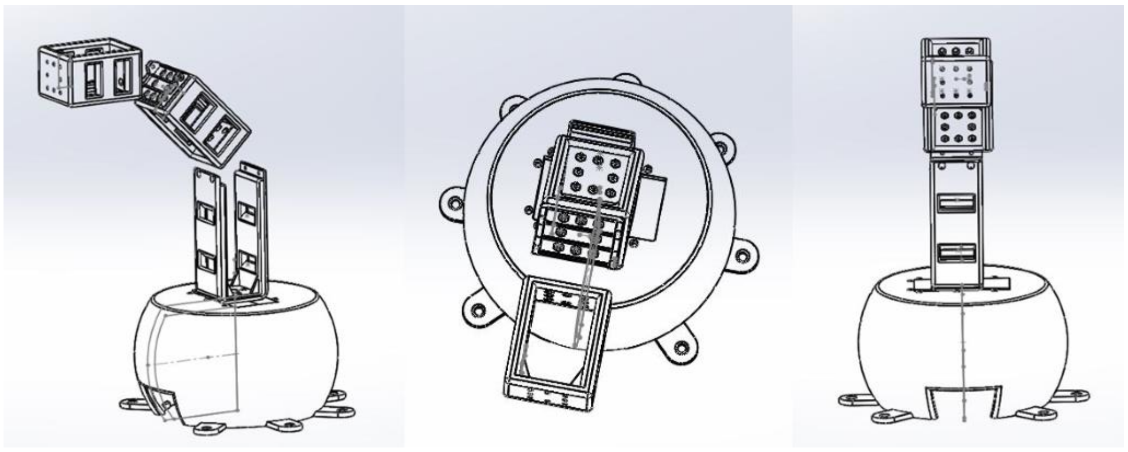


Figure 28 Model Assembly

Therefore, the second design's (Figure 29) thickness was heavily reduced.

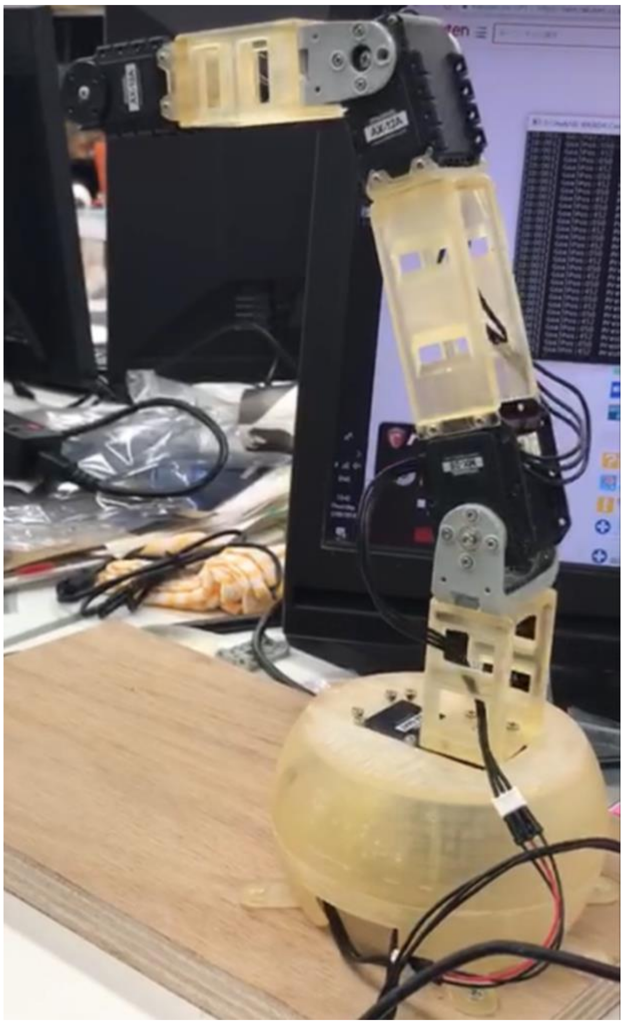


Figure 29 Prototype 2 – Assembly

To further decrease the weight of each joint, holes were carved into the material with reasonable sizing so that the joints still have sufficient strength while saving on material. The holes also act as the part making sure that the connecting wires are always kept in place and close to the arm

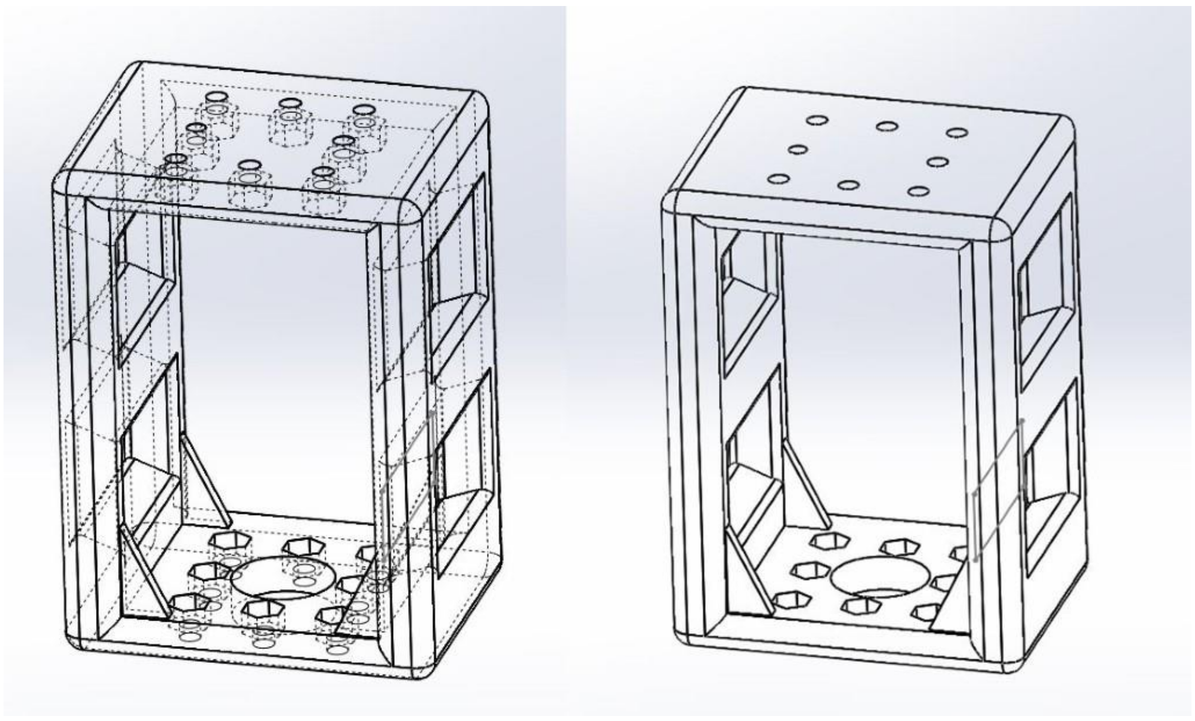


Figure 30 Prototype 2 - Joint 1

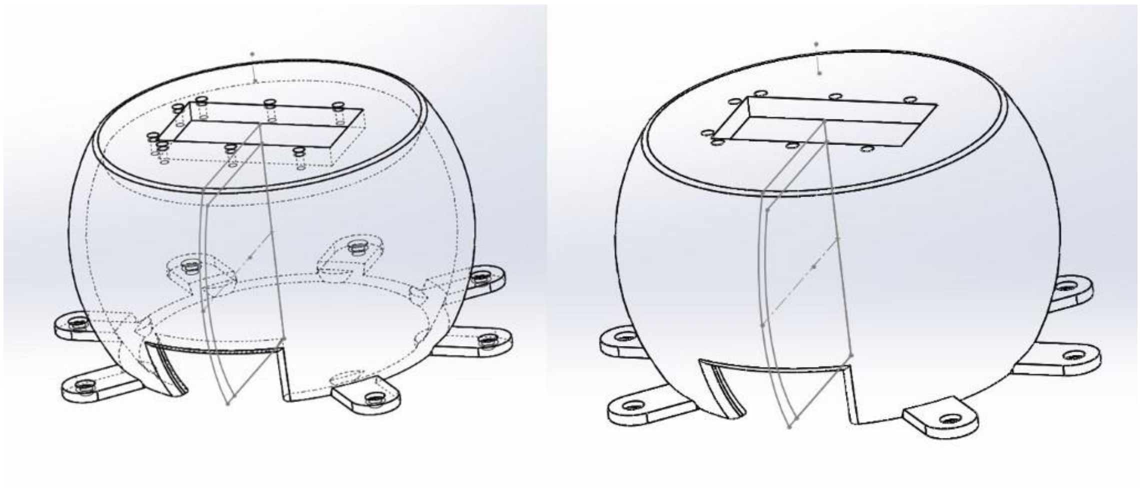


Figure 31 Prototype 2 - Base

The base was changed to a rounder design for aesthetic reason. It was also increased in height to keep all the components under. The seven holes around are for attaching the arm to a surface and a gate is for the power source that is going in.

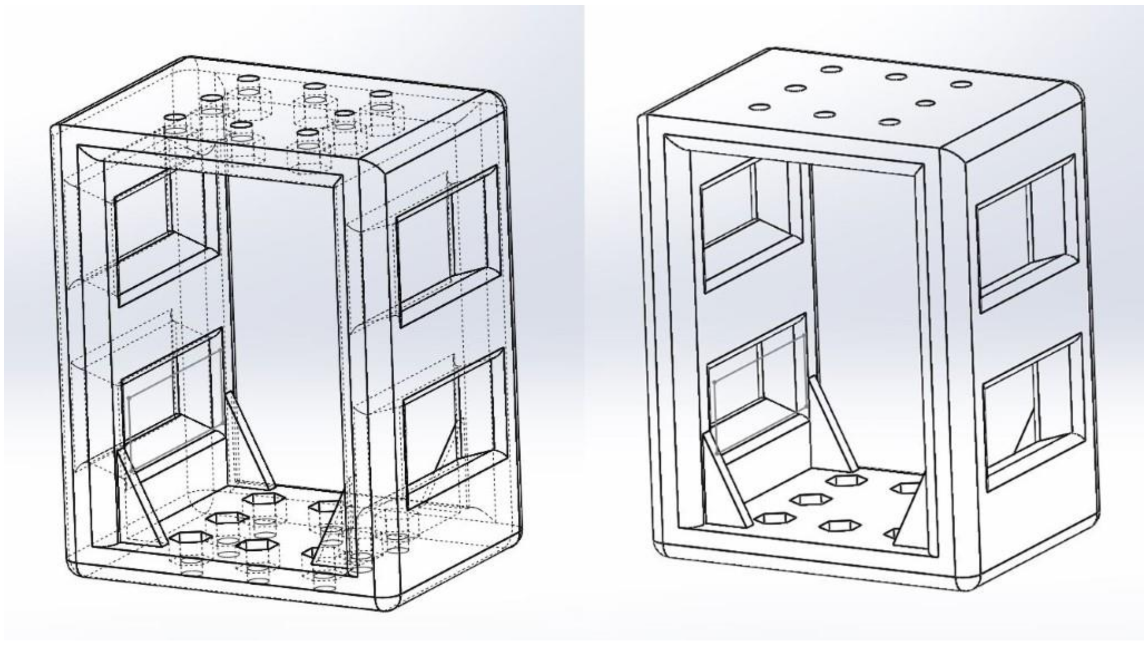


Figure 32 Prototype 2 - Joint 2

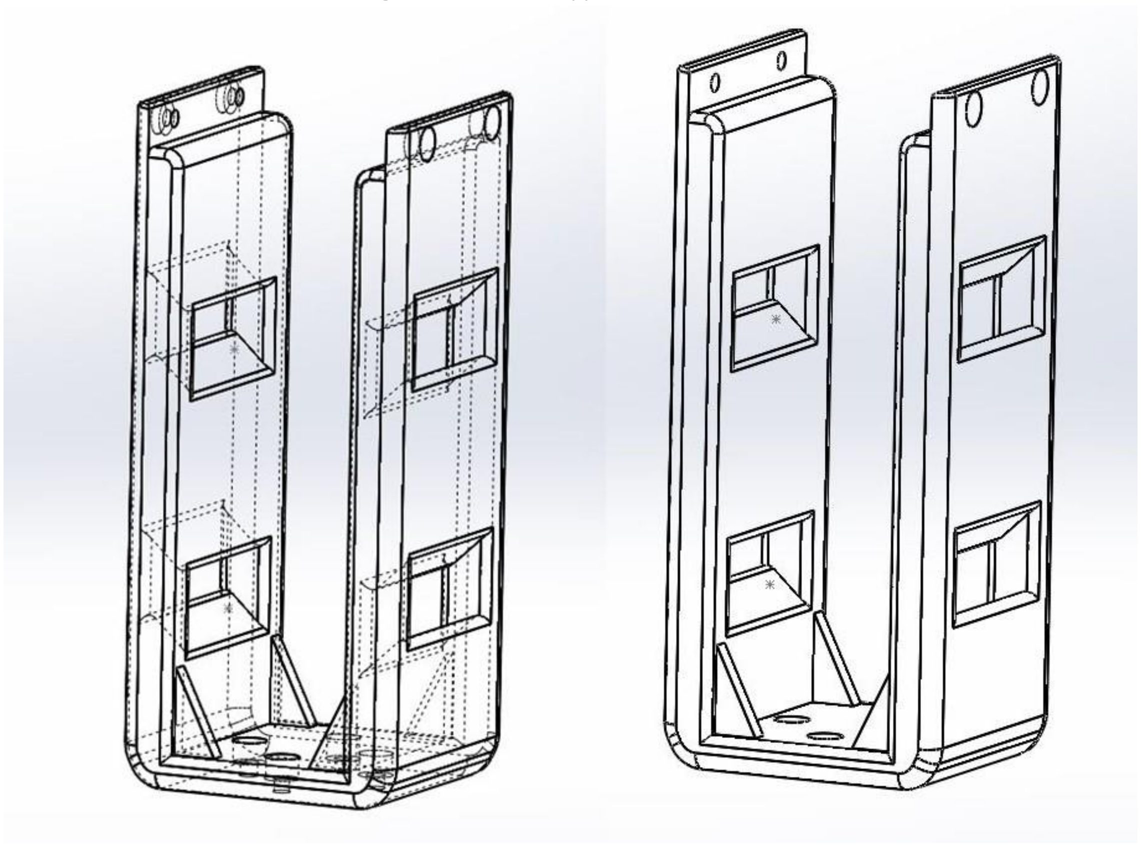


Figure 33 Prototype 2 - Joint 3

For most other parts, the design idea stays almost the same as in the previous design. This is also the last prototype to be printed. The joint were all made with carved holes, rounder

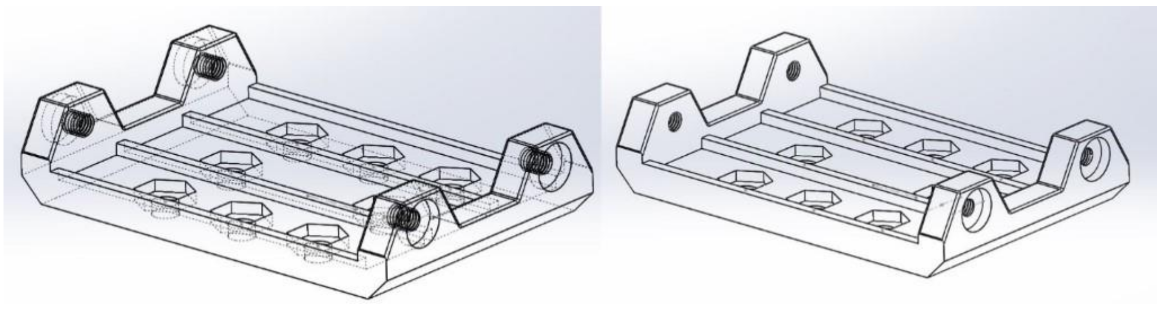


Figure 34 Prototype 2 - Connecting part

design and some changes was made to the sizing so that it fits better with the controlling. The new design is smaller, lighter, better-looking and easier to use.

3.3 Third prototype (unfinished)

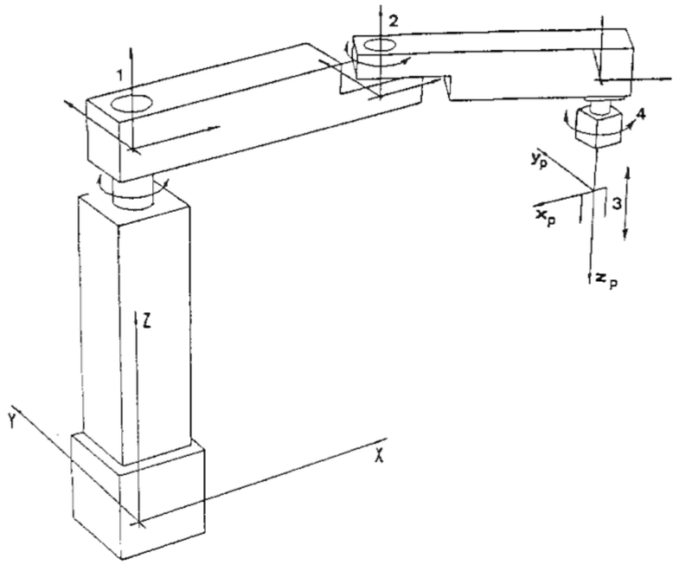


Figure 35 SCARA design

Concluding that the current design was not flexible enough as well as being inspired by the Japanese-born product MySpoon, it was decided to add one more DOF to the current design, making it 5 DOF. The design was called a “Scara Design”, based on the SCARA robot developed by professor Hiroshi Makino at University of Yamanashi in 1981. The design was considered a revolutionary work, presenting a completely new concepts for assembly robots and it became extremely popular.

Some of SCARA's signature points include: parallel-axis joint layout; compliant in X-Y direction by rigid in Z direction; the jointed two-link arm layout similar to human arms. All the feature allows for food retrieving operation, being able to move in confined area.

Some other advantages of this design also include clean and fast operation, small footprint with the ability to work in a floor place-limited environment and is easy to relocated the robot with an easy, unhindered form of mounting.

The beginning of the SCARA design stage were also the last few remaining days of my thesis working term. Therefore, I was only capable of making a 2D sketch and it was left for the next engineer to continue. A joint was already designed but in the end wasn't made into use, as seen below in Figure 36:

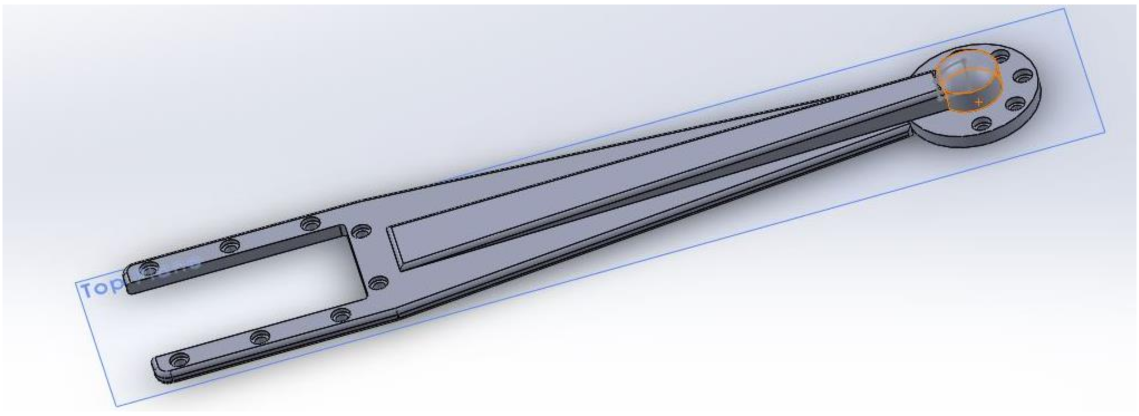


Figure 36 Prototype 3 - Joint

Regarding calculation and programming, the third prototype would not be in any mean considered. Readers who are looking at the following chapter should see the work applied for the first two prototypes.

4. MODEL ANALYSIS

4.1 Algebraic calculation

This section discusses the kinematics model of the arm, covering from each of the joint to the final total model. The calculation was made using matrix functions from MATLAB. This was made for testing against values, but was not for the calculation. Even though future project worker may benefit from this model, the control program was made according to the geometric solution.

According to DH convention that was discussed in chapter 2, the kinematic properties of each of the joint can be described as bellow with one matrix for each frame transformation:

$$ {}^{0}_{1}T = \begin{bmatrix} c\theta_1 & -s\theta_1 & 0 & 0 \\ s\theta_1 & c\theta_1 & 0 & 50 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} (3) $$

$$ {}^{1}_{2}T = \begin{bmatrix} c\theta_2 & -s\theta_1 & 0 & 65 \\ 0 & 0 & 1 & 0 \\ -s\theta_2 & -c\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} (4) $$

$$ {}^{2}_{3}T = \begin{bmatrix} c\theta_3 & -s\theta_3 & 0 & 185 \\ s\theta_3 & c\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} (5) $$

$$ {}^{3}_{4}T = \begin{bmatrix} c\theta_4 & -s\theta_4 & 0 & 115 \\ s\theta_4 & c\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} (6) $$

$${}^{4}_{E}T = \begin{bmatrix} 1 & 0 & 0 & 80 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} (7)$$

Using MATLAB for calculation, we achieve the following model as seen in Table 1:

Table 1 Forward Kinematics Model

i	α_i - 1	a_i-1	d_i-1	θ_i-1
1	0	0	50	θ_1
2	-90 deg	65	0	θ_2
3	0	185	0	θ_3
4	0	115	0	θ_4
E	0	80	0	0

The MATLAB code was as follow:

model.txt:

```
c4*(c1*c2*c3 - c1*s2*s3) - s4*(c1*c2*s3 + c1*c3*s2), -
c4*(c1*c2*s3 + c1*c3*s2) - s4*(c1*c2*c3 - c1*s2*s3), -s1, 65*c1 +
185*c1*c2 + 80*c4*(c1*c2*c3 - c1*s2*s3) - 80*s4*(c1*c2*s3 +
c1*c3*s2) + 115*c1*c2*c3 - 115*c1*s2*s3
c4*(c2*c3*s1 - s1*s2*s3) - s4*(c2*s1*s3 + c3*s1*s2), -
c4*(c2*s1*s3 + c3*s1*s2) - s4*(c2*c3*s1 - s1*s2*s3), c1, 65*s1 +
185*c2*s1 + 80*c4*(c2*c3*s1 - s1*s2*s3) - 80*s4*(c2*s1*s3 +
c3*s1*s2) + 115*c2*c3*s1 - 115*s1*s2*s3 + 50
- c4*(c2*s3 + c3*s2) - s4*(c2*c3 - s2*s3), s4*(c2*s3 + c3*s2) -
c4*(c2*c3 - s2*s3), 0, - 185*s2 - 115*c2*s3 - 115*c3*s2 -
80*c4*(c2*s3 + c3*s2) - 80*s4*(c2*c3 - s2*s3)
0, 0, 0, 1
```

Figure 37 The kinematic model in text

kinematics.m:

```
% fileID = fopen('model.txt','wt');
%Cosine and Sine value
syms c1 c2 c3 c4;
syms s1 s2 s3 s4;

%d and a value
d1 = 50;
a1 = 65;
a2 = 185;
a3 = 115;
a4 = 80;

%Kinematics Model
t01 = [c1 -s1 0 0 ; s1 c1 0 d1; 0 0 1 0; 0 0 0 1 ];
t12 = [c2 -s2 0 a1 ; 0 0 1 0; -s2 -c2 0 0; 0 0 0 1 ];
t23 = [c3 -s3 0 a2 ; s3 c3 0 0; 0 0 1 0; 0 0 0 1 ];
t34 = [c4 -s4 0 a3 ; s4 c4 0 0; 0 0 1 0; 0 0 0 1 ];
t4e = [1 0 0 a4 ; 0 1 0 0; 0 0 1 0; 0 0 0 1 ];

%Final forward kinematics transformation
t0e = t01*t12*t23*t34*t4e;
```

Figure 38 MATLAB Code for forward kinematics transformation

4.2 Geometric calculation

Fortunately, even though the arm has 4 DOFs, several axes are parallel to each other, thus making a geometric solution possible. In this subchapter, I will be describing how the calculation was made for the arm.

The problem can be summarized as:

- Given:

+ end effector orientation ϕ , as angle with regard to the plane center at O_3 and parallel to the ground plane

+ end effector position $[x_g, y_g, z_g]$

- Need:

+ all joint angle value including $\theta_1, \theta_2, \theta_3, \theta_4$

First, we will look at how the systems is viewed from the side (Figure 39).

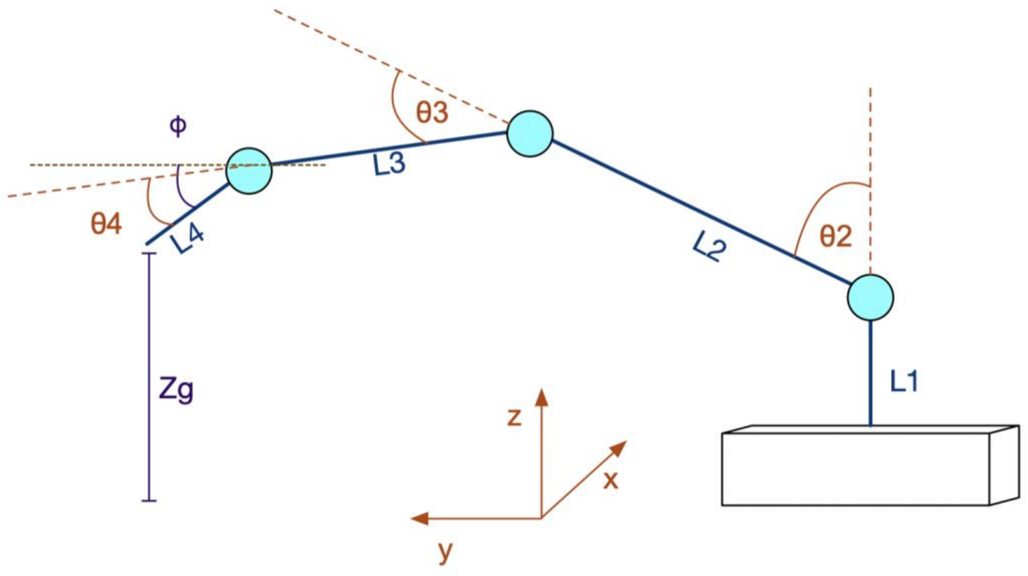


Figure 39 Side view graph

And how it looks from the top view (Figure 40):

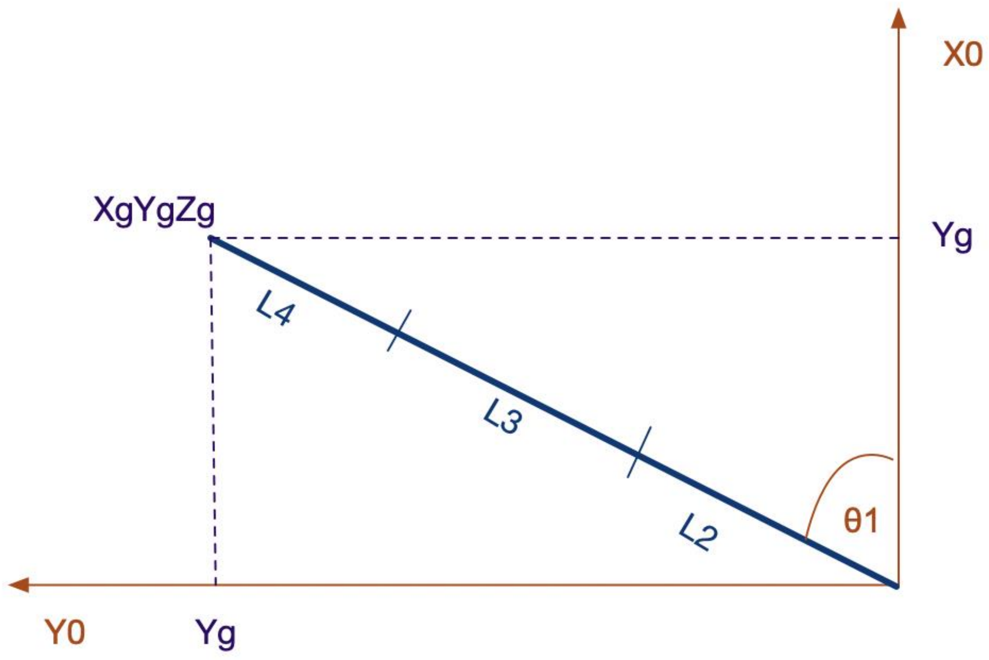


Figure 40 Top view graph

From the top view we get:

$$\theta_1 \quad = \quad atan2(y_g, x_g)(8)$$

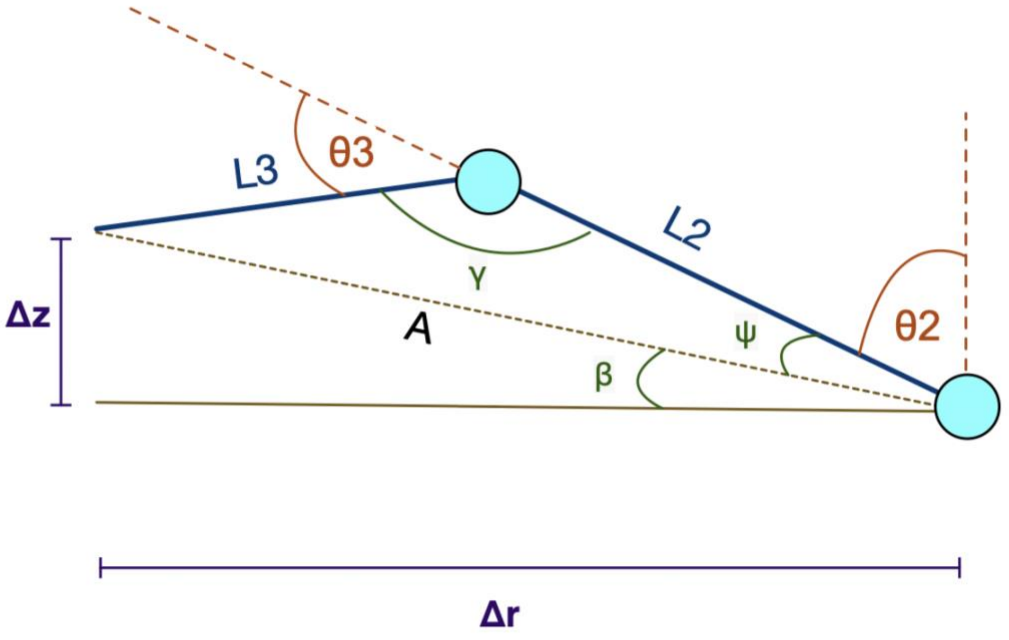


Figure 40 Side view of decoupled graph

Then from the sideview, using the decoupling technique (Figure 41), with which we separate the end-effector from the rest of the robot at the last joint and joint 1 from the rest of the robot.

Now we can solve for θ_3:

$$\left.\begin{array}{r} A = \sqrt{{\varDelta_z}^2 + {\varDelta_r}^2} \\ A^2 = L_2^2 + L_3^2 - 2L_2L_3cos\gamma \end{array}\right\}(9)$$

$$(9) \rightarrow \sqrt{\varDelta_z^2 + \varDelta_r^2} = \sqrt{L_2^2 + L_3^2 - 2L_2L_3cos(180^\circ - \theta_3)}(10)$$

$$\rightarrow \varDelta_z^2 + \varDelta_r^2 = L_2^2 + L_3^2 - 2L_2L_3cos(180^\circ - \theta_3)(11)$$

$$\rightarrow cos\theta_3 = \frac{\varDelta_x^2+\varDelta_r^2-L_2^2-L_3^3}{2L_2L_3}(12)$$

We can also get θ_2. Firstly:

$$\theta_2 = \begin{cases} \frac{\pi}{2} - \beta - \psi & if\theta_3 > 0, \\ \frac{\pi}{2} - \beta + \psi & if\theta_3 < 0. \end{cases}(13)$$

From the figure we can see:

$$\beta = atan2(\varDelta_z, \varDelta_r)(14)$$

Also:

$$L_3^2 = A^2 + L_2^2 - 2AL_2cos\psi(15)$$

$$\rightarrow cos\psi = \frac{A^2+L_2^2-L_3^2}{2AL_2}(16)$$

$$\rightarrow cos\psi = \frac{\varDelta_x^2+\varDelta_r^2+L_2^2-L_3^2}{2\sqrt{\varDelta_z^2+\varDelta_r^2}L_2}(17)$$

We then try to get θ_4. Putting back the end-effector to the decoupling, we get the following:

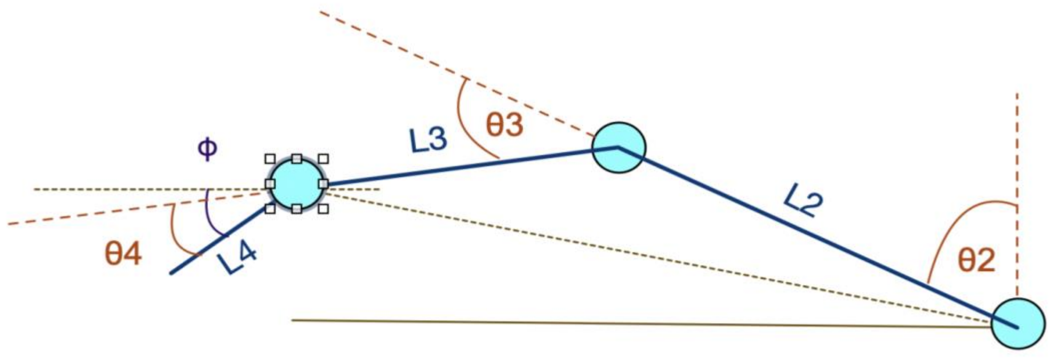


Figure 41 Side view decoupled with L4

From here, combining all joint so that they have the same starting point:

We can now calculate θ_4 as:

$$\theta_4 = \phi - \theta_2 - \theta_3 + 90° (18)$$

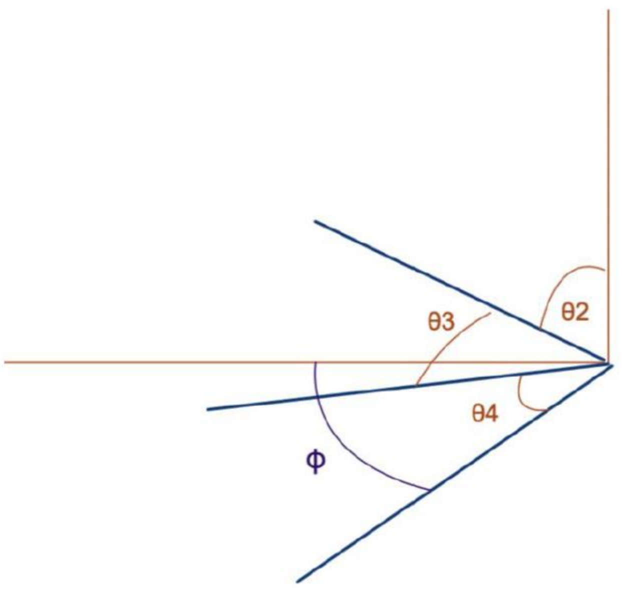


Figure 42 All joint degree

5. SOFTWARE IMPLEMENTATION

5.1 General Description

The idea behind the robot's prototype is rather easy to understand: Once it is powered up, it would tell the user that it is ready and once the user clicks a button, the torque on all motors loosen up just enough so that the user can manually adjust the tool (in our case, a spoon or chopsticks) to where the user's mouth is at. Then the user clicks the same button again to lock the position. With a second button, the arm will go back and forward between the mouth and the plate to carry food, and a click again at the first button would bring back to the adjustment mode to adjust the position of the mouth, if needed. The software implementation aimed to cover all of the most basic functions needed for future development with the arm, including:

- Moving the arm by joint degree value
- Calculating the degree value needed for each arm to reach the desired position in the XYZ plane
- Constant logging of calculation, moving, adjustment, communication activities and error into a log file as well as printing it to the screen
- Loosening the grip of the motors for manual adjusting using hands
- Initializing and control the motor
- Adjusting the speed of each motor by changing the torque

Most of the code written for this project was made in C++ using the Dynamixel SDK designed to work with the motors provided by ROBOTIS. The version being used at the time was 3.3.

5.2 Class Diagram

The following class diagram describes how all the functions are interconnected to each other. The log functions were aimed to be used by every available and future functions as it is the one that reports the status of the systems.

Each functions would be further discussed later.

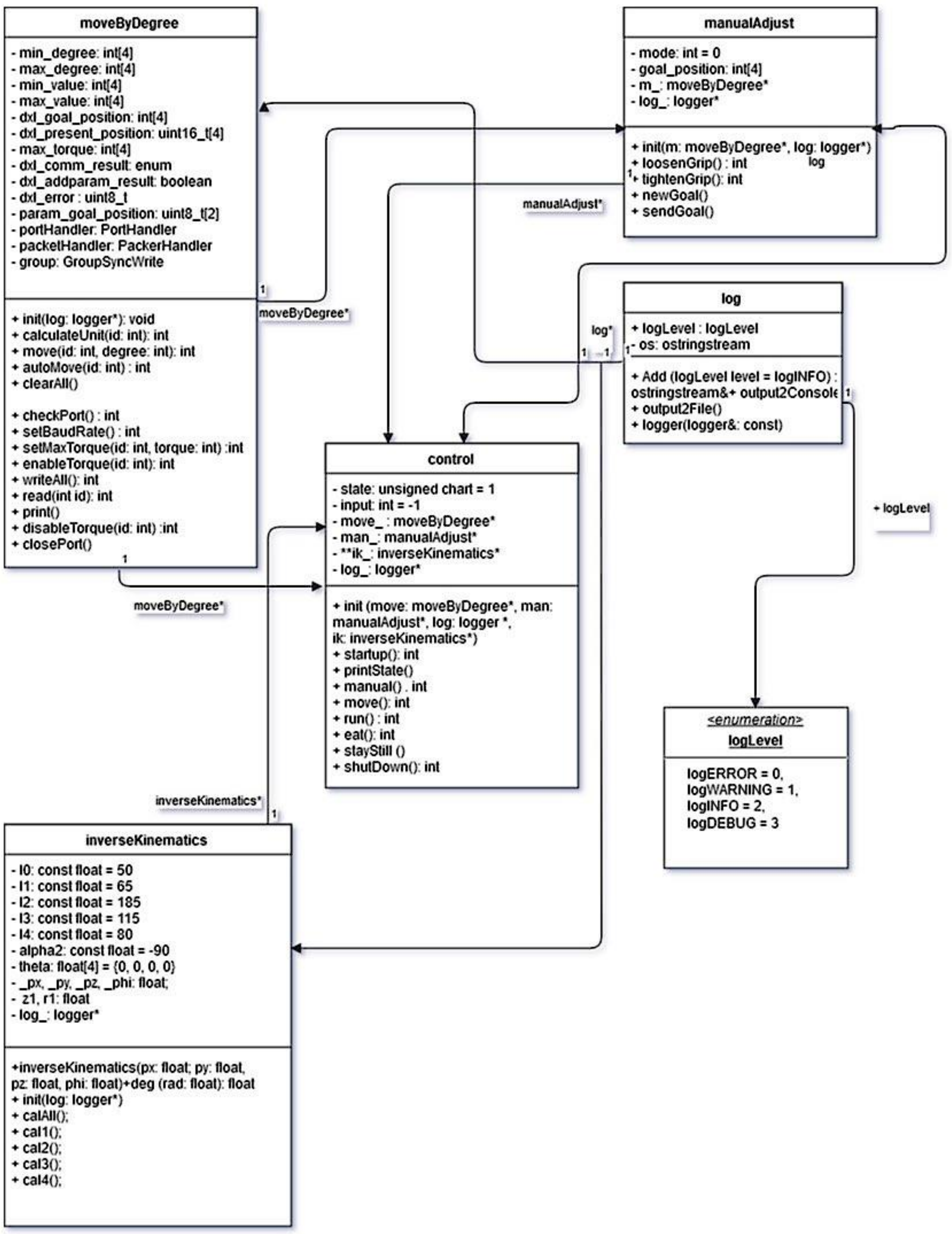


Figure 43 Class Diagram

5.3 Sequence Diagram

The following sequence diagram illustrates how each functions participates in a regular usage cycle.

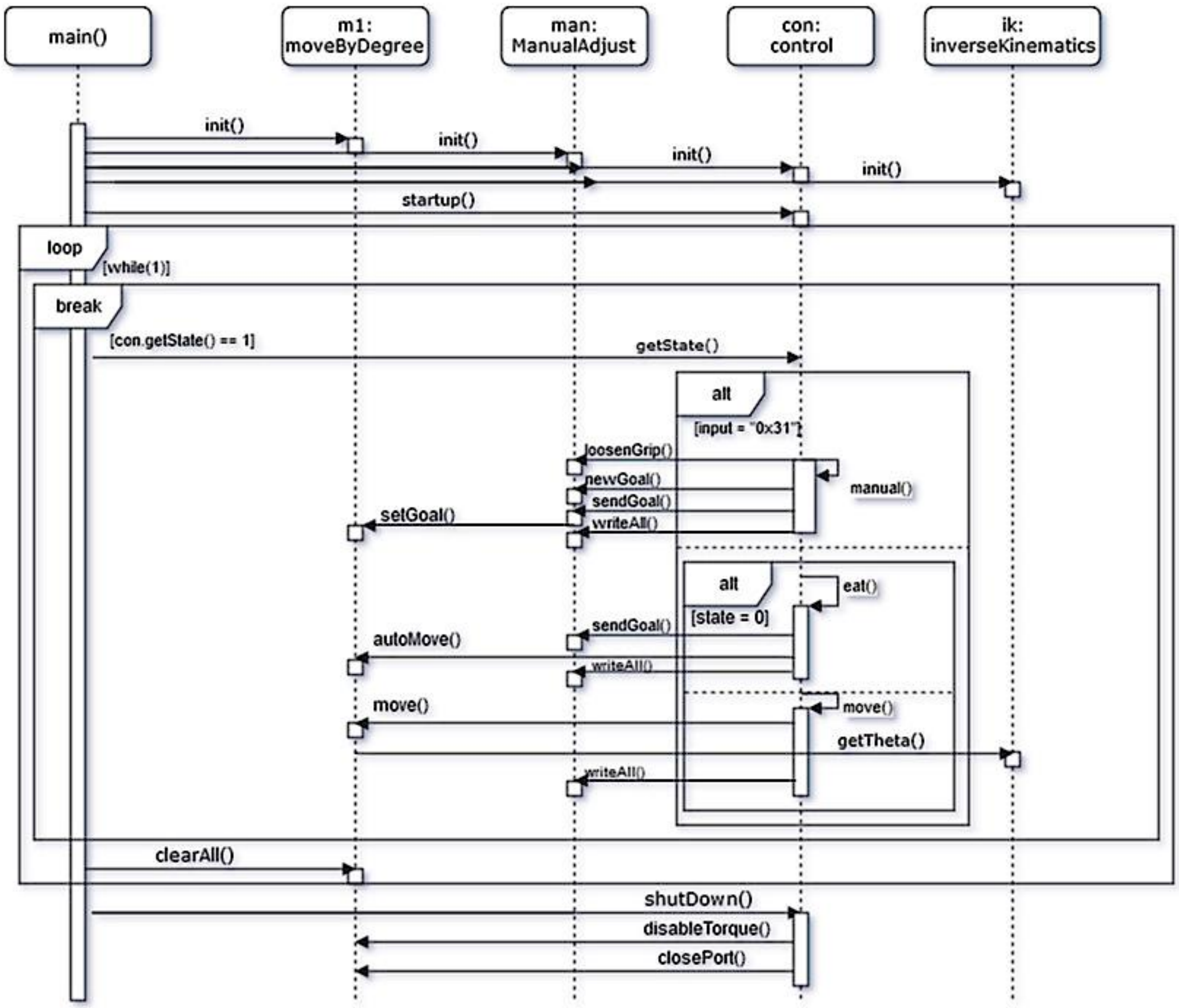


Figure 44 Sequence Diagram

First, for any function to be used at all, it needs to be initialized using the **init()** method. Therefore, the **main()** function calls all **init()** method at the beginning. Afterward, for torque to be applied to the motors so that the arm can erect, the **control** class's method **startup()** must be called. Afterward, the functions goes into an infinite loop with **while(1)** until the device is "shut off" by pressing 'ESC'.

At the beginning of each loop, the program check using **con.getState()** (‘con’ is for ‘control’, because this method belongs to the **control** class) to see if the ‘ESC’ button is being pressed. If not, the program may continue.

If it is in setup state, then the program will loosen the grip of all motors to allow manual adjustment. Then, the new goal position is recorded, sent to the motor and write the data to the proper data address inside the motors.

On the other hand, if the robot arm is in moving mode, by clicking one button, the arm will go back and forward between the mouth position and the plate position, unless told to change the goal position or the whole device is shutdown.

Lastly, when receiving the command to shut down, the program will clear all the location data stored, disable torque at all the motors and close all transmission ports between the computer and the motors.

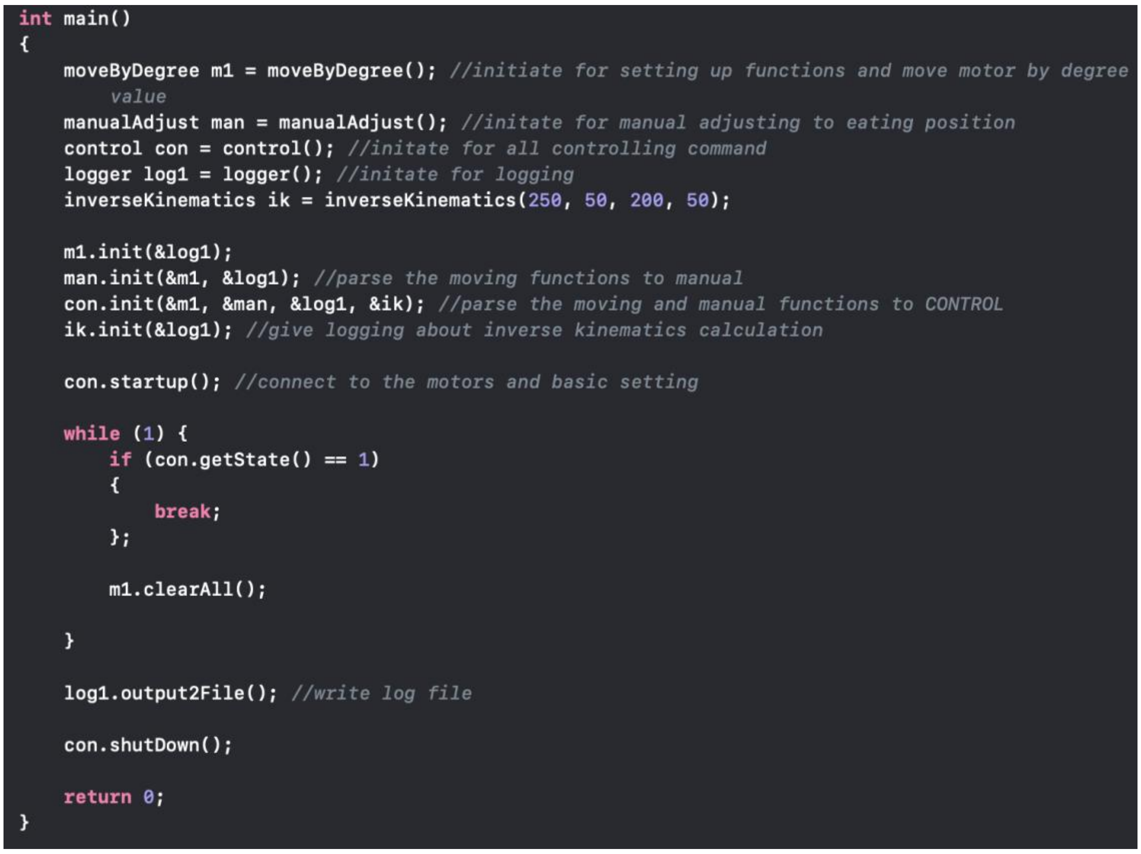
5.4 Function Details

In this section, the methods and logic behind each function that comply to the arm’s functionalities will be discussed. One special class **logger** which is used for logging all the data will be discussed separately in the next chapter along with all the error that could occur within the program.

For all systems related functions, a version for Linux OS was also included.

5.4.1 main() Function

The **main()** function (Figure 46) does the initialization of each needed classes, puts in the value for the location of the plate then runs the loop. Until the loop is finished by shutting down the device, the arm does the movement and constantly checks if the state is being changed. Lastly, after all has been done, the program writes everything to a log file and shutdown() all motors as well as all the connections.

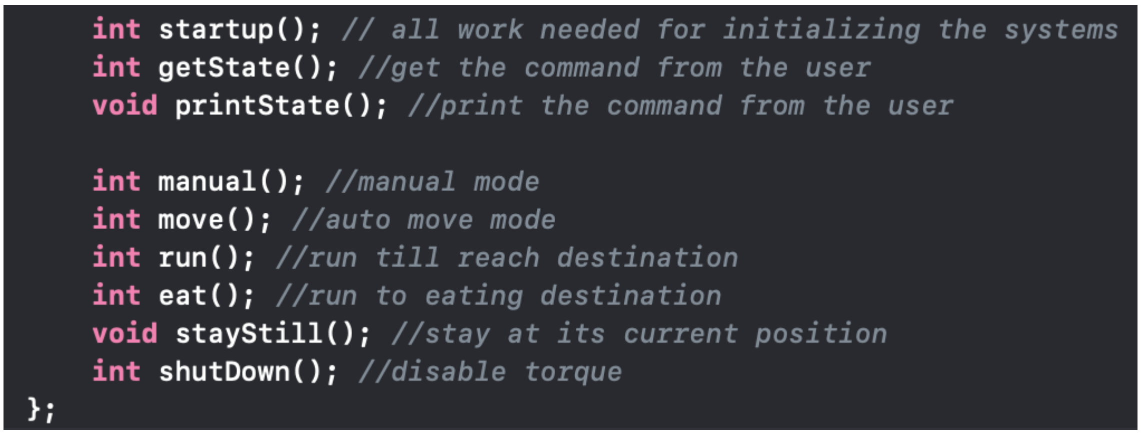


```cpp
int main()
{
    moveByDegree m1 = moveByDegree(); //initiate for setting up functions and move motor by degree
        value
    manualAdjust man = manualAdjust(); //initate for manual adjusting to eating position
    control con = control(); //initate for all controlling command
    logger log1 = logger(); //initate for logging
    inverseKinematics ik = inverseKinematics(250, 50, 200, 50);

    m1.init(&log1);
    man.init(&m1, &log1); //parse the moving functions to manual
    con.init(&m1, &man, &log1, &ik); //parse the moving and manual functions to CONTROL
    ik.init(&log1); //give logging about inverse kinematics calculation

    con.startup(); //connect to the motors and basic setting

    while (1) {
        if (con.getState() == 1)
        {
            break;
        };

        m1.clearAll();

    }

    log1.output2File(); //write log file

    con.shutDown();

    return 0;
}
```

Figure 45 main() function

5.4.2 control() Class

The **control()** class (Figure 47) contains the following methods:



```cpp
    int startup(); // all work needed for initializing the systems
    int getState(); //get the command from the user
    void printState(); //print the command from the user

    int manual(); //manual mode
    int move(); //auto move mode
    int run(); //run till reach destination
    int eat(); //run to eating destination
    void stayStill(); //stay at its current position
    int shutDown(); //disable torque
};
```

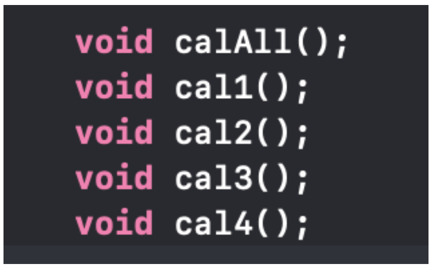
Figure 46 control class methods

This is considered to be the heart of the system. It contains all the functions relating to the movement of the arm and it is used by all other classes. **control** class acts as a middle man to put all the functions into places which help keep the main function as well as all other functions controlling the motors clean and readable.

- **startup()**: it calls all method that is needed to get the motors starting
- **getState()**: checks the user's command at the beginning of every loop, to see if they want to quit the program, continue running or change the goal position
- **manual()**: calls the function from class **manual** like loosening the grip, set and send the new goal.
- **move():** automatically moves to the plate position while calling **run()**
- **eat()**: automatically moves to the mouth position while calling **run()**
- **run()**: tells the motors to read and move according to the location data being stored
- **stayStill()**: stays at the curresnt location while keeping the torque
- **shutDown()**: disables torque of all the motors and close all communication port

5.4.3 inverseKinematics() Class

The class contains just five methods (Figure 48):



```
void calAll();
void cal1();
void cal2();
void cal3();
void cal4();
```

Figure 47 inverseKinematics() class method

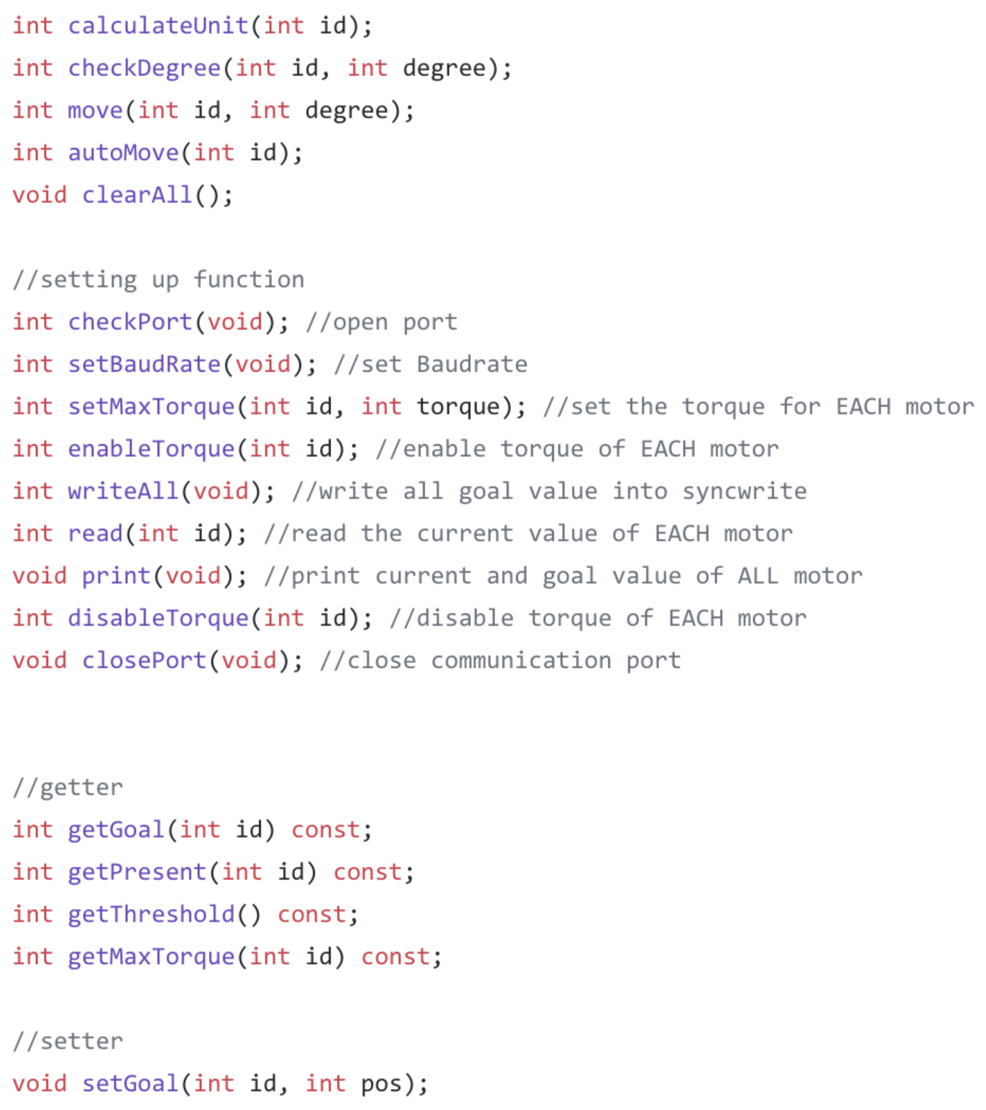
- **calAll()**: call all the other ‘cal’ methods
- **cal1()-cal4()**: calculate the degree for each joint using the axis data given by the user

5.4.4 moveByDegree() Class

moveByDegree() represents all communication between the firmware and the motors. It works with the most basic functionality and all the functions provided by Dynamixel SDK. All the method built to this class include:

- **calculateUnit()**: calculates from degree values of the joints into motor position value
- **checkDegree()**: checks if the input degree is within the range of the arm
- **move()**: sends signal for the motor to move after calling **checkDegree()**
- **autoMove():** sends motor signal to move to the location that is currently being stored
- **clearAll()**: clears all data that is being written to the motors
- **checkPort()**: checks if the connections to all the port on motors are fine

- **setBaudRate(), setMaxTorque()**: sets the baud rate and max torque value on the motors



```
int calculateUnit(int id);
int checkDegree(int id, int degree);
int move(int id, int degree);
int autoMove(int id);
void clearAll();

//setting up function
int checkPort(void); //open port
int setBaudRate(void); //set Baudrate
int setMaxTorque(int id, int torque); //set the torque for EACH motor
int enableTorque(int id); //enable torque of EACH motor
int writeAll(void); //write all goal value into syncwrite
int read(int id); //read the current value of EACH motor
void print(void); //print current and goal value of ALL motor
int disableTorque(int id); //disable torque of EACH motor
void closePort(void); //close communication port


//getter
int getGoal(int id) const;
int getPresent(int id) const;
int getThreshold() const;
int getMaxTorque(int id) const;

//setter
void setGoal(int id, int pos);
```

Figure 48 moveByDegree() class methods

- **enableTorque()**: applies torque on all motors

- **writeAll()**: writes all goal location data to the motors synchronously

- **read()**: reads the current location of the motors

- **print()**: prints current and goal data to the console

- **disableTorque:** stops applying torque on all motors by calling and putting 0 to **setMaxTorque()**

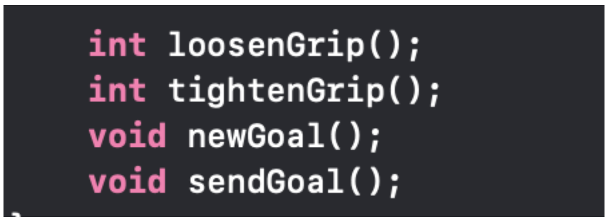
- **closePort()**: stops all communication between the program and the motors

- **getGoal(), getPresent(), getThreshold(), getMaxTorque:** getter functions for the other class to get the data they needed of the motors

- **setGoal()**: mainly used my **manualAdjust()** class, this function lets the other classes' method set the goal position to the motor

5.4.5 manualAdjust() class

The **manualAdjust()** class manages all functions needed when the users need to change the eating position that is stored in the device.



```
    int loosenGrip();
    int tightenGrip();
    void newGoal();
    void sendGoal();
```

Figure 49 manualAdjust() class methods

- **loosenGrip()**: decreases each motor's torque to a reasonable values so it can be moved around by the users

- **tightenGrip()**: sets the maximum torque of each motor back to the default values

- **newGoal()**: gets the current location of the arm (after having been moved by the users) as the goal location

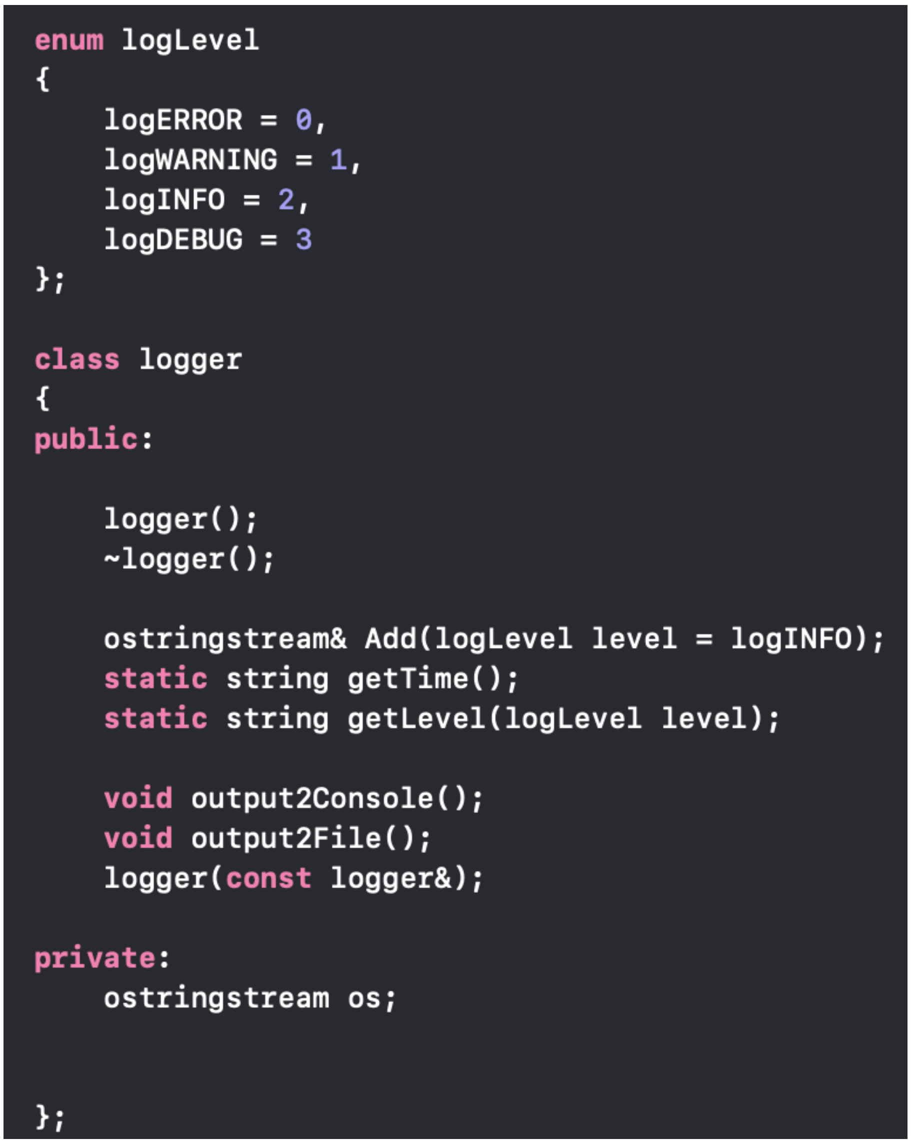
- **sendGoal()**: sends the location gotten from the users to **moveByDegree** class

6. LOGGING AND IMPORTANT ERRORS

6.1 logger() class

In the application, ostream is used to gather all the problem, status and information regarding the device and print them out at once to the log file when the program finishes.

The class's structure is illustrated as in Figure 51:



```
enum logLevel
{
    logERROR = 0,
    logWARNING = 1,
    logINFO = 2,
    logDEBUG = 3
};

class logger
{
public:

    logger();
    ~logger();

    ostringstream& Add(logLevel level = logINFO);
    static string getTime();
    static string getLevel(logLevel level);

    void output2Console();
    void output2File();
    logger(const logger&);

private:
    ostringstream os;


};
```

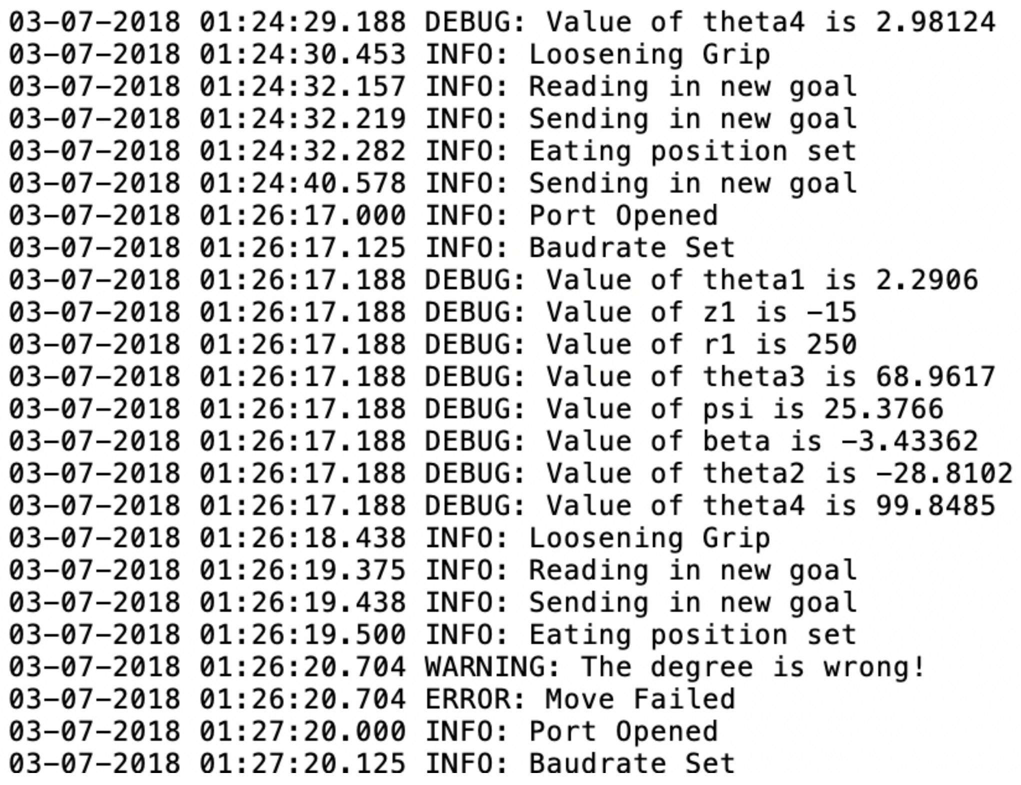
Figure 50 logger() class methods

There are 4 logging level for errors, warnings, information and extra for debugging purposes. An ostringstream named os carries all the data and is flushed out during the end of the program.

The methods within the class includes:

- **Add():** this method is used by all other classes to add the logging data onto the stream
- **getTime():** the functions get the current time of the computer and put the form “day-month-year hour:minute:second”
- **getLevel()**: returns the level of the log
- **output2Console()**: prints the stream to the console
- **output2File()**: prints the stream to the log file

An example log file made by the function can be seen below:



```
03-07-2018 01:24:29.188 DEBUG: Value of theta4 is 2.98124
03-07-2018 01:24:30.453 INFO: Loosening Grip
03-07-2018 01:24:32.157 INFO: Reading in new goal
03-07-2018 01:24:32.219 INFO: Sending in new goal
03-07-2018 01:24:32.282 INFO: Eating position set
03-07-2018 01:24:40.578 INFO: Sending in new goal
03-07-2018 01:26:17.000 INFO: Port Opened
03-07-2018 01:26:17.125 INFO: Baudrate Set
03-07-2018 01:26:17.188 DEBUG: Value of theta1 is 2.2906
03-07-2018 01:26:17.188 DEBUG: Value of z1 is -15
03-07-2018 01:26:17.188 DEBUG: Value of r1 is 250
03-07-2018 01:26:17.188 DEBUG: Value of theta3 is 68.9617
03-07-2018 01:26:17.188 DEBUG: Value of psi is 25.3766
03-07-2018 01:26:17.188 DEBUG: Value of beta is -3.43362
03-07-2018 01:26:17.188 DEBUG: Value of theta2 is -28.8102
03-07-2018 01:26:17.188 DEBUG: Value of theta4 is 99.8485
03-07-2018 01:26:18.438 INFO: Loosening Grip
03-07-2018 01:26:19.375 INFO: Reading in new goal
03-07-2018 01:26:19.438 INFO: Sending in new goal
03-07-2018 01:26:19.500 INFO: Eating position set
03-07-2018 01:26:20.704 WARNING: The degree is wrong!
03-07-2018 01:26:20.704 ERROR: Move Failed
03-07-2018 01:27:20.000 INFO: Port Opened
03-07-2018 01:27:20.125 INFO: Baudrate Set
```

Figure 51 Logging File

6.2 Important Errors

In the current section, we discussed the error that can be detected by the application, as well as possible solutions:

+ “The degree is wrong”: degree value given is out of the device range, should check on whether the xyz position is possible

+ “groupSyncWrite addparam failed”: the Dynamixel groupSyncWrite functions failed, should check on the move() or automove() funtionc

+ “Failed to open the port”: the port to the motors cannot be connected, should check on hardware connection

7. CONCLUSION AND POSSIBILITY FOR FURTHER IMPROVEMENTS

The goal of this thesis project was to develop as much as possible in a period of nearly four months a feeding robot arm aiding elderly people with their daily meals. Covering from design to 3D printing and programming as well as documentation, this project definitely cover a very wide range, but not without a clear purpose.

From personal point of view as well as regarding comments from the supervisor, the end results were much more than what was expected in the first place. We have concluded two functional prototypes, an API for future development on both Windows and Linux systems as well as much official documentations. The most challenging phase was definitely the designing part since I was just beginning to get used to the drawing software and was not familiar with the calculation that goes behind a mechanical design. Despite this, in the end, the design was accepted and sent to the patent office in China and Japan for registration of the product.

During the course of the project, I have definitely learnt a lot especially about the topic of robotics as well as much further improved my C++ skills. I have also got the rare opportunity to work with an industry-quality 3D printing device. Overall speaking, the experience that I got during my training and thesis period was definite treasurable.

Other than what has been achieved, there are a number of improvements that I look forward to seeing in the project, all of which has been considered and discussed during my time at the university but I did not find time to tackle:

- A change of design from 4 DOFs to 5 DOFs SCARA if possible and without an expense problem since an extra degree of freedom can really make a huge difference for the arm’s flexibility

- A functions for finding the food in the plate using vision; this was one of the ideas that sparked during my working period, but was not yet implemented during my time

- A function for the spoon to move and scoop the food; during the development phase I was aiming for a program which would have the spoon move horizontally along the plate and when the users find that it is at the right position, they can click the control button again for it to go horizontally and get the food.

- Improvement on the material; reasonably, the marketable products to be made should be made with metal; during my work, we have already chosen by buy sample material but implementation is yet to be made.
- A change of operating systems, though the program was made to run in Windows and Ubuntu, it is very vital to make the systems so that I can work in a real-time operating systems for optimized performance; this may require that we change the motor systems entirely but it a step that we cannot do the robot arm without.
- Utilising a custom-made control algorithm, the work done only use the default control algorithm embedded in the motors, but I find it’s important that we have extra closed-loop control on our own to improve the quality of the movements.

8. REFERENCES

/1/ Introduction to Robotics Mechanics and Control 3rd Edition - John J. Craig

/2/ Modern Robotics - Kevin M. Lynch and Frank C. Park

/3/ Wikipedia page about atan2

https://en.wikipedia.org/wiki/Atan2

/4/ Wikipedia page about SCARA

https://en.wikipedia.org/wiki/SCARA

/5/ Tutorial on logging in C - Petru Marginean

http://www.drdobbs.com/cpp/logging-in-c/201804215#disqus_thread

/6/ Dynamixel SDK e-Manual

http://emanual.robotis.com/docs/en/software/dynamixel/dynamixel_sdk/overview/

/7/ Inverse Kinematics slides - Prof. Alessandro Dre Luca

http://www.diag.uniroma1.it/~deluca/rob1_en/10_InverseKinematics.pdf

/8/ Wikipedia page on Microsoft Visual Studio

https://en.wikipedia.org/wiki/Microsoft_Visual_Studio

/9/ Wikipedia page on SolidWorks

https://en.wikipedia.org/wiki/SolidWorks

/10/ Obi product page

https://meetobi.com/

/11/ Bestic product page

https://www.camanio.com/sv/products/bestic/

/12/ MySpoon product page

https://www.secom.co.jp/english/myspoon/

/13/ ROBOTIS website

http://www.robotis.us/